

# Solving the growth model by linearizing the Euler equations

Paul Klein

Department of Economics

University of Pennsylvania

November 1998

## 1 Introduction

The overall approach of this cookbook is the following. First we write down a prototypical model economy. Then we write down sufficient conditions for an equilibrium in this model which include a transversality condition. Then we linearize these conditions around the steady state and reformulate the transversality condition. The result is a system of linear expectational difference equations with an associated stability condition that is closely related to the transversality condition. Finally, we use state-of-art numerical linear algebra algorithms to find a stable recursive representation of the solution to the system of difference equations. Thanks to these algorithms, we can find the linear law of motion for the economy in an easier way than by value function iteration (as described for instance in Chapter 2 of Cooley's book).

A more detailed version of the method described here is given in my paper "Using the Generalized Schur Form to Solve a System of Expectational Difference Equations". (See <http://www.ssc.upenn.edu/~kleinp>). My approach is very closely related to those of Harald Uhlig and Bob King.

## 2 The problem

We want an linear approximation of the recursive representation of the solution to a representative agent growth model. If the agent's problem has the required concavity properties, we can easily derive sufficient conditions, which consist of some Euler equations and a transversality condition. Typically the representative agent's problem can be written as

$$\max_{\{d_t, k_{t+1}\}_{t=0}^{\infty}} E_0 \left[ \sum_{t=0}^{\infty} \beta^t u(z_t, K_t, k_t, D_t, d_t) \right] \quad (1)$$

$$\text{s.t.} \quad k_{t+1} = g(z_t, k_t, d_t) \quad (2)$$

where the agent is also aware of the aggregate laws of motion and decision rules

$$z_{t+1} = \Gamma z_t + \varepsilon_{t+1} \quad (3)$$

$$D_t = H(z_t, K_t) \quad (4)$$

$$K_{t+1} = G(z_t, K_t, D_t) \quad (5)$$

where we assume that  $k_0$  is a given vector,  $z_0$  is a given vector and  $\varepsilon_t$  is an exogenously specified white noise process. The agent also faces the constraint that her decisions do not anticipate the flow of information generated by  $\varepsilon_t$ . This together with the law of motion (5) implies that  $k_{t+1} - E_t k_{t+1} \equiv 0$  so that the value of the state is known a period in advance. Notice that we have specified the process for  $z_t$  so that  $E[z_t] = 0$ . This involves no loss of generality.

To derive the Euler equations, we can write each term of the Lagrangian as follows

$$\begin{aligned} \mathcal{L}_t = & u(z_t, K_t, k_t, D_t, d_t) + \lambda_t^T k_{t+1} + \\ & \beta u(z_{t+1}, K_{t+1}, k_{t+1}, D_{t+1}, d_{t+1}) - \beta \lambda_{t+1}^T g(z_{t+1}, k_{t+1}, d_{t+1}) \end{aligned} \quad (6)$$

where  $\lambda_t^T$  denotes the vector of Lagrange multipliers associated to constraints (2) in period  $t$  transposed. Next, take the first order conditions with respect to  $d_t$  and  $k_{t+1}$  and then set

$D_t = d_t$  and  $K_t = k_t$  (the representative agent condition). Define  $f_i$  as the matrix of partial derivatives (gradient) of a function  $f$  with respect to its  $i$ th arguments. We get

$$\underbrace{u_5(z_t, K_t, K_t, D_t, D_t)}_{1 \times n_d} - \underbrace{\lambda_t^T}_{1 \times n_k} \underbrace{g_3(z_t, K_t, D_t)}_{n_k \times n_d} = 0 \quad (7)$$

and

$$\underbrace{\lambda_t^T}_{1 \times n_k} + \beta \underbrace{u_3(z_{t+1}, K_{t+1}, K_{t+1}, D_{t+1}, D_{t+1})}_{1 \times n_k} - \beta E_t \left[ \underbrace{\lambda_{t+1}^T}_{1 \times n_k} \underbrace{g_2(z_{t+1}, K_{t+1}, D_{t+1})}_{n_k \times n_k} \right] = 0. \quad (8)$$

