SFB 649 Discussion Paper 2008-034

JBendge: An Object-Oriented System for Solving, Estimating and Selecting Nonlinear Dynamic Models

Viktor Winschel*
Markus Krätzig**

* Universität Mannheim, Germany
** Humboldt-Universität zu Berlin, Germany

This research was supported by the Deutsche Forschungsgemeinschaft through the SFB 649 "Economic Risk".

http://sfb649.wiwi.hu-berlin.de
ISSN 1860-5664

SFB 649, Humboldt-Universität zu Berlin
Spandauer Straße 1, D-10178 Berlin

JBendge: An Object-Oriented System for Solving, Estimating and Selecting Nonlinear Dynamic Models [1]

Viktor Winschel, Markus Krätzig [a,b]

[a] ***winschel@rumms.uni-mannheim.de***, *University of Mannheim, Germany, phone: +49- 621-181-1839*

[b] ***mk@mk-home.de***, *Humboldt-Universität zu Berlin, Germany, phone: +49-30-2093 5725*

Abstract

We present an object-oriented software framework allowing to specify, solve, and estimate nonlinear dynamic general equilibrium (DSGE) models. The implemented solution methods for finding the unknown policy function are the standard linearization around the deterministic steady state, and a function iterator using a multivariate global Chebyshev polynomial approximation with the Smolyak operator to overcome the course of dimensionality. The operator is also useful for numerical integration and we use it for the integrals arising in rational expectations and in nonlinear state space filters. The estimation step is done by a parallel Metropolis-Hastings (MH) algorithm using a linear or nonlinear filter. Implemented are the Kalman, Extended Kalman, Particle, Smolyak Kalman, Smolyak Sum, and Smolyak Kalman Particle filters. The MH sampling step can be monitored and controlled interactively by sequence and statistics plots. The number of parallel threads can be adjusted to benefit from multiprocessor environments.

JBendge is based on the framework JStatCom, which provides a standardized application interface. All tasks are supported by an elaborate multi-threaded graphical user interface (GUI) with project management and data handling facilities.

Keywords: Dynamic Stochastic General Equilibrium (DSGE) Models, Bayesian Time Series Econometrics, Java, Software Development
JEL classification: C11, C13, C15, C32, C52, C63, C68, C87

[1] This research was supported by the Deutsche Forschungsgemeinschaft through the SFB 649 Economic Risk. JBendge is published under the General Public License (GPL). Binary and source distributions are available from *jbendge.sourceforge.net*.

1 Introduction

Specifying, solving and estimating dynamic macroeconomic models is an interactive multi-step procedure that combines many advanced numerical methods. The implementation in JBendge follows the methodology presented in Winschel and Krätzig (2008). Apart from innovations in the solution and estimation procedures, a distinguishing feature of our software is its highly interactive sampling environment. Our view is that the sampling process can greatly benefit from user interaction during the burn-in phase.

Existing popular toolkits for solving and estimating DSGE models, for example Juillard (1996), are mostly based on Matlab. In our project instead, we have chosen Java as the implementation language for various reasons. Firstly, this gave us the opportunity to use a modern development methodology based on well-established software engineering principles like object-orientation, unit-testing, version control, and design patterns (Gamma et al., 1995), which help to deal with the complexity and the size of the project. Furthermore, we could use an integrated development environment (IDE) to manage the rather large code base together with helpful tools for debugging and profiling. Also, Java offers multi-threading capabilities which are needed to run parallelized algorithms concurrently on several CPUs and to offer the user a responsive GUI even while expensive calculations are running. This feature is essential for achieving interactivity during running computations. The requirements for such a software are summarized in a rather general way in Krätzig (2006), and it is obvious that specialized number crunchers do not offer the feature set to meet all of these criteria. The choice of Java also allows to execute JBendge on almost any platform without the need to recompile or to adjust it in any way. This is especially important, because end-users will often run it on the Windows platform, while high-performance clusters are typically run under some Unix OS.

In the following we will describe the structure of the software as well as its use to economics as well as purely statistical applications. It should be mentioned that JBendge is an open source project and that interested researchers can easily access and modify the source code. Furthermore, due to its highly modular structure, the software can be extended in various ways. Especially, adding new filters and approximation methods is rather straightforward with the existing interfaces.

The paper is organized as follows. Section 2 describes the model class that can be solved and estimated, Section 3 gives an overview of the model solution steps, Section 4 discusses interactive estimation features, Section 5 describes general features of JBendge, and Section 6 concludes.

2 Model Specification

2.1 An Example

As an example for the used model class we take the standard real international business cycle model for N infinitely lived agents. The model has no distortions and a social planner decides on the allocation by solving the dynamic optimization

$$\max_{\{\{c_{n,t},l_{n,t},i_{n,t}\}_{n=1}^{N}\}_{t=0}^{\infty}} U = \mathrm{E}_0 \sum_{t=0}^{\infty} \sum_{n=1}^{N} \beta^t U_{n,t}$$

for $n = 1, \ldots, N$ countries and all future periods $t \geq 0$. The welfare function U is a discounted sum of country utilities

$$U_{n,t} = \frac{(c_{n,t}^{\theta_n}(1 - l_{n,t})^{1-\theta_n})^{1-\tau_n}}{1 - \tau_n}$$

with discount factor β, elasticity of intratemporal substitutions τ_n and consumption and leisure substitution rates θ_n. The policy variables are consumption $c_{n,t}$, labor $l_{n,t}$ and investment $i_{n,t}$ for each country. The world budget constraint

$$\sum_{n=1}^{N} (y_{n,t} - c_{n,t} - i_{n,t}) = 0$$

restricts the world output $\Sigma_n y_{n,t}$ to be either consumed or invested in one of the countries. The production technologies

$$y_{n,t} = e^{a_{n,t}} k_{n,t}^{\alpha_n} l_{n,t}^{1-\alpha_n}$$

depend on productivities $a_{n,t}$, capital $k_{n,t}$ and labor $l_{n,t}$ and the technical substitution rates α_n. The capital and productivity transitions are

$$k_{n,t+1} = (1 - \delta_n) k_{n,t} + i_{n,t} - 0.5 \kappa_n i_{n,t}^2 \tag{1}$$

$$a_{n,t+1} = \rho_n a_{n,t} + e_{n,t+1} \tag{2}$$

where δ_n is the depreciation rate and ρ_n the autocorrelation coefficient for the productivity process with normally distributed shocks $e_{n,t} \sim \mathcal{N}(0, \sigma_{e_n})$

independent across countries and time. The capital transition equation (1) contains a capital adjustment cost parameterized by κ_n. These costs assure that in the multicountry model the state of the system is not simply the aggregate capital stock but its distribution across the countries.

