Running LQ Robust Regulator Problems in Dynare

Prepared by the Artificial "Evil Agent"

February 2010 - Preliminary Version

1 Goal of this Document

This document shows how to formulate Linear Quadratic (LQ) problems that feature a decision maker that fears he or she might use a misspecified model of the economy. We borrow the LQ environment from Hansen & Sargent (Recursive Models of Dynamic Linear Economies) and present matlab files that take as input the matrices governing the economic environment in Hansen & Sargent and use the Dynare package to compute equilibrium outcomes and possibly estimate model parameters. Finally, we illustrate the matlab code using as an example the economy described by Hansen, Sargent & Tallarini (1999).

2 Hansen Sargent LQ Setup

This section describes the standard LQ setup that we borrow from Hansen & Sargent.

$$\max_{\{c_t,k_t\}} -(1/2)\sum_{t=0}^{\infty}\beta^t\left[(s_t-b_t)\cdot(s_t-b_t)+(g_t)\cdot(g_t)\right]$$

subject to the following set of restrictions:

$$\begin{aligned}\Phi_c c_t+\Phi_g g_t+\Phi_i i_t &= \Gamma k_{t-1}+d_t\\ k_t &= \Delta_k k_{t-1}+\Theta_k i_t\\ h_t &= \Delta_h h_{t-1}+\Theta_h c_t\\ s_t &= \Lambda h_{t-1}+\Pi c_t\end{aligned}$$

The exact interpretation of these equations will depend on the economic problem studied, but roughly speaking, the terms involved are: s_t is a variable that maps consumption units today, c_t, and a stock of habits, h_t, into utilities, while b_t is a shock affecting utility. g_t scales the utility, and may be seen as lump-sum transfers. The first restriction is a resource constraint. The second is the evolution of stocks of capital and the third is the process for the formation of habits (but these may also be seen as the accumulation of consumer durable after a purchase c_t).

Additionally, the random variables evolve according to:

$$z_{t+1} = A_{22} z_t + C_2 w_{t+1}, b_t = U_b z_t \text{ and } d_t = U_d z_t$$

w_{t+1} is Gaussian vector with identity covariance matrix.

The agent that is maximizing these equations chooses a sequence c_t, i_t, k_t, h_t, g_t contingent on the state k_{t-1}, h_{t-1}, z_t. Given the convexity of the objective function, we can setup the lagrangian and it will yield a set of first order necessary conditions (FONCs). The lagrangian will therefore be:

$$\begin{aligned}
&-(1/2) E\{\sum_{t=0}^{\infty} \beta^t [(s_t - b_t)\cdot(s_t - b_t) + (g_t)\cdot(g_t)] + ... \\
&+M_t^{d\prime} [\Phi_c c_t + \Phi_g g_t + \Phi_i i_t - \Gamma k_{t-1} - d_t] ... \\
&+M_t^{k\prime} [k_t - \Delta_k k_{t-1} - \Theta_k i_t] ... \\
&+M_t^{h\prime} [h_t - \Delta_h h_{t-1} - \Theta_h c_t] ... \\
&+M_t^{s\prime} [s_t - \Lambda h_{t-1} - \Pi c_t]\}
\end{aligned}$$

3 First Order Conditions

The FONCs for the problem, found in system 4.2.4 of Hansen and Sargent are:

$$
\begin{aligned}
(c_t) &: -M_t^{d\prime}\Phi_c + M_t^{h\prime}\Theta_h + M_t^{s\prime}\Pi = 0 \\
(g_t) &: g_t - \Phi_g M_t^d = 0 \\
(h_t) &: -M_t^{h\prime} + \beta E\left[M_{t+1}^{h\prime}\Delta_h + M_{t+1}^{s\prime}\Lambda\right] = 0 \\
(i_t) &: -M_t^{d\prime}\Phi_i + M_t^{k\prime}\Theta_i = 0 \\
(k_t) &: -M_t^{k\prime}\Delta_k + \beta E\left[M_{t+1}^{d\prime}\Gamma + M_{t+1}^{k\prime}\Delta_k\right] = 0 \\
(s_t) &: -(s_t - b_t) + M_t^s = 0
\end{aligned}
$$