Combining equations (2) with (7) and (8) and using the compact notation  $U_t = [D_t^T \ \lambda_t^T]^T$ , the sufficient conditions can be written in column form as

$$E_t [f(z_{t+1}, K_{t+1}, U_{t+1}, z_t, K_t, U_t)] = \underbrace{0}_{(2n_k + n_d) \times 1} \quad (9)$$

where  $E_t$  is the expectations operator conditional on information known at time  $t$ . (Notice that at this stage, when we have derived the sufficient conditions, we can drop the distinction between aggregate and individual variables). Typically these Euler equations will have a straightforward economic interpretation.

The transversality condition will be replaced in this context by a *stability* condition, which says, roughly, that the unconditional expectations of the squares of things dated  $t$  don't blow up as  $t \rightarrow \infty$ . This stability condition turns out to imply the transversality condition, so our conditions remain sufficient for an equilibrium.

More precisely, the transversality condition is

$$\beta^t \lambda_t^T k_t \rightarrow 0 \text{ in } \mathcal{L}^2.$$

If the sequences  $\lambda_t$  and  $k_t$  are stable in the sense that there is an  $M$  such that  $E[\lambda_t^T \lambda_t] \leq M$  for all  $t = 0, 1, \dots$ , and similarly for  $k_t$ , then the transversality condition clearly follows.

Throughout, we will also assume that all variables are adapted to the natural filtration

of  $\varepsilon_t$ . This is just a formalization of the constraint that all decisions taken at  $t$  have to be based only on information known at  $t$ .

### 3 The solution

In what follows, we will be calling the following user-defined Matlab functions: `grad.m`, `reorder.m`, and `solab.m`. The codes can be found in the Appendix and may also be downloaded from my homepage.

#### 3.1 Linearizing

If you already know that the steady state of the exogenous variables (the unconditional mean in case of shocks) is zero then you proceed to linearize (9), once you find the steady state. To find it, just solve

$$f(0, K, U, 0, K, U) = 0$$

for  $(K, U)$  and then use numerical derivatives to find the linearization around the steady state. The steps involved are as follows.

First define the function  $f$  as a Matlab function and call it (say) `euler.m`. When defining your variables, make sure that you have reason to believe that the decision rule in terms of those variables really is close to linear. Typically this involves working with deviations of logs from their steady state values. Typically then, if your original variable is  $x_t$ , your adjusted variable is  $\ln x_t - \ln \bar{x}$ , where  $\bar{x}$  is the steady state of  $x_t$ . On occasion, we may suspect that the steady state of a variable may be negative; in this case we do not take logs. Abusing the notation somewhat, we will use the notation  $U_t$  for adjusted  $U_t$  etc.

Then define the value of the Euler equations at a steady state in the following way.

```
function y = sseuler(x)
point = [zeros(nz,1);x;zeros(nz,1);x];
y = euler(point);
```

Then define an initial guess of the steady state and call it `ini`. As the final step involved in solving for the steady state, execute the following command which solves the Euler

equations numerically.

```
ss = fsolve('sseuler',ini);
```

Next we linearize the Euler equations around the steady state. To do that, we use a numerical gradient function. There is a built-in function in Matlab called `gradient.m` that does not do what we want. However, you can download the one that does from my homepage (`grad.m`). The function is also given in the Appendix. We write

```
gra = grad('euler',ss);
```

Now define

$$W_t \equiv \begin{bmatrix} z_t \\ K_t \end{bmatrix} \quad \text{and} \quad X_t \equiv \begin{bmatrix} W_t \\ U_t \end{bmatrix}. \quad (10)$$