2.2 A General Modelclass

The modelclass in JBendge is the same as the one used in the CompEcon Toolbox (Miranda and Fackler, 2002). The following functions have to be specified to define a model

$$\begin{aligned} 0 &= f(s_t, x_t, z_t; \theta) \\ z_t &= \mathrm{E}_e h(s_t, x_t, e_{t+1}, s_{t+1}, x_{t+1}; \theta) \\ s_{t+1} &= g(s_t, x_t, e_{t+1}; \theta). \end{aligned}$$

with the function types

- $f : \mathbb{R}^{d_s+d_x+d_z} \to \mathbb{R}^{d_x}$ - first order equilibrium conditions
- $h : \mathbb{R}^{d_s+d_x+d_e+d_s+d_x} \to \mathbb{R}^{d_z}$ - functions for rational expectations
- $g : \mathbb{R}^{d_s+d_x+d_e} \to \mathbb{R}^{d_s}$ - state transitions

and variable types

- $s_t \in S \subseteq \mathbb{R}^{d_s}$ - states
- $x_t \in X \subseteq \mathbb{R}^{d_x}$ - policies
- $z \in Z \subseteq \mathbb{R}^{d_z}$ - expected variables
- $e_t \sim \mathcal{N}(0, \Sigma_e)$ - stochastic state shocks
- θ - structural parameters

The shocks are assumed to be independent, thus, only the diagonal of Σ_e is specified. E_e denotes the expectation operator with respect to the distribution function of the random vector e and the information set given at time t.

In addition to the minimal specification required to solve a model, JBendge uses the following functions and variables:

$$\begin{aligned} y_t &= m(s_t, x_t, z_t; \theta) + u_t \\ r_t &= r(s_t, x_t, z_t; \theta) \end{aligned} \tag{3}$$

with the function types

- $m : \mathbb{R}^{d_s+d_x+d_z} \to \mathbb{R}^{d_m}$ - measurement equations
- $r : \mathbb{R}^{d_s+d_x+d_z} \to \mathbb{R}^{d_r}$ - Euler error functions

and variable types

- $y_t \in M \subseteq \mathbb{R}^{d_m}$ - measurements
- $u_t \sim \mathcal{N}(0, \Sigma_u)$ - additive stochastic measurement shocks
- $r_t \in R \subseteq \mathbb{R}^{d_r}$ - Euler errors

The measurement variables y_t correspond to observable data series and are used in the parameter estimation step. It should be noted that u_t enters as an additive term, because the proposed filters used for the likelihood evaluation are valid only for additive measurement shocks. Euler errors are normalized, economically interpretable estimates of the approximation error (Judd, 1992). They can be used to evaluate the quality of the function approximations and numerical integrations in terms of economic quantities.

In addition to the mentioned function and variable types, JBendge allows to specify definitions that help to split long equations into more readable terms. Definitions are automatically expanded before any terms are evaluated.

The model class matches with the example given in the previous section in the following way

$$\begin{aligned} s_t &= \{a_{n,t}, k_{n,t}\}_{n=1}^N \\ x_t &= \{c_{n,t}, l_{n,t}, i_{n,t}\}_{n=1}^N \\ e_t &= \{e_{n,t}\}_{n=1}^N \\ \theta &= \{\tau_n, \kappa_n, \theta_n, \alpha_n, \delta_n, \rho_n, \beta, \sigma_{e_n}\}_{n=1}^N \end{aligned} \tag{4}$$

The functions f correspond to $2N-1$ first order conditions arising from the Bellman equation, the functions g are the N state transitions given in equations (1) and (2), and the functions h are the arguments to the expectation operator appearing in the Euler equations. For a detailed derivation, see Winschel and Krätzig (2008).

2.3 The Model Parser in JBendge

JBendge offers a model parser that automatically recognizes variables and functions together with their corresponding types according to a number of ad hoc conventions. This should make the model definition as convenient as

possible for the user. In general, functions can be defined with the usual arithmetic expressions. It should be noted that 'e' is a reserved symbol for the Euler constant.

Every expression must be ended with a semicolon. A simple end of line character is not sufficient because this way long expressions can be split over separate lines. Variable names follow the same restrictions as in Matlab. They must start with a letter and the only allowed special character is the underscore. However, the suffix '_f' (always case insensitive) denotes leading variables and should not be used for anything else.

The following rules are applied in the same order by the parser to recognize the various types:

(1) symbols starting with lower case letters denote structural parameters, all variables must start with an uppercase letter
(2) expressions without '=' are recognized as first order conditions
(3) expressions with ':=' are definitions, the corresponding left hand side (LHS) variable gets the definition type
(4) expressions with '=' and a LHS variable:
    (a) if the LHS variable ends with '_f' then it is a leading state, the variable with the same name but without the '_f' suffix is the corresponding current state
    (b) if the LHS variable starts with 'Z' it gets the expectation type
    (c) if the LHS variable starts with 'R' it gets the (Euler) error type
    (d) all other LHS variables are of the measurement type
(5) for the remaining symbols not appearing as LHS variables:
    (a) symbols ending with '_f' denote a leading policy, the symbol with the same name but without the '_f' suffix is the corresponding current policy
    (b) symbols starting with 'E' are state shocks
    (c) symbols starting with 'M' are measurement shocks

Listing (1) shows the example model for $N = 1$ in the described format. The above described variable and function types are automatically recognized. For example, the symbol $Y1$ is of the definition type, which means that it is internally replaced by its right hand side expression. The symbols $Ym1, Km1, Cm1$ and $Lm1$ are parsed as measurement variables with the corresponding measurement shocks. The last expression specifies the Euler error $R1$. Here one error function is defined, but zero or more are possible as well.

