Instruction Manual: Relaxation Algorithm

Supplement to "Trimborn, Koch and Steger (2008)"

Version 3.1

Timo Trimborn

June 2008

1 Introduction

This instruction describes how to simulate the transition process of a wide range of infinite horizon optimization models. It supplements the working paper of Trimborn, Koch and Steger (2007) and the associated MatLab files (Version 3.1). The researcher who intends to simulate a specific model only has to provide the dynamic system and the set of underlying parameters.[1]

2 Installation

There are two different types of files, the system files and the user files. The system files are general to every model and should be copied into a separate folder, which has to be assigned as a MatLab search path (menu *File* → *Set Path*). These files should not be modified. The user files carry the information due to a particular model and should be copied into a model-related folder, which has to be assigned as the MatLab *Current Directory*.

3 Preparation of the model

First of all, the dynamic system under study must be transformed into stationary variables, i.e. into variables that approach a finite constant asymptotically as time goes to infinity. It is assumed that now the model consists of n ordinary differential equations, $n_1 (\leq n)$ initial conditions and (optional) n_3 static equations that have to be satisfied at all times.[2] Notice that you do not have to modify these equations. They are plugged into the algorithm as they are written in continuous time. Only the static equations (if present in the model) must be rewritten as a residual ($g(y) = 0$).[3] You should go through the following steps carefully.

[1] The code allows for non-autonomous differential equations as well as consideration of additional equations, which have to be satisfied at all points in time. Moreover, there are many applications for the algorithm beyond the application presented here. To implement these, some more modifications are required.

[2] For convenience, we define $N := n + n_3$.

[3] For some algorithms it is necessary to differentiate the static equations with respect to time. This is not necessary for the application of the relaxation procedure.

4 Structure of the Code

The aim of the structure of the code is not only to make standard applications comfortable, but also to allow for advanced applications, e.g. multiple simulations of the same model with different initial conditions or parameter values. Therefore, in this section the structure of the code and the usage of the two most important functions *initrelax.m* and *relax.m* will be described in more detail. Note, that for convenience some files are double among the user files and the system files. The reason is that files in the current directory have priority when Matlab is executing them. Therefore it is possible to place empty files in the system folder to keep the number of files in the user directory small and concise.

The central file of the code is the file *main.m*, which calls the different scripts and functions in the following order:

globalpar	initializes the global parameters
parini	loads the values of the parameters
relaxsetting	loads the settings for the relaxation procedure
initrelax	converts the settings into a form suitable for *relax.m*
relax	executes the Relaxation algorithm
varex	disentangles the variables and stores them in the memory

The actual relaxation algorithm and the code to prepare the model for the relaxation algorithm are exported to functions and are not integrated into *main.m* as a script. The reason for this is that as a script they would be able to manipulate any variables in the memory. If they are implemented as functions their interactions with model specific variables in the memory is kept as small as possible. Detailed information about the functions *initrelax.m* and *relax.m* can be listed by calling *help initrelax* and *help relax* at the MatLab command window. They are used according to

```
[guess, start, errorcode] = initrelax(@funcODE, @funcSTAT, n,
 n1, n3, nu, y, M, statev, t)

[t, x] = relax(@funcODE, @funcSTAT, @funcINI, @funcfinal, n,
 n1, n3, nu, y, M, start, Endcond, maxit, tol, damp, dampfac)
```

Output from *initrelax.m* are the variables *guess*, *start*, and *errorcode*. The first two are again input arguments for *relax.m*, the last is a code indicating $error = 0$ for the case of no error and $error > 0$ for different errors that the code detected in the model set-up. Output of *relax.m* is a row vector t and a matrix x. Each column of x is the list of the variable values at the point of time of the corresponding entry of t. The file *varex.m* disentangles the variables, such that each variable is a row vector with corresponding time vector t.

Input for both functions are different parameters, which will be described in more detail below. Further, input arguments for both functions are different function handles, which pass the information about a specific function name. These functions have the following shape:

function	input arguments	output arguments
@funcODE	time, vector of variables (dim N)	vector of RHS of ODE (dim n)
@funcSTAT	time, vector of variables (dim N)	vector of residuals of static equations (dim n_3)
@funcINI	time, vector of variables (dim N)	vector of residuals of initial boundary conditions (dim n_1)
@funcfinal	time, vector of variables (dim N)	vector of residuals of final boundary conditions (dim $n - n_1$)

These functions are among the system files with the above names but may well be replaced by more user specific files.

5 Simulation of a model

The model dependent files are the following:

globalpar.m
varex.m
parini.m
relaxsetting.m
ODE.m

with the optional files

StatEq.m
shock.m
initbound.m
finalbound.m

Go through the following steps carefully.[4]

- **File globalpar.m**
  List all the names of the parameters with the command *global*.
- **File varex.m**
  List all the variables like in the original with an increasing index. You have to bring the variables in a special ordering. First write down all the state variables then all the control (or adjoint) variables and finally all variables, for which the static equations should hold. If you have an auxiliary variable that does not appear in the static equations you have to place it at the very end.
- **File parini.m**
  Assign all parameter values which are valid at the new steady state.

[4]Note, that you cannot name any variable or parameter with x.

