Optimal Control of RE Models using Ox

Andrew P. Blake

1 Control problem

We begin by considering the general control problem:

$$\min_{u_t} \frac{1}{2} \sum_{0}^{\infty} \rho^t (z_t' Q z_t + 2 z_t' U u_t + u_t' R u_t) \tag{1}$$

subject to the model:

$$z_{t+1} = A z_t + B u_t. \tag{2}$$

This has no rational expectations. We will consider this later.

1.1 Equivalence of dynamic programming and Hamiltonian formulations

1.2 Dynamic programming

The dynamic programming problem is:

$$\frac{1}{2} z_t' S z_t = \min_{u_t} \frac{1}{2} (z_t' Q z_t + 2 z_t' U u_t + u_t' R u_t + \rho z_{t+1}' S z_{t+1}) \tag{3}$$

in which we substitute the model (2) into to give at the optimum:

$$z_t' S z_t = \min_{u_t} (z_t' Q z_t + 2 z_t' U u_t + u_t' R u_t + \rho (u_t' B' + z_t' A') S (A z_t + B u_t)). \tag{4}$$

The first order condition that gives this optimum is:

$$0 = U' z_t + R u_t + \rho B' S (A z_t + B u_t)$$

which gives:

$$u_t = -(R + \rho B' S B)^{-1} (\rho B' S A + U') z_t = -F z_t. \tag{5}$$

Usually this is used directly in (3) to give a recursion for S after dropping the common z_t, given by:

$$S = Q - UF - F'U' + F'RF + \rho(A' - F'B')S(A - BF) \tag{6}$$

but notice we can collect terms in (4) to give:

$$z_t'Sz_t = z_t'Qz_t + 2z_t'(U + \rho A'SB)u_t + u_t'(R + \rho B'SB)u_t + \rho z_t'A'SAz_t. \tag{7}$$

Substituting in $u_t = -Fz_t$ gives:

$$z_t'Sz_t = z_t'Qz_t - 2z_t'(U + \rho A'SB)Fz_t + z_t'F'(R + \rho B'SB)Fz_t + \rho z_t'A'SAz_t \tag{8}$$

or as a recursion in S alone::

$$S = Q - 2(U + \rho A'SB)F + F'(R + \rho B'SB)F + \rho A'SA \tag{9}$$

However, notice that:

$$\begin{aligned}(U + \rho A'SB)F &= (U + \rho A'SB)(R + \rho B'SB)^{-1}(\rho B'SA + U') \\ &= (U + \rho A'SB)(R + \rho B'SB)^{-1} \\ &\qquad (R + \rho B'SB)(R + \rho B'SB)^{-1}(\rho B'SA + U') \\ &= F'(R + \rho B'SB)F\end{aligned}$$

where we have used the symmetry of $(R + \rho B'SB)$, so we can write (9) as:

$$\begin{aligned}S &= Q - 2(U + \rho A'SB)F + (U + \rho A'SB)F + \rho A'SA \\ &= Q - (U + \rho A'SB)F + \rho A'SA \\ &= Q - (U + \rho A'SB)(R + \rho B'SB)^{-1}(\rho B'SA + U') + \rho A'SA. \end{aligned} \tag{10}$$

This means that we can represent exactly equivalent recursions for S by (6) and (10).

1.2.1 Hamiltonian approach

We derive two equivalent solutions using Lagrange multipliers. The Hamiltonian for the control problem is:

$$H_t = \frac{1}{2}\rho^t(z_t'Qz_t + 2z_t'Uu_t + u_t'Ru_t) + \mu_{t+1}'(Az_t + Bu_t - z_{t+1}) \tag{11}$$

where the first order conditions together are:

$$Qz_t + Uu_t + \rho A'\lambda_{t+1} - \lambda_t = 0 \tag{12}$$

and:

$$Ru_t + U'z_t + \rho B'\lambda_{t+1} = 0 \tag{13}$$

where $\lambda_t = \mu_t \rho^{-t}$. Note that we can re-arrange (13) as:

$$u_t = -R^{-1}U'z_t - \rho R^{-1}B'\lambda_{t+1}. \tag{14}$$

From (2) and (14) we get:

$$z_{t+1} = Az_t + Bu_t = (A - BR^{-1}U')z_t - \rho BR^{-1}B'\lambda_{t+1} \tag{15}$$

or:

$$z_{t+1} = (A - BR^{-1}U')z_t - \rho BR^{-1}B'Sz_{t+1}$$

so:

$$(I + \rho BR^{-1}B'S)z_{t+1} = (A - BR^{-1}U')z_t$$

implying:

$$z_{t+1} = (I + \rho BR^{-1}B'S)^{-1}(A - BR^{-1}U')z_t. \tag{16}$$

Now consider the second first order condition (12) and substitute out for λ to give:

$$Qz_t + Uu_t + \rho A'Sz_{t+1} - Sz_t = 0$$

then substitute in for u_t from (14):

$$Qz_t - UR^{-1}U'z_t + \rho(A' - UR^{-1}B')Sz_{t+1} - Sz_t = 0$$

and use (16) for z_{t+1} to give:

$$\begin{aligned} 0 &= Qz_t - UR^{-1}U'z_t \\ &\quad +\rho(A' - UR^{-1}B')S(I + \rho BR^{-1}B'S)^{-1}(A - BR^{-1}U')z_t - Sz_t \end{aligned}$$

or dropping z_t:

$$S = Q - UR^{-1}U' + \rho(A' - UR^{-1}B')S(I + \rho BR^{-1}B'S)^{-1}(A - BR^{-1}U'). \tag{17}$$

This is another recursion in S.

In stacked from we can write equations (12) and (13) as:

$$\begin{bmatrix} I & \rho BR^{-1}B' \\ 0 & \rho(A' - UR^{-1}B') \end{bmatrix} \begin{bmatrix} z_{t+1} \\ \lambda_{t+1} \end{bmatrix} = \begin{bmatrix} (A - BR^{-1}U') & 0 \\ -(Q - UR^{-1}U') & I \end{bmatrix} \begin{bmatrix} z_t \\ \lambda_t \end{bmatrix}. \tag{18}$$

We can derive the above routines from this, by recogising the saddlepath nature of the solution to (18). We can solve this using the method outlined by Söderlind (1999). This is done in the Ox code below, and can be seen as a generalisation of the method due to Blanchard and Kahn (1980).

This is, of course, equivalent to solving for (17) as essentially all we have done is substitute in a value for S.

1.2.2 Equivalence

Note we can rewrite (12) as:

$$Ru_t + U'z_t + \rho B'S(Az_t + Bu_t) = 0 \tag{19}$$

assuming a solution for $\lambda_{t+1} = Sz_{t+1}$ so that collecting terms we get:

$$(R + \rho B'SB)u_t = -(U' + \rho B'SA)z_t$$

or:

$$u_t = -(R + \rho B'SB)^{-1}(U' + \rho B'SA)z_t \tag{20}$$

which is exactly the same as (5).

From (12) by eliminating the z's we can obtain:

$$Qz_t + (U + \rho A'SB)u_t + \rho A'SAz_t - Sz_t = 0 \tag{21}$$

and use (20) and drop the common z_t to get a recursion is S:

$$S = Q - (U + \rho A'SB)(R + \rho B'SB)^{-1}(U' + \rho B'SA) + \rho A'SA. \tag{22}$$

This is identical to (10) and we have shown the equivalence of the two approaches.

1.3 Singular control problems

If the model is instead:

$$Ez_{t+1} = Az_t + Bu_t$$

with E a possibly singular matrix, then the natural Hamiltonian is:

$$\begin{bmatrix} E & \rho BR^{-1}B' \\ 0 & \rho(A' - UR^{-1}B') \end{bmatrix} \begin{bmatrix} z_{t+1} \\ \lambda_{t+1} \end{bmatrix} = \begin{bmatrix} (A - BR^{-1}U') & 0 \\ -(Q - UR^{-1}U') & E' \end{bmatrix} \begin{bmatrix} z_t \\ \lambda_t \end{bmatrix}$$

which is in the form of the singular RE model and can be solved using the methods outlined in Söderlind (1999).

We can also write the model a little differently by not substituting out for the instruments, as:

$$\begin{bmatrix} R & 0 & \rho B' \\ -B & E & 0 \\ U & 0 & \rho A' \end{bmatrix} \begin{bmatrix} u_t \\ z_{t+1} \\ \lambda_{t+1} \end{bmatrix} = \begin{bmatrix} 0 & -U' & 0 \\ 0 & A & 0 \\ 0 & -Q & E' \end{bmatrix} \begin{bmatrix} u_{t-1} \\ z_t \\ \lambda_t \end{bmatrix}. \tag{23}$$

This useful formulation indicates that the matrix R need not be non-singular. We need to remember that the number of predetermined variables is increased!