If we linearize around the steady state and consolidate with the law of motion for the exogenous state variables, we get

$$A E_t \{X_{t+1}\} = B X_t \quad (11)$$

or

$$A E_t \begin{bmatrix} z_{t+1} \\ K_{t+1} \\ U_{t+1} \end{bmatrix} = B \begin{bmatrix} z_t \\ K_t \\ U_t \end{bmatrix} \quad (12)$$

where

$$\underbrace{A}_{(nz+2nk+nd) \times (nz+2nk+nd)} = \begin{bmatrix} f_1(\bar{X}) & f_2(\bar{X}) & f_3(\bar{X}) \\ I_{nz} & 0 & 0 \end{bmatrix} \quad (13)$$

and

$$\underbrace{B}_{(nz+2nk+nd) \times (nz+2nk+nd)} = \begin{bmatrix} -f_4(\bar{X}) & -f_5(\bar{X}) & -f_6(\bar{X}) \\ \Gamma & 0 & 0 \end{bmatrix} \quad (14)$$

where we recall that the arguments of  $f$  are  $z_{t+1}$ ,  $K_{t+1}$ ,  $U_{t+1}$ ,  $z_t$ ,  $K_t$ , and  $U_t$ . Notice that there is no constant term, since we are working with deviations from the steady state.

Note that the first  $2nk + nd$  rows of  $A$  are just the first  $nz + 2nk + nd$  columns of `gra`, and similarly that the first  $2nk + nd$  rows of  $B$  are the negative of the last  $nz + 2nk + nd$  columns of `gra`. Notice that the matrices  $A$  and  $B$  have just as many rows and columns as

we have variables that we want to solve for. More specifically, we can define  $A$  and  $B$  as follows.

```
A = [gra(:,1:nz+2nk+nd)
      eye(nz) zeros(nz,2nk+nd)];
B = [-gra(:,1:nz+2nk+nd+1:end)
      GAMMA zeros(nz,2nk+nd)];
```

## 3.2 Solving difference equations

### 3.2.1 The scalar case

Consider

$$\alpha E_t[x_{t+1}] = \beta x_t. \quad (15)$$

If  $|\alpha| < |\beta|$ , it is a theorem that the only stable solution to this equation is  $x_t \equiv 0$ . This case includes  $\alpha = 0$ .

On the other hand, if  $|\alpha| > |\beta|$  the situation is rather different. Then the stability condition does not exclude anything, and instead we have to pin down the initial value and the expectations error by reference to auxiliary assumptions. If we have  $x_0$  and  $x_{t+1} - E_t x_{t+1} = \xi_{t+1}$  given, then there is a unique solution which is defined recursively via

$$x_{t+1} = \frac{\beta}{\alpha} x_t + \xi_{t+1}.$$

Note that this defines a stable process if, say,  $\xi_t$  is homoskedastic.

We do not consider the possibility  $|\alpha| = |\beta|$ . If  $|\alpha| = |\beta| = 0$  our equation becomes  $0 = 0$  which obviously is of no help in determining  $x_t$ . If  $|\alpha| = |\beta| \neq 0$ , then we have a unit root in which case there is no stable solution unless  $\xi_t \equiv 0$ .

### 3.2.2 The multivariate case

Since we now know how to deal with the scalar case, it would be great if we could diagonalize (11) and solve it row by row. It turns out that *triangularization* is just as good, and what's more it's always possible. Not only is it possible, there are super-efficient triangularization

algorithms written in Fortran by Bo Kågström and Peter Poromaa of Umeå University. You can download them for free from <http://www.netlib.org/lapack/>. The routines you need are ZGEGS and ZTGSEN. (If you use Matlab or Gauss, check my homepage for advice on what to do. You can find code and an example there too).

The following theorem (see Golub & van Loan: *Matrix Computations* for a proof) guarantees that all linear systems can be triangularized. Given any square matrices  $A$  and  $B$  there are unitary matrices  $Q$  and  $Z$  and upper triangular matrices  $S$  and  $T$  such that

$$\begin{cases} QA = SZ^H \\ QB = TZ^H. \end{cases} \quad (16)$$

where  $Z^H$  is the Hermitian transpose of  $Z$ . (Unitary means that  $Z^H Z = Z Z^H = I$ ; upper triangularity means that all the elements below the main diagonal are zero.)