and it represents a system of linear difference equations of order n, where the order depends on the nature of the problem studied. The reason why the derivative of the squared terms does not have a 2 in front is because we premultiplied the objective by 1/2. You may also note that the second and sixth blocks have the matrixes transposed. This is a result of taking the derivatives of a variable that appears in the objective function, which yields the vector itself, and in the restriction which yields the j-th variable, the sum of the j-th row of whatever matrix they multiply and the j-th multiplier. This equations together with the restrictions determine the solution:

$$
\begin{aligned}
\Phi_c c_t + \Phi_g g_t + \Phi_i i_t &= \Gamma k_{t-1} + d_t \\
k_t &= \Delta_k k_{t-1} + \Theta_k i_t \\
h_t &= \Delta_h h_{t-1} + \Theta_h c_t \\
s_t &= \Lambda h_{t-1} + \Pi c_t \\
b_t &= U_b z_t \\
d_t &= U_d z_t
\end{aligned}
$$

4 LQ Robustness Setup

The problem of a decision maker that fears model misspecification for the same economic problem has a similar construction. It is described by a similar setup in which to agents play

the following game:

$$\min_{\{w_{t+1}\}} \max_{\{c_t,k_t\}} -E\{\sum_{t=0}^{\infty} \beta^t \left(\frac{1}{2}\right) [(s_t - b_t)\cdot(s_t - b_t) + (g_t)\cdot(g_t)] - \beta\theta w_{t+1}\cdot w_{t+1}\}$$

subject to the following set of restrictions:

$$\begin{aligned}
\Phi_c c_t + \Phi_g g_t + \Phi_i i_t &= \Gamma k_{t-1} + d_t \\
k_t &= \Delta_k k_{t-1} + \Theta_k i_t \\
h_t &= \Delta_h h_{t-1} + \Theta_h c_t \\
s_t &= \Lambda h_{t-1} + \Pi c_t \\
b_t &= U_b z_t \\
d_t &= U_d z_t
\end{aligned}$$

and the minimizer or "evil" agent chooses w_{t+1}, that will affect:

$$z_{t+1} = A_{22} z_t + C_2 w_{t+1}$$

w_{t+1} is not an exogenous Gaussian vector any more, but corresponds to a chosen process by the evil agent (It will happen though that in these types of problems, the worst choice happens to be Gaussian too with mean w_{t+1}! and this is a feature we exploit later). Then, these shocks are transformed accordingly to $b_t = U_b z_t$ and $d_t = U_d z_t$.

The Robustness problem can also be written as a Lagrangian:

$$\begin{aligned}
&\min_{\{w_{t+1}\}} \max_{\{c_t,k_t\}} -\{\sum_{t=0}^{\infty} (1/2)\,\beta^t [(s_t - b_t)\cdot(s_t - b_t) + (g_t)\cdot(g_t)] - \beta\theta \frac{w_{t+1}\cdot w_{t+1}}{2}\} \\
&+M_t^{d\prime}[\Phi_c c_t + \Phi_g g_t + \Phi_i i_t - \Gamma k_{t-1} - d_t] \\
&+M_t^{k\prime}[k_t - \Delta_k k_{t-1} - \Theta_k i_t] \\
&+M_t^{h\prime}[h_t - \Delta_h h_{t-1} - \Theta_h c_t] \\
&+M_t^{s\prime}[s_t - \Lambda h_{t-1} - \Pi c_t] \\
&+M_t^{z\prime}[-z_{t+1} + A_{22} z_t + C_2 w_{t+1}]\}
\end{aligned}$$