It is recommended that the parser rules of JBendge are followed, but users also have the possibility to use the model specification GUI to redefine variable and function types. We argue that by means of the model parser it is especially convenient to specify even large and complex models. Furthermore, because

```
C1^((1 - tau1) * theta1 - 1) * (1 - L1)^((1 - tau1) * (1 -
    theta1)) - beta * Z1;
K1_f = Inv1 + (1 - delta1) * K1;
A1_f = Ea1 + A1 * rho1;
Z1 = C1_f^((1 - tau1) * theta1 - 1) * (1 - delta1 + e^A1_f *
     alpha1 * (K1_f * L1_f^(-1))^(alpha1 - 1)) *  (1 - L1_f)
    ^((1 - tau1) * (1 - theta1));
Ym1 = My1 + Y1;
Km1 = K1 + Mk1;
Cm1 = C1 + Mc1;
Lm1 = L1 + Ml1;
Invm1 = Inv1 + Minv1;
Inv1 := Y1 - C1;
Y1 := e^A1 * K1^alpha1 * L1^(1 - alpha1);
Inv1_f := Y1_f - C1_f;
Y1_f := e^A1_f * K1_f^alpha1 * L1_f^(1 - alpha1);
C1 := e^A1 * (1 - alpha1) * (1 - L1) * (K1 * L1^(-1))^alpha1
     * (1 - theta1)^(-1) * theta1;
C1_f := e^A1_f * (1 - alpha1) * (1 - L1_f) * (K1_f * L1_f
    ^(-1))^alpha1 * (1 - theta1)^(-1) * theta1;
R1 = 1 - C1^(-1) * (beta * (1 - L1)^(-((1 - tau1) * (1 -
    theta1))) * Z1)^(((1 - tau1) * theta1 - 1)^(-1));
```

Listing 1. Example model in parseable format

it is almost impossible to detect small typos in the formulas when they are just presented as ASCII text, JBendge uses the excellent *jeuclid* math viewer library to visually display the model equations, see Figure (1). [1]

2.4 Specification of Shock Distributions and Parameters

After the equations and variables are defined, the model is usually solved at given parameter values for certain shock distributions. JBendge offers dialogs and parsers to conveniently input this information. A typical calibration for the example model is shown in Listing (2). This format can directly be parsed, alternatively, GUI components may be used.

To specify the shock distributions with the respective parameters, JBendge offers the component shown in Figure (2). As before, a parser recognizing the distribution specifications is provided. This parser is also used in other parts of the program, for example to input prior distributions in the estimation stage. Currently, the normal and the uniform distributions are supported. Listing (3) shows the syntax for the distribution parser. Internally, distributions are

[1] The homepage of the jeuclid project is *jeuclid.sourceforge.net*.

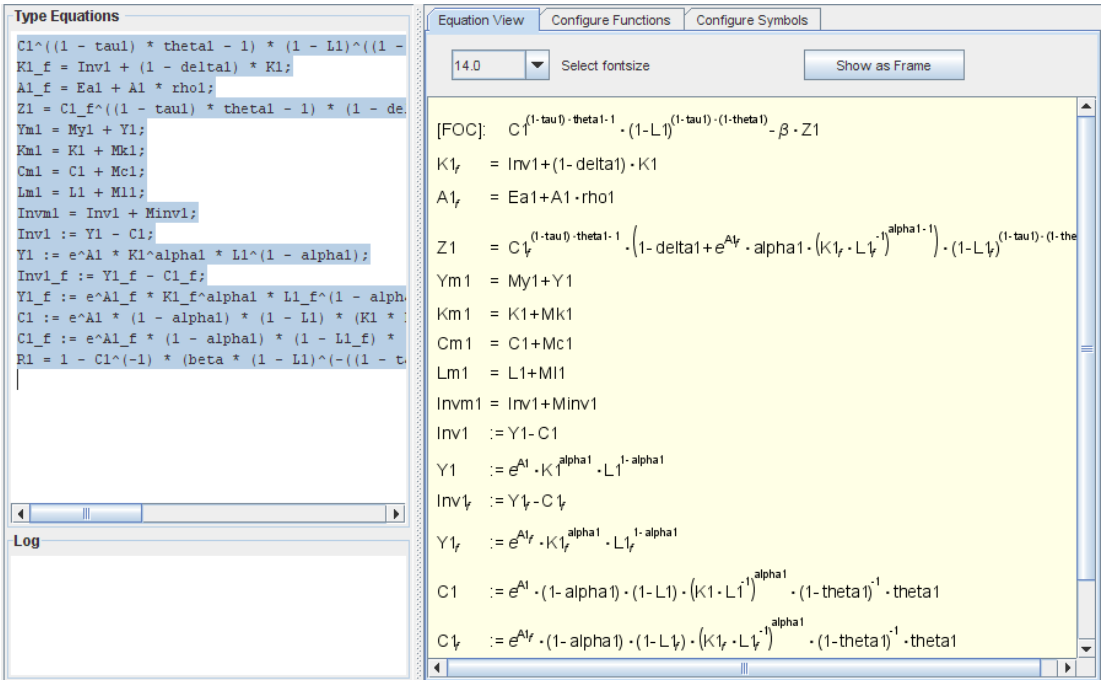

Fig. 1. Model specification in JBendge

```
alpha1=0.4;
beta=0.99;
delta1=0.02;
rho1=0.95;
tau1=2.0;
theta1=0.357;
```

Listing 2. Parameter parser

```
variable name : distribution type, Property = value,
    Property = value, ...;

// example:
alpha : UNIFORM, LBOUND=0.2, UBOUND=0.6;
k : NORMAL, MEAN=23.0, SIGMA=3.0;
```

Listing 3. Distribution syntax

represented by a single class with the combination of a type identifier and a parameter map. The type identifier is itself a class that provides the special implementations needed for all distribution specific tasks, like sampling or evaluating the density. Adding new distributions merely amounts to specifying the corresponding type identifier, which will then automatically fit into the existing framework, including the GUI and the parser.