- **relaxsetting.m**
  - Specify the dimensions:
    n1 number of initial boundary conditions
    (equals the number of state variables, for instance,
    n1=1 in the case of the RCK model)
    n number of differential equations
    n3 number of static equations to be solved simultaniously
    M number of mesh points
  - *normal* specifies, which variables you want to normalize to 1 at the steady state. The vector [1 3 6] for example will normalize variables 1, 3 and 6. If you do not want to normalize any variable return an empty vector [ ].
  - You have to provide a rough guess for the steady state. Here the same ordering for the variables should be maintained as for the file *varex.m*.
  - You can induce a transition process by reducing one of the state variables compared to its new steady state value. E.g. $statev(1) = .5$ halves the value of the first state variable. If you do not enter anything, the initial values of the state variables will be calculated due to a shock in parameter values as specified in *shock.m*. If you assign $statev = 0$, you can specify initial conditions in the file *initbound.m* (see below).
  - Change the last section of the file only if problems occur.
    Endcond: Specifies the final conditions. The vector [1 3 6] for example will set the RHS of differential equations 1, 3 and 6 to zero. Default setting is to use the differential equations of the control (or adjoint) variables. You should mind the dimensions when changing this. If you set $Endcond = 0$ you can enter the final boundary conditions in the file *finalbound.m* (see below).
    tol: This declares the tolerance for the Newton procedure.
    maxit: Maximum number of iterations
    nu: This denotes the parameter for time transformation. Higher values allocate the mesh more towards the beginning of time, while lower values increase the density around the steady state. A good value for *nu* depends crucially on the speed of convergence towards the steady state. It should be chosen such that it minimizes the maximum change of the variables between the mesh points. For models with a half-life of about $50 - 80$ time units, a value of $nu = 0.04$ is advisable.
    damp: You can dampen the Newton procedure by the dampening factor *damp*. It will be multiplied with the factor *dampfac* in every iteration until it equals 1. A value of 1 does not damp, values between 0 and 1 increase the convergence radius of the Newton procedure but also lead to slower convergence and therefore more iterations.

- **File ODE.m**
  List the right-hand-sides of your differential equations here. Assign them in vector form to *funcODE* as in the original. You should mind the ordering, since the first differential equation should belong to the first variable and so on. In the non-autonomous case time is denoted as t.

The following files are optional. If they are needed, create them in the model-related current directory.

- **File StatEq.m**
  List the residuals of your static equations here. Assign them in vector form to *funcSTAT* as in the original. Time is again denoted as t.

- **File shock.m**
  In this file you can specify a specific shock in the parameter values. Assign (e.g. in an *if* loop) the old steady state values to the time $t = -1$ and the new steady state values to all other t. It is also possible to investigate continuous parameter changes or interior (expected) jumps in the parameters.

- **File initbound.m**
  If you want to employ variable initial boundary conditions, enter them here and set $statev = 0$ in the file *relaxsetting.m*. Assign the residuals to the vector *funcINI*. Mind that you need $n1$ equations.

- **File finalbound.m**
  If you want to employ final boundary conditions different to the RHS of the differential equations, enter them here and set $Endcond = 0$ in the file *relaxsetting.m*. Assign the residuals to the vector *funcfinal*. Mind that you need $n - n1$ equations.

After running the code by entering *main*, the variables are stored in the memory with their assigned names. For plotting the results you should consider the time vector t.

6 Additional functions

There are a number of additional functions that are placed among the system files. These functions simplify the analysis and presentation of dynamic models, and their designation of arguments is in accordance with the files of the relaxation algorithm.

6.1 Plotting

You can view the results quickly by calling *plotrelax(t,x,n1,time)*, whereas t and x are output from *relax.m*. The results will be plotted for the time span $(t_0, time)$. For further information call *help plotrelax*.

6.2 Calculation of eigenvectors and eigenvalues

Eigenvalues and eigenvectors of the linearized system at the steady state can be evaluated by the function *eigDAS.m*. This routine can also handle differential algebraic systems. The routine derives the Jacobian matrix at a specific point numerically, and calculates eigenvectors and eigenvalues at this point numerically. For further information call *help eigDAS*.

6.3 Forth order relaxation

While the function *relax.m* employs a second order procedure, it is possible to increase the order to forth degree, only by tripling the computational effort.[5] This is achieved by extrapolation.[6] The idea of extrapolation is to solve the same problem twice with a different number of mesh points. Since both solutions are of order two, the local error reduces by order two and can be calculated by comparing both solutions. Then, by a recombination of both solutions, the local error of order two can be eliminated, such that only lower order errors remain. This is conducted by the procedure *relax4.m*, which can be handled exactly in the same way as *relax.m*. Internally, *relax4.m* calls *relax.m* twice and calculates a forth order solution, which is then the output of *relax4.m*. For further information call *help relax4*.

Note that the precision of the solution is only one aspect, since the mesh has to be dense enough to plot results or compute utility integrals. Our experience is that the second order procedure *relax.m* is precise enough for most economic applications. However, if you require a more precise representation of a solution trajectory on a moderate mesh grid, we recommend to employ *relax4.m*.

6.4 Error control

By employing extrapolation, we can estimate the relative error of the second order procedure. The function *relax2e.m* can be handled in the same way as *relax.m*, but it delivers an estimation of the relative error in addition. This comes in expense of a tripled computational effort. The procedure employs extrapolation as described above. For further information call *help relax2e*.

References

Ascher, U. M., and L. R. Petzold (1998): *Computer Methods for Ordinary Differential Equations and Differential-Algebraic Equations*. Philadelphia : Society for Industrial and Applied Mathematics.

Trimborn, T., K.-J. Koch, and T. M. Steger (2008): "Multi-Dimensional Transitional Dynamics: A Simple Numerical Procedure," *Macroeconomic Dynamics*, 12(3), 1–19.

[5] If the number of mesh points is increased by factor x, the error reduces by factor x^2 and x^4 for a second and forth order procedure, respectively.

[6] See e.g. Ascher and Petzold (1998, pp. 207).