To find such matrices, use the MATLAB command

```
[S,T,Q,Z] = qz(A,B)
```

This upper triangular factorization is not unique, indeed  $S$  and  $T$  can be arranged in such a way that the pairs of diagonal elements  $(s_{ii}, t_{ii})$  appear in any order we like. Our choice will be to arrange them in ascending order in the sense that if  $|s_{jj}| \cdot |t_{ii}| < |s_{ii}| |t_{jj}|$  then  $(s_{ii}, t_{ii})$  appears before  $(s_{jj}, t_{jj})$ . To do this, we need the functions `reorder.m` and `qzswitch.m` which appear in the Appendix. We write

```
[S,T,Q,Z] = reorder(S,T,Q,Z)
```

We now partition our matrices conformably with the partitioning  $\begin{bmatrix} W_t \\ U_t \end{bmatrix}$ . We write

$$Z = \begin{bmatrix} Z_{11} & Z_{12} \\ Z_{21} & Z_{22} \end{bmatrix} \quad (17)$$

and similarly for the other matrices. We will also write

$$Z^H = \begin{bmatrix} Z_{11}^H & Z_{12}^H \\ Z_{21}^H & Z_{22}^H \end{bmatrix}. \quad (18)$$

The coding is as follows.

```
z21 = z(nw+1:end,1:nw);
z11 = z(1:nw,1:nw);
s11 = s(1:nw,1:nw);
t11 = t(1:nw,1:nw);
```

Note that in this code, `nw` represents the number of elements in  $W_t$ , i.e.  $nz + nk$ . Note also that `S11`, `T11`, `Z11` and `Z21` are the only matrices we need in order to find the solution; see below.

At this stage we must check if we have the right number of stable eigenvalues in order to pin down the solution. What we need is that the number of eigenvalue pairs with  $|s_{ii}| < |t_{ii}|$  is the same as the number of elements of  $U_t$ . (In an undistorted competitive equilibrium this is certainly the case. See Hansen & Sargent's manuscript for a proof. If the economy is distorted, anything can happen. See Farmer (1993).) To check this, we execute the following commands.

```
if abs(t(nw,nw)) > abs(s(nw,nw)) | abs(t(nw+1,nw+1)) < abs(s(nw+1,nw+1));
    warning('Wrong number of stable eigenvalues.');
```

Note that if we do have the right number of unstable eigenvalues, then  $S_{11}$  is invertible by construction. We need to invert  $Z_{11}$  as well, however, so we check its invertibility and then calculate its inverse. The required commands are

```
if rank(z11) < nw;
    error('Invertibility condition violated')
end
z11i = z11 \ eye(nw);
```

At this stage we are in a position to state and prove a representation result for the solution of our difference equation. The calculations involved are carried out by executing the code

`[F,P] = solab(A,NB,nk)`

which takes matrices **A** and **B** and the number of state variables **nk** as inputs and gives the matrices **F** and **P** in the following representation of the stable solution as outputs.

$$\begin{cases} U_t &= FW_t \\ W_{t+1} &= PW_t + \xi_{t+1}. \end{cases} \quad (19)$$

In the remainder of this section, we discuss the derivation of mathematical expressions for **F** and **P**. The basic result is the following.

**Theorem 1** *If the eigenvalue and invertibility conditions are satisfied and there are no unit roots, there is a unique stable solution to (11) and a recursive representation of this solution is given by*

$$\begin{cases} U_t &= Z_{21}Z_{11}^{-1}W_t \\ W_{t+1} &= Z_{11}S_{11}^{-1}T_{11}Z_{11}^{-1}W_t + \xi_{t+1}. \end{cases} \quad (20)$$

where  $\xi_{t+1} = \begin{bmatrix} 0 \\ \varepsilon_{t+1} \end{bmatrix}$ .