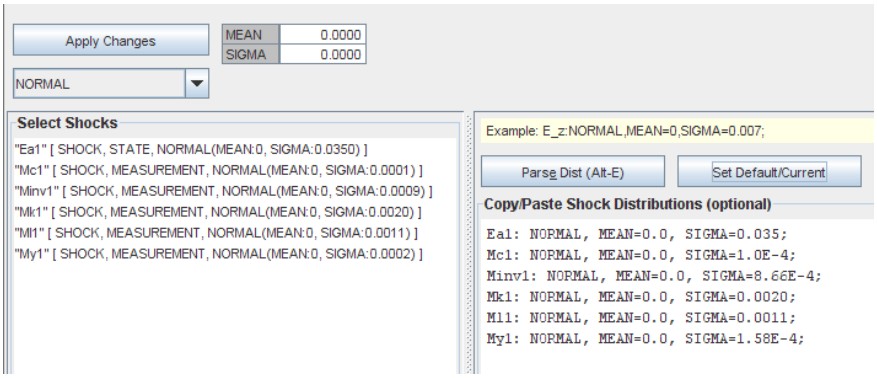

Fig. 2. Specification of shock distributions

3 Model Solution at given Parameters

The model solution at given parameters is often of interest, for example, for comparing simulated data from the calibrated model to real data series. Furthermore, solving the model at different parameters may help to understand its dynamic behavior. Another purely technical aspect is to check whether the model can be solved at all and whether it leads to a plausible solution. JBendge offers two general solution methods, linearization around a steady state, and a function iterator using Chebyshev polynomials to approximate the solution together with Gaussian quadrature to evaluate the integrals arising from rational expectations. In JBendge, the linear solution is used to provide starting values for the nonlinear solution.

3.1 Linear Solution around a Steady State

A solution to the model is a policy function $x(s)$ which takes the state variables as arguments. For the linear solution it is assumed that $x(s)$ is a linear function of s. JBendge uses the generalized real Schur decomposition based on a linearized version of the model, following the methodology suggested by Klein (2000). In a first step, the steady state is found by setting all shocks to zero and all future variables equal to its current values. This results in the nonlinear system

$$\begin{aligned} 0 &= f(\bar{s}, \bar{x}, h(\bar{s}, \bar{x}, 0, \bar{s}, \bar{x})) \\ \bar{s} &= g(\bar{s}, \bar{x}, 0) \end{aligned}$$

of N unknown steady states, $\bar{s}$ and $\bar{x}$, in N equations. To solve this system, JBendge uses a Newton root finder as described in Press et al. (2007). After-

wards, the system is linearized with a first order Taylor approximation around the given steady state values. It should be mentioned that JBendge uses a symbolic differentiation and simplification engine to avoid problems with the numerical approximation of derivatives. After these two steps the system is in the form

$$AE_t[z_{t+1}] = Bz_t, \quad t = 0, 1, \ldots \tag{5}$$
$$z_t = \begin{bmatrix} s_t \\ x_t \end{bmatrix}$$

where z_t is an $N \times 1$ vector of endogenous variables and $A_{N\times N}$ and $B_{N\times N}$ are coefficient matrices. It should be mentioned that any exogenous variables are endogenized which is not optimal for the performance of the solution algorithm if the number of truly exogenous variables is quite large. Under certain assumptions, especially when the number of stable generalized eigenvalues of the matrix pencil $Ay - B$ is equal to the number of states, the system (5) has a unique stable solution. In JBendge, this solution is found with the real Schur (or QZ) decomposition and the stability condition is checked. If the condition is violated, a log message is generated which is displayed in the *Control - Show Logs* panel.

3.1.1 Steady State

Because finding the steady state involves using a nonlinear root finder, starting values have to be provided by the user. This can be done with a GUI component or via the parser which takes input of the form `variable name = value`. If the steady state cannot be found or if it does not make sense, it may help to try different starting values.

In many cases analytical steady states are available, for the one country model with states a and k the steady states are

$$\bar{a} = 0$$
$$\bar{k} = -\frac{(\alpha-1)\alpha^{\frac{1}{1-\alpha}}\beta^{\frac{1}{1-\alpha}}(\beta(\delta-1)+1)^{\frac{\alpha}{\alpha-1}}\theta}{-\alpha\delta\beta+\delta\beta+\alpha\theta\beta-\beta-\alpha\theta+1}.$$

Using them may significantly speed up the linear solution as well as improve its numerical accuracy. Sometimes, analytical steady states are available for only

a subset of $\bar{s}$ and $\bar{x}$. In this case the complexity of the root finding problem can be reduced by eliminating some variables and reducing the number of nonlinear equations. JBendge allows to specify analytic steady states with a component that works similar to the model parser. All equations must be of the definition type, meaning that '`:=`' must be used. The LHS variables must be the states or policies without the '`_f`' suffix.

If an analytic steady state is set for a certain variable, it does not make sense to set its starting value anymore. Therefore the corresponding element in the GUI component is grayed out and displays the steady state value determined from the given definition. If not all steady states are defined, JBendge takes care of reducing the nonlinear system in such a way that its Jacobian remains nonsingular. The following algorithm is used:

(1) for the analytic steady states of the state variables the corresponding state transitions are removed
(2) for the analytic steady states of the policy variables:
    (a) all first order conditions that have no free variables anymore after substituting the analytic steady states are removed
    (b) the remaining first order conditions are sorted according to the number of free variables and those with the highest number of variables are removed (to make the Jacobian sparse), such that the number of free variables is equal to the number of equations

The output of the linear solution is given in Listing (4). It displays the values of all defined variables at the steady state and the linear policy functions state transitions and measurement functions. The linear measurement functions and state transitions together form a linear state space system.