2 Rational expectations models

Now consider a model with a rational expectations structure, i.e.:

$$\begin{bmatrix} E_{11} & E_{12} \\ E_{21} & E_{22} \end{bmatrix} \begin{bmatrix} z_{t+1} \\ x_{t+1} \end{bmatrix} = \begin{bmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{bmatrix} \begin{bmatrix} z_t \\ x_t \end{bmatrix} + \begin{bmatrix} B_1 \\ B_2 \end{bmatrix} u_t \tag{24}$$

where x_t are expectational or jump variables. E may be singular.

2.1 Time inconsistent control

Nothing is intrinsically different, except that we need to calculate different initial conditions for a subset of the co-states. Thus we set the co-states associated with the jump variable to zero and allow the jump variables to jump.

This turns out to be easy. The easiest is to take the model under control (23) and re-order the model so that the pre-determined co-states and the jump variables swap. Then set the pre-determined co-states to zero and solve the model. From a re-ordered (23) we can easily obtain

$$u_t = -F \begin{bmatrix} z_t \\ \lambda_t \end{bmatrix}$$

and:

$$x_t = -N \begin{bmatrix} z_t \\ \lambda_t \end{bmatrix}$$

where λ_t are the pre-determined co-states. These are the policy rule and the reaction function respectively.

We need to calculate the loss function separately. Remember the per-period cost is:

$$C_t = \begin{bmatrix} z_t' & x_t' \end{bmatrix} Q \begin{bmatrix} z_t \\ x_t \end{bmatrix} + \begin{bmatrix} z_t' & x_t' \end{bmatrix} U u_t + u_t' U' \begin{bmatrix} z_t \\ x_t \end{bmatrix} + u_t' R u_t.$$

so that using our rules above:

$$\begin{bmatrix} z_t \\ x_t \end{bmatrix} = \begin{bmatrix} I & 0 \\ -N_1 & -N_2 \end{bmatrix} \begin{bmatrix} z_t \\ \lambda_t \end{bmatrix} = T \begin{bmatrix} z_t \\ \lambda_t \end{bmatrix}.$$

Let $S = T'QT - T'UF - F'U'T + F'RF$ so we can now write the per-period cost function as:

$$C_t = \begin{bmatrix} z_t' & \lambda_t' \end{bmatrix} S \begin{bmatrix} z_t \\ \lambda_t \end{bmatrix}$$

so that the welfare loss-to-go is:

$$W_t = \begin{bmatrix} z_t' & \lambda_t' \end{bmatrix} P \begin{bmatrix} z_t \\ \lambda_t \end{bmatrix}$$

with $P = \rho \hat{A}' P \hat{A} + S$.

2.2 Time consistent control

As before, there is no restriction on E being nonsingular but we need to be able to write the model in the form:

$$\begin{bmatrix} I & E_{12} \\ E_{21} & E_{22} \end{bmatrix} \begin{bmatrix} z_{t+1} \\ x_{t+1} \end{bmatrix} = \begin{bmatrix} A_{11} & A_{12} \\ A_{21} & I \end{bmatrix} \begin{bmatrix} z_t \\ x_t \end{bmatrix} + \begin{bmatrix} B_1 \\ B_2 \end{bmatrix} u_t. \tag{25}$$

We need to solve the rational expectation and the control problem together using dynamic programming. If we assume $x_{t+1} = -N z_t$, then using the definitions:

$$\begin{aligned} \tilde{A}_{11} &= (I - E_{12}N)^{-1} A_{11} \\ \tilde{A}_{12} &= (I - E_{12}N)^{-1} A_{12} \\ \tilde{B}_1 &= (I - E_{12}N)^{-1} B_1 \end{aligned}$$

we can obtain:

$$x_t = J z_t + K u_t \tag{26}$$

where:

$$J = (I + (E_{22}N - E_{21})\tilde{A}_{12})^{-1}((E_{22}N - E_{21})\tilde{A}_{11} + A_{21}) \tag{27}$$
$$K = (I + (E_{22}N - E_{21})\tilde{A}_{12})^{-1}((E_{22}N - E_{21})\tilde{B}_1 + B_2) \tag{28}$$

from (25).