**Proof.**

(Sketch.) Define

$$y_t = Z^H X_t \quad (21)$$

and partition

$$y_t = \begin{bmatrix} y_{1t} \\ y_{2t} \end{bmatrix} \quad (22)$$

conformably with the partition of  $X_t$ . Note that since  $Z^H$  is nonsingular, the stability of  $X_t$  is equivalent to the stability of  $y_t$ . Now premultiply equation (11) by  $Q$ . Since  $Q$  is nonsingular, this involves no loss of information. Our equation becomes

$$\begin{bmatrix} S_{11} & S_{12} \\ 0 & S_{22} \end{bmatrix} E_t \begin{bmatrix} y_{1,t+1} \\ y_{2,t+1} \end{bmatrix} = \begin{bmatrix} T_{11} & T_{12} \\ 0 & T_{22} \end{bmatrix} \begin{bmatrix} y_{1,t} \\ y_{2,t} \end{bmatrix} \quad (23)$$

By construction, the matrix pair  $S_{22}$  and  $T_{22}$  have only unstable eigenvalue pairs, i.e. pairs  $(s_{ii}, t_{ii})$  with  $|s_{ii}| < |t_{ii}|$ . But then any stable solution must have

$$y_{2,t} \equiv 0. \quad (24)$$

This has three implications, namely that

$$E_t [y_{1,t+1}] = S_{11}^{-1} T_{11} y_{1,t}, \quad (25)$$

that

$$Z_{11}^H W_t + Z_{12}^H U_t = 0, \quad (26)$$

and that

$$Z_{11} W_t = y_{1,t}. \quad (27)$$

Together with the unitary nature of  $Z$  and the invertibility of  $Z_{11}$ , the result follows after some tedious algebra. ■

If you want an intuitive way to think about the result, the first equation means that we are on the saddle path; it corresponds to the familiar result from the theory of difference equations that a stable solution must be in the space spanned by the eigenvectors associated with the stable eigenvalues. The second equation says that the dynamics of the state are governed by the stable eigenvalues only (notice that  $T_{22}$  and  $S_{22}$  are not there).

## A Codes

### A.1 grad

```
function grdd = grad(f,x0,para1,para2,para3);
%
% Purpose: To calculate the gradient of the function f at the point x0
% Format:
% If f takes parameters, then the format is
%   gradient = grad('f',x0,para1,para2,para3)
% If not, the format is
```

```

%   gradient = grad('f',x0)
% where f is a function and f(x) is an mxn matrix
% x0 is a kx1 matrix
%
% Output:  the gradient, an mxk matrix defined as in
% Magnus and Neudecker (1988).
if ~isreal(x0);
    if abs(imag(x0))>eps;
        error('Not implemented for complex matrices');
    else
        x0 = real(x0);
    end
end
if nargin==2;
    f0 = vec(feval(f,x0));
end
if nargin==3;
    f0 = vec(feval(f,0,para1));
end
if nargin==4;
    f0 = vec(feval(f,x0,para1,para2));
end
if nargin==5;
    f0 = vec(feval(f,x0,para1,para2,para3));
end
n = size(f0,1);
k = size(x0,1);
grdd = zeros(n,k);
ax0 = abs(x0);
if ~(x0==0);
    dax0 = x0./ax0;
else
    dax0 = 1;
end
dh = 10^(-8)*max([ax0,(1e-2)*ones(k,1)]')'.*dax0;
xdh = x0+dh;
dh = xdh-x0;
res = x0*ones(1,k);
arg = diag(xdh) + res - diag(diag(res));
for i = 1:k;
    if nargin==2;
        grdd(:,i)=vec(feval(f,arg(:,i)));
    end
    if nargin==3;

```

```

    grdd(:,i)=vec(feval(f,arg(:,i),para1));
end
if nargin==4;
    grdd(:,i)=vec(feval(f,arg(:,i),para1,para2));
end
if nargin==5;    grdd(:,i)=vec(feval(f,arg(:,i),para1,para2,para3));
end
end
grdd = (grdd-f0*ones(1,k))./(ones(n,1)*dh');