3.2 Nonlinear Solution

Our solution approach is to solve for the policy functions $x^* : \mathbb{R}^{d_s} \to \mathbb{R}^{d_x}$ that map states into policies. The algorithm is a function iteration scheme where we repeatedly solve the first order conditions $f(s, x^{(k+1)}, \mathrm{E}_e h(..., x^{(k)}, ...)) = 0$ for the next $k+1$ iteration of the policy $x^{(k+1)}$ for given expected variables $z = \mathrm{E}_e h(..., x^{(k)}, ...)$ based on the previous policy x^k in iteration step k. This approach has the advantage that it decomposes one big system, when one solves for x in $f(s, x, \mathrm{E}_e h(..., x, ...)) = 0$, into several independent smaller ones. This means that for each grid point we solve one small system for all policies. The independence between the grids is achieved because $z = \mathrm{E}_e h(..., x^{(k)}, ...)$ does not depend on $x^{(k+1)}$ anymore and is constant during the search for the optimal policy at a given grid point. Another advantage is that analytical derivatives are available because the exact expressions for the involved

```
Steady State
------------

FOC 1:                          -3.20854e-14
A1 [STATE]:                      0.00000000
C1 [DEFINITION]:                 1.28563281
C1_f [DEFINITION]:               1.28563281
Cm1 [MEASUREMENT]:               1.28563281
Inv1 [DEFINITION]:               0.46536617
Inv1_f [DEFINITION]:             0.46536617
Invm1 [MEASUREMENT]:             0.46536617
K1 [STATE]:                      23.26830866
Km1 [MEASUREMENT]:               23.26830866
L1 [POLICY]:                     0.31210444
Lm1 [MEASUREMENT]:               0.31210444
R1 [ERROR]:                      0.00000000
Y1 [DEFINITION]:                 1.75099899
Y1_f [DEFINITION]:               1.75099899
Ym1 [MEASUREMENT]:               1.75099899
Z1 [EXPECTATION]:                0.91362113

Linear Solution
---------------

state transitions:
K1_f = 23.26830866399936 + 0.9737761225371353 * (K1 -
    23.26830866399936) + 1.8132706272447607 * A1
A1_f = 0.95 * A1 + Ea1

policy functions:
L1 = 0.3121044396634933 - 0.0020665798069341218 * (K1 -
    23.26830866399936) + 0.19583634163843358 * A1

measurement functions:
Cm1 = 0.6022795798303447 + 0.029368410144493817 * K1 +
    0.5969485265606075 * A1 + Mc1
Invm1 = 0.6101852751727221 - 0.00622387746286435 * K1 +
    1.8132706272447614 * A1 + Minv1
Km1 = K1 + Mk1
Lm1 = 0.3601902564900247 - 0.0020665798069341218 * K1 +
    0.19583634163843358 * A1 + Ml1
Ym1 = 1.212464855003067 + 0.023144532681629464 * K1 +
    2.410219153805369 * A1 + My1
```

Listing 4. Output of nonlinear solution

nonlinear functions are given by the first order conditions. If we would solve $f(s, x^{(k)}, \mathrm{E}_e h(..., x^{(k)}, ...)) = 0$ instead, this would result in one big and complicated system involving the multidimensional numerical integration at each evaluation of functions and derivatives, not to mention the impact on memory consumption for larger models. Our approach has been found to perform well for models of up to 30 states, furthermore the independence of the small systems allows a straightforward parallelization.

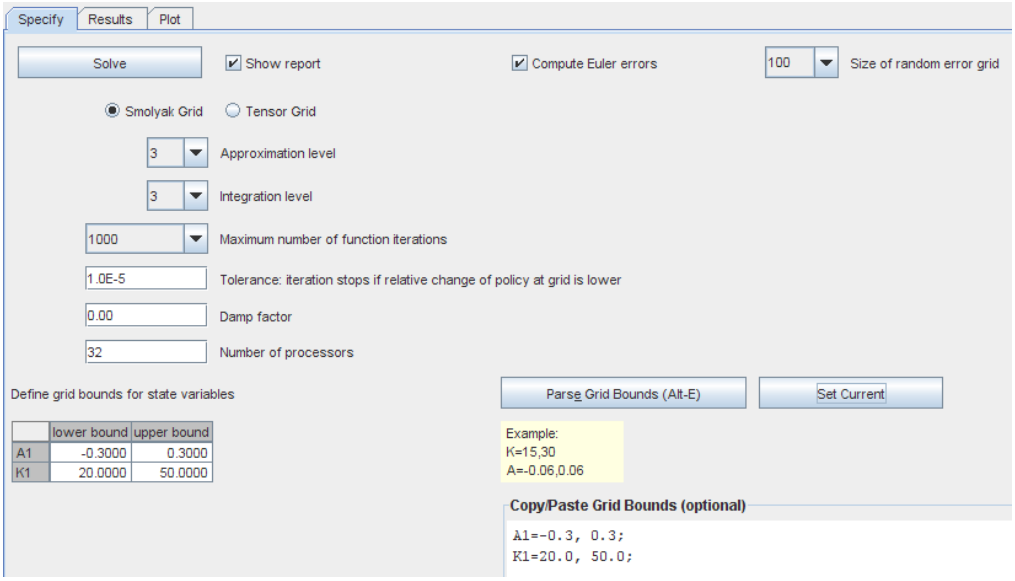

Fig. 3. Specification of nonlinear solution in JBendge

3.2.1 Approximation of the Policy Function

Unlike in pertubation approaches which rest on higher order approximations around the steady state, JBendge uses a global polynomial approximation of the unknown policy functions by using projection methods (Judd, 1992). Figure (3) shows the controls offered to the user. The nonlinear solution or policy function $x(s)$ resides in the infinite dimensional space of all functions. In a practical approximation for just a single state variable, we search in the m_i dimensional space of polynomials $\hat{x}(s;c) = \sum_{j=1}^{m_i} c_j b_{j-1}(s)$ characterized by the coefficient vector c. We use orthogonal Chebyshev polynomials as basis functions $b_j(s)$ defined by $b_0(s) = 1, b_1(s) = s$ and $b_{j+1}(s) = 2s\, b_j(s) - b_{j-1}(s)$ for $j \geq 1$. To identify the m_i elements of the coefficient vector c we use the same number of policy values at the grid $s^i = \{s_1^i, s_2^i, ..., s_{m_i}^i\}$.