Substituting out for x_t we can then differentiate the appropriate value function similar to (4) to obtain:

$$F = (R + (Q_{22} - U_2')K - U_2 + \rho(\tilde{B}_1 - \tilde{A}_{12})'S(\tilde{B}_1 - \tilde{A}_{12}K))^{-1} \\ (\rho(B_1 - \tilde{A}_{12})'S(\tilde{A}_{11} - \tilde{A}_{12}J) + U_1' + (Q_{22} - U_2')J - Q_{21}). \tag{29}$$

From (26)it must be the case that the rational expectation is given by

$$N = J - KF. \tag{30}$$

Finally, define:

$$W = \tilde{A}_{11} - \tilde{A}_{12}N - \tilde{B}_1F$$
$$V = U_1 - N'U_2$$

so the Ricatti equation can be written:

$$S = Q_{11} - N'Q_{21} - Q_{12}N + N'Q_{22}N - VF - F'V' + F'RF + \rho W'SW.$$

This gives us the final recursion for time consistent control. We iterate on every equation in this section (except the model) in order to arrive at the solution.

3 An illustrative model

We set up the Galí and Monacelli (2002) model, which can be replaced by any model in state space. The model can be written in algebra as:

$$y_t = y^e_{t+1} - \frac{1}{\sigma_a}(i_t - \pi^e_{t+1}) \tag{31}$$

$$\pi_t = \beta\pi^e_{t+1} + \kappa y_t \tag{32}$$

$$\pi^c_t = \pi_t + \frac{\alpha}{1-\alpha}(q_t - q_{t-1}) \tag{33}$$

$$q_t = q^e_{t+1} - (i_t - \pi^e_{t+1}) \tag{34}$$

$$i_t = i_{t-1} + di_t \tag{35}$$

where α is the share of home relative to foreign goods in the consumption basket, η is the elasticity of substitution between them, β is the social discount factor, θ the proportion of fixed price firms every period, and the per-period utility function is $U(C,\ N) = \left(\frac{C^{1-\sigma}}{1-\sigma} + \frac{N^{1+\psi}}{1+\psi}\right)$ with C consumption and N labour supply.

The equations are respectively an IS curve (31), a New Keynesian Phillips curve (32), the CPI equation (33) and the real exchange rate (34). A final equation (35) is used to create a lag of the interest rate. In this case di becomes the instrument.

We assume the values $\eta = 1$, $\alpha = 0.4$, $\beta = 0.99$, $\theta = 0.75$, $\sigma = 1$, $\psi = 3$, $\omega = \sigma\eta + (1-\alpha)(\sigma\eta - 1)$, $\sigma_a = \sigma/(1 - \alpha + \alpha\omega)$, $\lambda = (1-\theta)(1-\beta\theta)/\theta$ and $\kappa = \lambda(\psi + \sigma_a)$.

In state space we write this as:

$$\begin{bmatrix} 1 & 0\,0\,0 & 0 & 0 \\ 0 & 1\,0\,0 & 0 & 0 \\ 0 & 0\,1\,0 & 0 & 0 \\ -\frac{1}{\sigma_a} & 0\,0\,1 & \frac{1}{\sigma_a} & 0 \\ 0 & 0\,0\,0 & \beta & 0 \\ 1 & 0\,0\,0 & -1 & 1 \end{bmatrix} \begin{bmatrix} i_t \\ q_t \\ pi^c_t \\ y^e_{t+1} \\ \pi^e_{t+1} \\ q^e_{t+1} \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & \frac{\alpha}{\alpha-1} & 0 & 0 & 1 & \frac{\alpha}{1-\alpha} \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & -\kappa & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} i_{t-1} \\ q_{t-1} \\ \pi^c_{t-1} \\ y_t \\ \pi_t \\ q_t \end{bmatrix} + \begin{bmatrix} 1 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix} di_t.$$