5 The Robust Solution

The set of linear equations determining the outcome will be similar to those of the original LQ problem:

$$
\begin{aligned}
(c_t) &: -M_t^{d\prime}\Phi_c + M_t^{h\prime}\Theta_h + M_t^{s\prime}\Pi = 0 \qquad (1) \quad \{\texttt{SYSTEM1}\}\\
(g_t) &: g_t - \Phi_g M_t^d = 0\\
(h_t) &: -M_t^{h\prime} + \beta E\left[M_{t+1}^{h\prime}\Delta_h + M_{t+1}^{s\prime}\Lambda\right] = 0\\
(i_t) &: -M_t^{d\prime}\Phi_i + M_t^{k\prime}\Theta_i = 0\\
(k_t) &: -M_t^{k\prime}\Delta_k + \beta E\left[M_{t+1}^{d\prime}\Gamma + M_{t+1}^{k\prime}\Delta_k\right] = 0\\
(s_t) &: -(s_t - b_t) + M_t^s = 0\\
(w_{t+1}) &: \beta\theta w_{t+1} + C_2^{'} M_t^z = 0\\
(z_{t+1}) &: -\beta E\left[U_d^{\prime} M_{t+1}^d\right] + \beta E\left[A_{22}^{\prime} M_{t+1}^z\right] - \beta E\left[(s_{t+1} - b_{t+1})\, U_b^{\prime} M_{t+1}^z\right] - M_t^z = 0
\end{aligned}
$$

This set of equations is then complemented by:

$$
\begin{aligned}
\Phi_c c_t + \Phi_g g_t + \Phi_i i_t &= \Gamma k_{t-1} + d_t\\
k_t &= \Delta_k k_{t-1} + \Theta_k i_t\\
h_t &= \Delta_h h_{t-1} + \Theta_h c_t\\
s_t &= \Lambda h_{t-1} + \Pi c_t
\end{aligned}
$$

and the minimizer or "evil" agent chooses w_{t+1}, that will affect:

$$z_{t+1} = A_{22} z_t + C_2 (w_{t+1} + \varepsilon_t)$$

where $\varepsilon_t \sim N(0, I)$.

Before explaining how the matlab code works we need to explain how we map the worst case distortions into our LQ framework. Notice that ε_t was not appearing in our original problem.

In the original Robustness Setup, the evil agent chooses a distribution over the sequences of outcomes w_t, but not the outcomes themselves. By the certainty equivalence principle that shows up in LQ problems, we know that for any choice of the distribution (in the standard setup) for w_t, the maximizer's optimal behavior will depend only on the first moment. Hence the choice of w_{t+1} in this setup is equivalent to the choice of the mean of a particular process. But why is then ε_t also Gaussian, and moreover $N(0,I)$? Well, the distorted probability will behave like we changed the maximizer agents model's mean zero Gaussian shocks to $N(w_{t+1},I)$. Gaussianity is a property of the Robust Problem when the approximating model is Gaussian. The i.i.d with variance 1 of the shocks is in fact, not strictly correct: the worst Gaussian Variance-Covariance matrix in fact needs to be computed differently, but at this stage, we only care for the optimal policy of the maximizing agent and the mean of the worst case scenario and this is sufficient. Please take this into account when interpreting (and scaling) impulse responses. We also include in our zip file of matlab codes a file called

```
HSLQ_dyn.m
```

that computes the standard LQ problem without robustness described above. That file treats w directly as a vector of exogenous disturbances.

6 Using Dynare to Compute Robust Policies

The HS_Robust_dyn.m code enables the user to compute robust policies with dynare by simply by inputting matrixes from the LQ setup. The program reads matrixes from the LQ setup and uses the general solution to the problem to write a MOD file for the system of stochastic difference equations. The instructions are very simple.

6.1 What to do?

What to do? The user must write up a simple m-file to use to declare all of the matrixes in the linear quadratic setup and declare some of the optional parameters. The user should write up an m-file with that includes the following:

1. **Declaration of matrixes:** The first thing the user has to do is declare ALL of these matrixes: `Phi_c, Phi_g, Phi_i, Gamma, Delta_k, Theta_k, Delta_h,.Theta_h, Lambda, Pi, A22,C2, U_b, U_d;`

2. **Declaration of options for Dynare:** Variables that go in the standard set of options for dynare should be included. For example one can include statements like:

   - `periods=5000;` (for number of simulations in Dynare)
   - `irfperiods=15;` (for number of periods in irf's in Dynare)

3. **File Name:** A statement assigning the variable `filename` a script should be included:(e.g. `filename ='Hall_Test.mod';`

4. **Optional:** Assign a value for the optional dummy variable `run` if you decide whether or not you want to directly execute the generated dynare. Assign `run=1;`

5. **Constant in state vector:** If there is a constant in the state vector z, you have to set `constant=1`. The constant has to be the first element of the z vector.