JBendge uses the Gauss-Lobatto grid defined by $s_1^1 = 0$ and $s_j^i = -\cos(\pi(j-1)/(m_i-1))$ for $j = 1, ..., m_i$ and $i > 1$. The range of these points is from -1 to 1 and the grid in the desired approximation state space is obtained by the

simple linear transformation

$$\tilde{s}_j^i = a + \frac{b-a}{2}(s_j^i + 1),$$

where the lower and upper state bounds are a and b. In JBendge, the state bounds can be specified with a GUI component or with a parser. The bounds parser accepts the simple syntax

```
variable name = lower bound, upper bound;
```

If the functions to be approximated or integrated depend on several variables the univariate approximation operator must be extended to many dimensions. The usual extension applies the tensor operator which uses all possible combination of states. The resulting grid is the Cartesian product of the univariate grids. Clearly, this operator imposes an exponentially growing cost of approximation if the number of states increases. A cure for this problem is to use the Smolyak Operator, see Bungartz and Griebel (2004). This operator makes the states grid sparse, which results in a larger approximation. However, Winschel and Krätzig (2008) have shown for a number of different specifications that the performance gains are dramatic and that the approximation error is usually tolerable. JBendge offers both options, the Smolyak as well as the Tensor operator, but it should be mentioned that the latter is computationally infeasible except for very small models of up to 6 states.

The approximation level i must be set by the user. One should notice that the grid size increases exponentially with the approximation level, therefore typical choices are 2 or 3. A reasonable strategy is to choose it such that the approximation error is still tolerable. The Clenshaw-Curtis function $m_1 = 1$ and $m_i = 2^{i-1} + 1$ for $i > 1$ translates the approximation level i into the polynomial degree of the approximation. In JBendge the approximation level is the same for every dimension.

3.2.2 Integration of the Rational Expectations

In nonlinear models involving rational expectations the integral of the expectation operator must be evaluated. JBendge uses a multidimensional Gaussian quadrature for this. Since it suffers from the same curse of dimensionality as the multidimensional function approximation, the Smolyak operator is used again to keep the problem tractable also for larger models. The integration problem can be stated as

$$\mathrm{E}_e h(..., e, ...) = \int h(..., e, ...)p(e)de \approx \sum_j w_j h(..., e_j, ...),$$

where the continuous random variable e and its density $p(e)$ is essentially discretized into some realizations e_j with weights w_j. For the quadrature in JBendge it is assumed that the density $p(e)$ is normally distributed because this allows the generate the corresponding nodes and weights, e_j and w_j. However, the general approach is by no means restricted to normally distributed shocks, but could be extended to other distributions as well when needed.

Also for the integration problem the integration level i must be chosen. The level i determines the number of nodes and weights that are used in each dimension.

3.2.3 Function Iteration

The approach of finding the unknown optimal policy function can be summarized as follows

(0) find the initial policy: $x^{(0)}$ at grid s^i, JBendge uses the linear policy
(1) approximate the policy function at the states grid and find the Chebychev interpolant $c^{(k)} = B(s^i)^{-1}x^{(k)}$, where $B(s^i)^{-1}$ is the inverse of the matrix holding the Chebychev bases evaluated at the states grid
(2) rational expectations:
  (a) for all discrete shock realizations from the quadrature rule $j = 1, ..., J$
    (i) calculate the state transition: $s'_j = g(s^i, x^{(k)}, e'_j)$
    (ii) interpolate the next policy using the Chebychev interpolant: $x'_j = B(s'_j)c^{(k)}$
  (b) evaluate the expectations with Gaussian quadrature: $z = \sum_j w_j h(s^i, x, e'_j, s'_j, x'_j)$.
(3) function iteration:
  (a) solve for x^e in $f(s^i, x^e, z(x^{(k)})) = 0$ given $x^{(k)}$ with a rootfinder
  (b) iteration: $x^{(k+1)} = \alpha x^{(k)} + (1 - \alpha)x^e$
  (c) calculate the residual, JBendge applies the infinity norm (maximum absolute row sum): $R = ||x^e - x^{(k)}||_\infty$
(4) k=k+1, go to 1. until $R \approx 0$

It should be noted that the inverse of the Chebychev basis matrix at the states grid, $B(s^i)$, must be calculated only once because the grid does not change over the iterations. JBendge provides GUI controls for setting the residual tolerance as well as the maximum number of function iterations. The damping factor α may be changed as well from its zero default value, but usually this is not necessary. Furthermore, users have the option to change the number of processors, which sets the concurrent thread count used for the solution. In general it should be equal to the number of CPUs.