4 Ox code

The procedure `Hamilton` is for the most general control problem above. In this program it does not require the system to have anything but a simple one equation per variable.

```
#include<oxstd.h>
#include<oxdraw.h>
#include<oxfloat.h>


// Time inconsistent routines

// Generalised Blanchard-Kahn based on generalised real Schur

calNG(const al, const ar, const j)
 {decl n=rows(ar)-1, a, b, s, t, vl, vr, r;
  r=decschurgen(ar, al, &a, &b, &s, &t, &vl, &vr, 0, 1.00001);
  b=b+(b .== 0)*1e-08; r=cdiv(a,b);
  print("\n     Real        Imag        Abs", "%#12.5g", r'~cabs(r)');
  if (sumr(cabs(r) .> 1.0000001) != j) print("\nNot saddlepath");
   else print("\nBlanchard-Kahn conditions satisfied");
  vl=vr[0:n-j][0:n-j]; vr=vr[n-j+1:n][0:n-j];
  return {vl*(t[0:n-j][0:n-j]^-1)*s[0:n-j][0:n-j]*vl^-1, -vr*vl^-1};}

// Set up a Hamiltonian: L z_{t+1} = R z_t

HamiltonU(const l, const a, const b, const rho, const q, const u, const r)
 {return { r ~ zeros(columns(b),rows(a)) ~ rho*b'|
          -b ~ l                         ~ zeros(rows(a),rows(a)) |
           u ~ zeros(rows(a),rows(a))    ~ rho*a',
            zeros(columns(b),columns(b)) ~ -u' ~ zeros(columns(b),rows(a)) |
            zeros(rows(a),columns(b))    ~  a  ~ zeros(rows(a),rows(a)) |
            zeros(rows(a),columns(b))    ~ -q  ~ l'}; }

// Reorder Hamiltonian to put jumps last

OrderHamU(l, r, m, u)
 {decl n=(rows(l)-u)/2;
  l=l[0:u+n-m-1][]|l[u+2*n-m:u+2*n-1][]|l[u+n:u+2*n-m-1][]|l[u+n-m:u+n-1][];
  l=l[][0:u+n-m-1]~l[][u+2*n-m:u+2*n-1]~l[][u+n:u+2*n-m-1]~l[][u+n-m:u+n-1];
  r=r[0:u+n-m-1][]|r[u+2*n-m:u+2*n-1][]|r[u+n:u+2*n-m-1][]|r[u+n-m:u+n-1][];
  r=r[][0:u+n-m-1]~r[][u+2*n-m:u+2*n-1]~r[][u+n:u+2*n-m-1]~r[][u+n-m:u+n-1];
  return { l, r }; }
```

```
// Time inconsistent control from Hamiltonian form

TIHU(e, a, b, q, u, r, beta, m)
 {decl n=rows(a)-1, j=n-m, nu=columns(b), al, ar, aa, ee, s;
  [al, ar] = HamiltonU(e, a, b, beta, q, u, r);
  [al, ar] = OrderHamU(al, ar, m, nu);
  [aa, s]  = calNG(al, ar, n+1);
  return {aa[nu:nu+n][nu:nu+n], s[][nu:nu+n], -aa[0:nu-1][nu:nu+n]}; }

// Time consistent routines

// Partition system matrices conformable with jumps

PartMatricesE(const e, const a, const b, const q, const u, const m)
 {decl i=rows(a)-1-m,j=rows(a)-1,ei=e[0:i][0:i]^-1,ai=a[i+1:j][i+1:j]^-1;
  return {ei*e[0:i][i+1:j], ai*e[i+1:j][0:i], ai*e[i+1:j][i+1:j],
          ei*a[0:i][0:i], ei*a[0:i][i+1:j], ai*a[i+1:j][0:i],
          ei*b[0:i][], ai*b[i+1:j][],
          q[0:i][0:i], q[0:i][i+1:j], q[i+1:j][0:i], q[i+1:j][i+1:j],
          u[0:i][], u[i+1:j][]}; }

// Time consistent control

controlTCTE(const e, const a, const b, const rho, const q, const u, const r,
            const m, p, const maxit)
 {decl v, n, s, w, j, k, i=1, so, h, f, no,
       e12, e21, e22, a11, a12, a21, b1, b2,
       a11t, a12t, b1t, q11, q12, q21, q22, u1, u2;
  fuzziness(1e-16);

  //
  // Expects input in the format:
  //   [ e11 e12 ] [z_{t}  ] = [ a11 a12 ] [z_{t-1}] + [ b1 ] u_t