6. **Calling the file:** Call the Hansen Sargent converter by simply writing the following statement `HS_Robust_dyn;`

6.2 How does the code work?

How does the code do it? The code does something simple. It reads the matrixes from HS-LQ setup and declares variables and equations in a mod.file using dynare syntax. The steps followed are:

1. Reading Dimensions of Matrixes
   - For example, for the matrix `Phi_c,` it uses the following statement to set in memory the number of rows and columns:
     [rr_Phi_c cc_Phi_c] =size(Phi_c) ;
2. Checking Consistency:
   - The matrixes must be conformable in HS-LQ setup. The code checks for consistency along columns and rows:

```
if  cc_Phi_c~=cc_Theta_h || cc_Pi~=cc_Theta_h || cc_Phi_i~=cc_Theta_k || ...
cc_Delta_h~=cc_Lambda || cc_Delta_k~=cc_Gamma || cc_U_b~=cc_U_d || cc_U_b~=cc_A22
disp('Error -- dimensions of matrices do not match... check number of columns')
end

if  rr_Phi_c~=rr_Phi_g || rr_Phi_c~=rr_Phi_g || rr_Phi_c~=rr_Gamma || ...
rr_Delta_k~=rr_Theta_k || rr_Delta_h~=rr_Theta_h || rr_Lambda~=rr_Pi || rr_A22~=rr_C2
disp('Error -- dimensions of matrices do not match... check number of rows')
end


    disp('Error -- dimensions of matrices do not match... check number
of columns')
end
```

3. Creating an m-file:
   - This statement generates an object in memory that is used to record what we are writing:
     `fid = fopen(filename,'wt');`
4. Declaring Variables:

- For each of the matrixes in the LQ setup. The converter, will read the number of columns and assign a variable. The number of columns corresponds to the number of consumption variables. For example, it uses the following statement to declare the consumption goods variables in the resource constraint block:

```
fprintf(fid,'\n var ');
for i=1:(cc_Phi_c)
    fprintf(fid,'c%g ',i);
end
```

5. Declaring Parameters:

- A similar proceduer is used to declare parameters. This block writes up the paramter declaration followed by the variables `beta, theta,` and the elements in `Phi_c`:

```
fprintf(fid,'parameters ');
fprintf(fid,'beta ') ;
fprintf(fid,'theta ') ;
for j=1:rr_Phi_c
    for i=1:cc_Phi_c
         fprintf(fid,'Phi_c_%g%g ',j,i);
    end
end
```

6. Setting Parameter Values:

- Using a similar statement we declare parameters. The variable %g in the statement is assign the value of the variable after the second comma. `\n` means that the lines ends there:

```
fprintf(fid,'beta=%g ;\n',beta) ;
fprintf(fid,'theta=%g ;\n',theta) ;
for j=1:rr_Phi_c
    for i=1:cc_Phi_c
        fprintf(fid,'Phi_c_%g%g =%g ;\n',j,i,Phi_c(j,i));
    end
end
```

7. Declaring Equations:

- This section of the code writes up the corresponding equation for the system that characterizes the solution to the robust problem (1). For example, to write in the first block of equations in the system: $M_t^{d\prime}\Phi_c + M_t^{h\prime}\Theta_h = 0$ we use the following commands:

```
for i=1:(rr_Phi_c)
    for j=1:(cc_Phi_c)
        fprintf(fid,'Phi_c_%g%g*c%g + ',i,j,j);
    end
    for j=1:(cc_Phi_g)
        fprintf(fid,'Phi_g_%g%g*g%g + ',i,j,j);
    end
    for j=1:(cc_Phi_i)
        fprintf(fid,'Phi_i_%g%g*i%g = ',i,j,j);
    end
    for j=1:(cc_Gamma)
        fprintf(fid,'Gamma_%g%g*k%g + ',i,j,j);
    end
    fprintf(fid,'d%g',i);
    fprintf(fid,'; \n');
end
```