```

## A.2 qzswitch

```

function [A,B,Q,Z] = qzswitch(i,A,B,Q,Z)
% Written by Chris Sims
% Takes U.T. matrices A, B, orthonormal matrices Q,Z, interchanges
% diagonal elements i and i+1 of both A and B, while maintaining
% QAZ and QBZ unchanged. Does nothing if ratios of diagonal elements
% in A and B at i and i+1 are the same. Aborts if diagonal elements of
% both A and B are zero at either position.
a = A(i,i); d = B(i,i); b = A(i,i+1); e = B(i,i+1);
c = A(i+1,i+1); f = B(i+1,i+1);
wz = [c*e-f*b, (c*d-f*a)'];
xy = [(b*d-e*a)', (c*d-f*a)'];
n = sqrt(wz*wz');
m = sqrt(xy*xy');
if n == 0
    return
else
    wz = n\wz;
    xy = m\xy;
    wz = [wz; -wz(2)', wz(1)'];
    xy = [xy; -xy(2)', xy(1)'];
    A(i:i+1,:) = xy*A(i:i+1,:);
    B(i:i+1,:) = xy*B(i:i+1,:);
    A(:,i:i+1) = A(:,i:i+1)*wz;
    B(:,i:i+1) = B(:,i:i+1)*wz;
    Z(:,i:i+1) = Z(:,i:i+1)*wz;
    Q(i:i+1,:) = xy*Q(i:i+1,:);
end;

```

### A.3 reorder

```
function [s,t,q,z] = reorder(s,t,q,z)
% Takes U.T. matrices S, T, orthonormal matrices Q,Z, rearranges them
% so that abs(T(i,i)/S(i,i)) are in ascending order w
% while preserving U.T. and orthonormal properties and QAZ and
% QBZ.
n = size(s,1);
i = 1;
while i<=n-1;
    if abs(t(i,i)*s(i+1,i+1))> abs(s(i,i)*t(i+1,i+1));
        [s,t,q,z] = qzswitch(i,s,t,q,z);
        if ~(i==1);
            i = i-2;
        end
    end
    i=i+1;
end
```

### A.4 solab

```
function [f,p] = solab(a,b,nk);
% Written by Paul Klein
% Purpose: Solves for the recursive representation of the stable solution
% to a system of linear difference equations.
% Inputs: Two square matrices a and b and a natural number nk
% a and b are the coefficient matrices of the difference equation
%  $a*x(t+1) = b*x(t)$ 
% where  $x(t)$  is arranged so that the state variables come first,
% nk is the number of state variables.
% Outputs: the decision rule f and the law of motion p. If we write
%  $x(t) = [k(t);u(t)]$  where  $k(t)$  contains precisely the state variables, then
%  $u(t) = f*k(t)$  and
%  $k(t+1) = p*k(t)$ .
% Calls: reorder
global lambda
[s,t,q,z] = qz(a,b); % upper triangular factorization
% of the matrix pencil b-za
[s,t,q,z] = reorder(s,t,q,z); % reordering of generalized eigenvalues in
% ascending order

lambda = [diag(s) diag(t)];
z21 = z(nk+1:end,1:nk);
z11 = z(1:nk,1:nk);
```

```

z11i = z11\eye(nk);
if rank(z11)<nk;
    error('Invertibility condition violated')
end
s11 = s(1:nk,1:nk);
t11 = t(1:nk,1:nk);
if abs(t(nk,nk))>abs(s(nk,nk)) | abs(t(nk+1,nk+1))<abs(s(nk+1,nk+1));
    warning('Wrong number of stable eigenvalues.');
```

## References

- [1] Farmer, R. "The macroeconomics of self-fulfilling prophecies," MIT Press.
- [2] Golub and van Loan: *Matrix Computations*, Johns Hopkins.
- [3] My homepage, <http://www.ssc.upenn.edu/~kleinp>
- [4] Hansen & Sargent's manuscript, <http://riffle.stanford.edu/hansen.html>
- [5] LAPACK, <http://www.netlib.org/lapack/>