  //   [ e21 e22 ] [x_{t+1}]   [ a21 a22 ] [x_t    ]   [ b2 ]
  // There is no restriction on e being nonsingular, but e11 and a22
  // are required to be. PartMatricesE calculates the inverses and
  // puts the model in the form:
  //   [ I   e12 ] [z_{t}  ] = [ a11 a12 ] [z_{t-1}] + [ b1 ] u_t
  //   [ e21 e22 ] [x_{t+1}]   [ a21 I   ] [x_t    ]   [ b2 ]
  //

  if (rows(p) == 1) {p=p+zeros(columns(b),1);} p=diagonalize(p);
  [e12,e21,e22,a11,a12,a21,b1,b2,q11,q12,q21,q22,u1,u2]
    = PartMatricesE(e,a,b,q,u,m);
```

```
  s=q11; n=a21;
  while (i <= maxit)
   {no=n; so=s;
    w = (unit(rows(a)-m)-e12*n)^-1;
    a11t = w*a11;
    a12t = w*a12;
    b1t = w*b1;
    w = (unit(m)+(e22*n-e21)*a12t)^-1;
    j = w*((e22*n-e21)*a11t+a21);
    k = w*((e22*n-e21)*b1t+b2);
    h = k*p;
    f = (r-u2'k+h'(q22*k-u2)+rho*(b1t-a12t*h)'s*(b1t-a12t*k))^-1 *
          (rho*(b1-a12t*h)'s*(a11t-a12t*j)+u1'-u2'j+h'(q22*j-q21));
    n = j-k*f;
    w = a11t-a12t*n-b1t*f;
    v = u1-n'u2;
    s = q11-n'q21-q12*n+n'q22*n-v*f-f'v'+f'r*f+rho*w's*w;
    if (isfeq(so,s)+isfeq(no,n) == 2) { break; }
    ++i;}
  print("\nNumber of iterations = ", i);
  return {w, n, f, s}; }



// Utilities: Simulation, costs, plotting

// Sumulate a model, jumps, outputs and controls: Use zeros for no eqns

simANCDF(const a, const c, const d, const f, const n, z, const nsim)
 {decl i=1; while (i < nsim) { z=z~a*z[][i-1]; ++i; }
  return {z, -n*z, (c*(unit(rows(z))|-n)-d*f)*z, -f*z};}

// Lyapunov solver based on vectorisation

lyapunov(const a, const r, const o)
 {decl i=rows(a); return shape((unit(i*i)-r*a**a)^-1 * vec(o), i, i);}

// Cost-to-go

wl(const z, const v)
 {return sumc(z.*(v*z))/2;}
```

```
// Calculate value function

CalS(q, u, r, n, f, a, b)
 {decl t=(unit(rows(q)-rows(n))~zeros(rows(q)-rows(n),rows(n)))|-n;
  return lyapunov(a', b, t'q*t - t'u*f - f'u't + f'r*f);}

// Plots all sets of series: If zero no plots. Saves in file "name"
// USe file extension to set the type (ps, eps, gwg)

PlotRE(const z, const x, const y, const u, const c, const name)
 {decl i=0, n=range(1, columns(z));
  if (max(fabs(z)) != 0)
   {DrawXMatrix(i, z, 0, n, 0); DrawTitle(i, "States: $z_t$"); ++i;}
  if (max(fabs(x)) != 0)
   {DrawXMatrix(i, x, 0, n, 0); DrawTitle(i, "Jumps: $x_t$"); ++i;}
  if (max(fabs(y)) != 0)
   {DrawXMatrix(i, y, 0, n, 0); DrawTitle(i, "Outputs: $y_t$"); ++i;}
  if (max(fabs(u)) != 0)
   {DrawXMatrix(i, u, 0, n, 0); DrawTitle(i, "Controls: $u_t$"); ++i;}
  if (max(fabs(c)) != 0)
   {DrawXMatrix(i, c, 0, n, 0); DrawTitle(i, "Cost-to-go");}
  SetDrawWindow(name); SaveDrawWindow(name); ShowDrawWindow(); }
```

```
// Model

// Gali & Monacelli model v 1a

modelGM1()

 {decl a=zeros(6,6), b=zeros(6,1), e=unit(6), c=zeros(2,6),
       d=zeros(2,1), m=3,
       eta=1, alpha=0.4, beta=0.99, theta=0.75, sigma=1, psi=3,
       omega=sigma*eta+(1-alpha)*(sigma*eta-1),
       sigma_a=sigma/(1-alpha+alpha*omega),
       lambda=(1-theta)*(1-beta*theta)/theta,
       kappa=lambda*(sigma_a+psi), epsilon=6,