3.2.4 Checking the Approximation Error

It is of paramount importance to check the accuracy of the involved numerical approximations with appropriate error functions. The exact policy functions would imply zero residuals in the complete state space and any deviation is therefore due to the policy function approximation or the inaccurate numerical integration. Judd (1992) proposed to normalize the residual from the evaluation of the Euler equations for an economic interpretation. JBendge allows the user to specify the error functions as part of the model specification. For example, for the described model the Euler error r^E in terms of consumption is given by

$$\begin{aligned} r^c(s) &= c_n(s) - \left(\frac{\beta \mathrm{E}_t h(s, x^*(s), e, s', x^*(s'); \theta)}{(1 - l_n(s))^{(1-\theta_n)(1-\tau_n)} / (\kappa_n i_{n,t} - 1)} \right)^{\frac{1}{\theta(1-\tau)-1}} \\ \boldsymbol{s}' &= g(s, x^*(s), e') \\ r^E &= |r^c(s)/c_n(s)|. \end{aligned}$$

The corresponding specification for the model parser is given in the last line of Listing (1). A $\log_{10}$ error of -3 means that the utility loss due to the approximation is less than one per 1000 dollars.

When the error is specified, it still has to be decided at which points in the state space it should be calculated. JBendge offers the possibility to evaluate all specified error functions at N randomly sampled points inside the states bounds and reports the maximum error.

3.2.5 Output and graphical Analysis

Listing (5) shows the output for the nonlinear solution. The states grid for the two state variables k and a is shown together with the corresponding linear and nonlinear policies. The mean percentage difference between the linear and nonlinear policies give an idea of the significance of the nonlinearity in the model. An important check for the quality of the approximations is the error estimate. In the example listing, a $\log_{10}$ Euler error of -5 indicates that the loss due to the approximation is very low and that the solution is sufficiently accurate.

Figure (4) shows the GUI for plotting the nonlinear policy function as well as the Euler errors over certain intervals of the state variables. JBendge allows to easily specify 2D and 3D plots. Similar plot configuration dialogs are available for the linear solution for comparisons.

```
Nonlinear Solution
------------------

Number of Grid Points: 13

states grid                    linear pol.     nonlinear pol.
K1              A1             L1              L1
20.000000       -0.300000      0.260108        0.260256
20.000000        0.000000      0.318859        0.319295
20.000000        0.300000      0.377610        0.375159
24.393398        0.000000      0.309779        0.309836
35.000000       -0.300000      0.229109        0.229867
35.000000       -0.212132      0.246317        0.247862
35.000000        0.000000      0.287860        0.291565
35.000000        0.212132      0.329403        0.334654
35.000000        0.300000      0.346611        0.352108
45.606602        0.000000      0.265941        0.277379
50.000000       -0.300000      0.198110        0.209297
50.000000        0.000000      0.256861        0.272273
50.000000        0.300000      0.315612        0.335101   ...

mean difference between linear and nonlinear policy =
   2.0895587363838324 %

Number of function iterations: 76
Euler errors (over random states grid of size 100):
R1: 5.6651886125580475E-5

Computing time: 0.08 sec
```

Listing 5. Output of nonlinear solution

3.2.6 Model Simulation

After the model has been solved, linearly or nonlinearly, it is possible to simulate the resulting system and to generate data series from it. JBendge generates series for all states and measurements. These series may then be used as benchmark input for the model estimation to check how good the original parameter values are recovered. The model simulation uses the specification of the shock distributions for the random number generation and initializes all variables with their steady state values. To eliminate the effect of the initial conditions, a reasonable strategy would be to simulate a large number of observations and then take only the last fraction of it. Simulating the model from the linear solution might also give insights into how the state bounds for the nonlinear solution should be chosen. As a rough guide, JBendge prints the min and max bounds for the generated states series for that purpose.

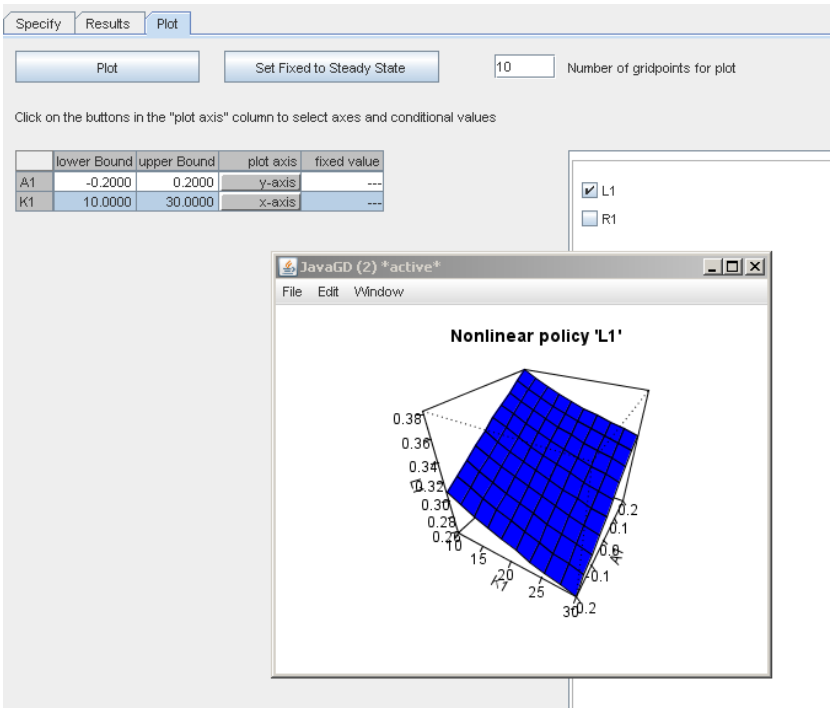

Fig. 4. Graphical analysis of nonlinear solution

4 Model Estimation

In JBendge, a likelihood based estimation of the structural parameters using the Metropolis-Hastings (MH) algorithm is implemented. The model estimation can be done based on the linear or the nonlinear solution of the model. This affects which filters are applicable for the calculation of the likelihood.

4.1 Linear Estimation

If the estimation is based on the linearized version of the model, we obtain the model's empirical implication for the observables in terms of a linear state space model

$$s_t = Bs_{t-1} + e_t$$
$$y_t = Hs_t + u_t,$$

where the first equation describes the state transition and the second is the measurement equation which relates the observables to the unobserved states. The noise processes e_t and u_t are assumed to be independent Gaussian with time invariant known covariance matrices. The matrices B and H are also known and nonstochastic for given structural parameter values. With the

Kalman (1960) filter it is possible to evaluate the likelihood of the complete sample, $\mathcal{L}(\theta; y_{1:T}) = p(y_{1:T}|\theta)$, for every parameter vector θ. Clearly, the model has to be solved for each evaluation to arrive at the linear state space representation.