8. Running Dynare:

- This step simply follows the dynare syntax to call dynare according to the previously defined preferences:

```
fprintf(fid,'shocks; \n');
for i=1:cc_C2
    fprintf(fid,'var e%g; \n',i);
    fprintf(fid,'stderr 1; \n');
end
fprintf(fid,'end; \n )
fprintf(fid,' \n');
fprintf(fid,'steady; \n % n');

fprintf(fid,'stoch\_simul(solve\_algo=3, periods=\%g); \n',irfperiods,periods);
```

```
fclose(fid);
if (run)
    dynare(filename)
end
```

7 One Example

This section shows how the model in Hansen, Sargent & Tallarini can be solved using our code. Details on how to map a LQ decision problem to the form used here can be found in the LQ Hansen and Sargent Manuscript.

7.1 Hansen, Sargent and Tallarini (1999, RES)

7.1.1 The Problem

This is a version of the model of Hansen, Sargent and Tallarini (1999). The planner solves the following problem:

$$\min_{\{w_{t+1}\}} \max_{\{c_t,k_t\}} E\{\sum_{t=0}^{\infty} \beta^t - (1/2)\left[(s_t - b_t)^2 - \beta\Theta\frac{w_{t+1}\cdot w_{t+1}}{2}\right]$$

The input of the utility function are an indirect service for the household, s_t, and a preference shock b_t that satisfy the following:

$$s_t = \Lambda h_{t-1} + \Pi c_t$$

The resource constraint for this economy is a linear production function and

$$c_t + i_t = (1+r)\,k_{t-1} + y_t$$

an exogenous productivity shift:

$$y_t = U_y z_t$$

The Robust planner problem is different from the standard model in that: a minimizer or "evil" agent chooses w_{t+1}, that will affect:

$$z_{t+1} = A_z z_t + C_Z w_{t+1}$$

and we parameterize this to have a stable solution:

$$\gamma := (1+r) = \frac{1}{\beta}$$

For a pair of AR(2) processes, we have the following values of A_z and C_Z:

$$A_z = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 \\ 0 & \rho_{22} & \rho_{23} & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & \rho_{44} & \rho_{45} \\ 0 & 0 & 0 & 1 & 0 \end{bmatrix} \text{ and } C_z = \begin{bmatrix} 0 & 0 \\ 1 & 0 \\ 0 & 0 \\ 0 & 1 \\ 0 & 0 \end{bmatrix}$$

$U_y = [5\ 1\ 0\ 1\ 0]$.

7.1.2 The Code

The following code (

`Example_HST.m`

) computes the equilibrium of the model described above.

```
% Declaration of Parameters

delta=0.05;

beta=1/1.05;

theta=10;


% Matrix Form

Phi_c=1 ;

Phi_g=0 ;

Phi_i=1 ;
```

```
Gamma=0.1 ;

Delta_k=1-delta ;

Theta_k=1 ;

Delta_h=0 ;

Theta_h=1 ;

Lambda=0 ;

Pi=1 ;

A22=[1 0 0 0 0; 0 0.1 0.2 0 0; 0 1 0 0 0; 0 0 0 0.1 0.2; 0 0 0 0 1];

C2=[0 0; 1 0; 0 0; 0 1; 0 0] ;

U_b=[30 0 0 0 0] ;

U_d=[5 1 0 1 0] ;

%% User Preferences for Dynare

periods=5000; %parameters for simulation

irfperiods=15; %parameters for irf's

filename='HST_Test_AR2.mod'; %pick filename for mod file

%Optional: decide whether or not you want to directly execute the generated
dynare

%code (set run=1 to execute)

run=1;

constant=1;

%set=1 if first element of z vector is a constant

%% Call the Hansen Sargent converter:

HS_Robust_dyn;
```

This piece of matlab code generates a Mod-File that dynare is able to run. The matlab file is saved as `'HST_Test_AR2.mod'`. We can alter this mod file to be able to estimate the model.