       q=zeros(2,2), nu=1.0, r=nu;

  // Variable order: i(-1), q(-1), pic, y, pi, q

  e[3][<0; 4>]          = -1/sigma_a~1/sigma_a;
  e[4][4]               = beta;
  e[5][<0; 4>]          = 1~-1;

  a[0][0]               = 1;
  a[1][5]               = 1;
  a[2][<1; 4; 5>]       = alpha/(alpha-1)~1~alpha/(1-alpha);
  a[3][3]               = 1;
  a[4][<3; 4>]          = -kappa~1;
  a[5][5]               = 1;

  b[0]                  = 1;

  c[0][3]               = 1;
  c[1][<1; 4; 5>]       = alpha/(alpha-1)~1~alpha/(1-alpha);

  d[0][0]               = 0.0;

  q[0][0]               = epsilon/lambda;
  q[1][1]               = 1+psi;

  return{e, a, b, zeros(6,1), c'q*c, c'q*d, d'q*d+r, beta, m};}
```

```
// Main routine ********************************************************

main()
 {decl e, a, b, c, d, q, u, r, beta, n, f, s, ys, us, xs, zs,
       z, nsim=10, m, aa, at, nt, vt, ft, ctg, ctg2, n11, n22, nn, i,
       n21, n12s, n22s, rho;
 format("%#12.5f");

 // Set up model
 [e, a, b, bb, q, u, r, beta, m] = modelGM1();
 // Taylor rule
 rho=0.25;
 ft=(rho-1)~0~0~(1-rho)*1.5~(1-rho)*2.5~0;

 print("\ne", e, "\na", a, "\nb", b, "\nq", q, "\nu", u,
       "\nr", r, "\nbeta", beta, "\nft", ft);

 i=rows(a); z=zeros(i,1); z[0] = 1;

 [at, nt] = calNG(e, a-b*ft, 3);
 print("\nTaylor policy:\nat", at, "\nnt", nt);
 [zs, xs, ys, us] = simANCDF(at, 0, 0, 0, nt, z[0:2], nsim);
 PlotRE(zs, xs, ys, us, 0, "taylor.gwg");

 // Hamiltonian solution and simulation
 print("\nHamiltonian simulation, rule returned");
 [aa, n, f] = TIHU(e, a, b, q, u, r, beta, m);
 nn=n[i-m:i-1][];
 s=CalS(q, u, r, nn, f, aa, beta);
 print("\naa", aa, "\nn ", n, "\nf ", f);
 [zs, xs, ys, us] = simANCDF(aa, 0, 0, f, nn, z, nsim);
 ctg=cabs(wl(zs, s));
 ctg2=cabs(wl(zs[0:i-m-1][], s[0:i-m-1][0:i-m-1]));
 PlotRE(zs, xs, ys, us, ctg|ctg2|cabs(ctg-ctg2), "optH.gwg");
 print("\nTime inconsistent (H): Loss =", z's*z/2);

 z = z[0:i-m-1];
 // FBN
 print("\nMethod for possibly singular models: FBN");
 [at,nt,ft,vt] = controlTCTE(e, a, b, beta, q, u, r, m, 0, 50000);
 [zs,xs,ys,us] = simANCDF(at, 0, 0, ft, nt, z, nsim);
 PlotRE(zs, xs, ys, us, wl(zs, vt), "fbn.gwg");
 print("\nn ", nt, "\nf ", ft, "\ns = ", vt);
 print("\nFBN: Loss = ", z'vt*z/2, "\n");
```

```
// FBS
print("\nMethod for possibly singular models: FBS");
[at,nt,ft,vt] = controlTCTE(e, a, b, beta, q, u, r, m, 1, 50000);
[zs,xs,ys,us] = simANCDF(at, 0, 0, ft, nt, z, nsim);
PlotRE(zs, xs, ys, us, wl(zs, vt), "fbs.gwg");
print("\nn ", nt, "\nf ", ft, "\ns = ", vt);
print("\nFBS: Loss = ", z'vt*z/2, "\n");  }
```

References

Blanchard, O. J. and C. M. Kahn (1980). The solution of linear difference models under rational expectations. *Econometrica 48*(5), 1305–1311.

Galí, J. and T. Monacelli (2002). Monetary policy and exchange rate volatility in a small open economy. NBER Working Paper No. 8905.

Söderlind, P. (1999). Solution and estimation of RE macromodels with optimal policy. *European Economic Review 43*, 813–823.