The task is now to directly estimate the structural parameters of the model, which also includes the standard deviations of the shock distributions. A maximum likelihood approach would use some nonlinear optimization procedure to find the optimal parameter values. Typical problems with this approach are that the solution may only be a local maximum and that it may be difficult to use if the parameter space is very large. Reasonable starting values have to be provided for all parameters. Furthermore, only an asymptotic approximation of the distribution of the parameter estimates can be determined which may be misleading for the inference.

4.2 Interactive Metropolis-Hastings

In JBendge, a Bayesian estimation procedure with the Metropolis-Hastings sampler is implemented. The general task is to generate samples from the posterior distribution of the parameters, $p(\theta|y_{1:T})$. It is related to the likelihood and the prior density for the parameters, $p(\theta)$, by Bayes law

$$p(\theta|y_{1:T}) \propto p(y_{1:T}|\theta)p(\theta). \tag{6}$$

From the generated samples one may plot histograms and use the sample mean and the sample standard deviation for inference about the parameters.

The MH algorithm allows to generate a sequence of samples from a probability distribution that is difficult to sample from directly, see for example Chib and Greenberg (1995). In our case, an analytical expression is neither available for the likelihood nor the posterior. The MH procedure requires only that a function proportional to the target density can be evaluated. For the current problem, this function is given by Equation (6). The MH algorithm constructs a Markov chain that has the posterior distribution as its unique stationary distribution. If the Markov chain is converged then realizations from it can be regarded as samples from that distribution.

A summary of the basic algorithm is

(1) choose the starting value $\hat{\theta}_1$ and select the variance Σ_ϵ for an acceptance ratio of $\approx 30\%$
(2) for $n = 2$, while $n - J < N$, $n = n + 1$
  (a) generate the candidate: $\hat{\theta}^*_n = \hat{\theta}_{n-1} + \epsilon$, where $\mathcal{N}(\epsilon; 0, \Sigma_\epsilon)$.

(b) acceptance: $\hat{\theta}_n = \begin{cases} \hat{\theta}^*_n & \text{if } U(0,1) \leq \frac{p(y_{1:t}|\hat{\theta}^*_n)p(\hat{\theta}^*_n)}{p(y_{1:t}|\hat{\theta}_{n-1})p(\hat{\theta}_{n-1})} \\ \hat{\theta}_{n-1} & \text{otherwise} \end{cases}$

(c) decide on J by diagnostic tests

(3) disregard burn-in draws $\hat{\theta}_{1:J}$.

The parameter space is traversed by a random walk. If the acceptance ratio is tuned by the random walk variances to be around 30% we obtain after convergence a representative sample from the target distribution. The critical choices of the algorithm are the starting values $\hat{\theta}_1$, the density to generate candidates $\hat{\theta}^*_n$ and the number of draws N. The choice of $\hat{\theta}_1$ drives the number of draws before convergence. It can be set via the *Specify initial draws* panel and works in a similar way as the specification for the shock distributions. A similar component is available to specify the prior distributions for each structural parameter.

The chosen variances in Σ_ϵ influence the region covered by the sequence. Sampling around the mode of the posterior with large variances will generate candidates far from the current value and a low acceptance probability. Smaller variances increase the acceptance ratio but decrease the region being covered so that low probability regions may be undersampled. The recommended acceptance ratio of 30% results from the attempt to balance this trade of.

It is usually quite challenging to find good values for all elements of Σ_ϵ simultaneously. A common approach is to run costly training sequences and to estimate the variances from these runs. JBendge offers a much more convenient approach by using multiple sequences simultaneously and not sequentially. By that we can assure robustness with respect to the starting values, calculate unbiased convergence diagnostic tests, estimate the innovation variance on the fly and finally implement ideas from evolutionary algorithms to improve the search for the modus of the posterior in the beginning of the sampling. The pseudo code for this parallel Metropolis-Hastings Algorithm is given in in the following:

(1) choose the starting values $\hat{\theta}_{1,m}$ for all $m = 1, ..., M$ and select b, γ^{GE} for an acceptance ratio of $\approx 30\%$

(2) for $n = 2$, while $n - J < N$, n=n+1

(a) Repeat for $m = 1, ..., M$

(i) draw m_1 and m_2 such that $m_1 \neq m_2 \neq m$

(ii) generate the candidate: $\hat{\theta}^*_m = \hat{\theta}_{m,n-1} + \gamma^{GE}(\hat{\theta}_{m_1,n-1} - \hat{\theta}_{m_2,n-1}) + \epsilon$, $\mathcal{N}(\epsilon; 0, bI)$

(iii) acceptance:

$$\hat{\theta}_{m,n} = \begin{cases} \hat{\theta}^*_m & \text{if } U(0,1) \leq \frac{p(y_{1:t}|\hat{\theta}^*_m)p(\hat{\theta}^*_m)}{p(y_{1:t}|\hat{\theta}_m)p(\hat{\theta}_m)} \\ \hat{\theta}_{m,n-1} & \text{otherwise} \end{cases}$$

(b) decide on J by diagnostic tests

(3) disregard burn-in draws $\hat{\theta}_{1:J,1:M}$

The problem of choosing all variances of the random walk shocks is reduced in the proposed parallel variant to the choice of only two scalars b and γ^{GE}. In the estimation specification dialog shown in Figure (5), b is set with the *Innovation scaling* field and γ^{GE} with *Sequence mixing*. The number of sequences M can be set as well. A simple random walk without mixing can also be achieved by setting $\gamma^{GE} = 0$. It may also make sense to edit the relations of the diagonal entries in the innovation variance matrix directly with the corresponding editing component.

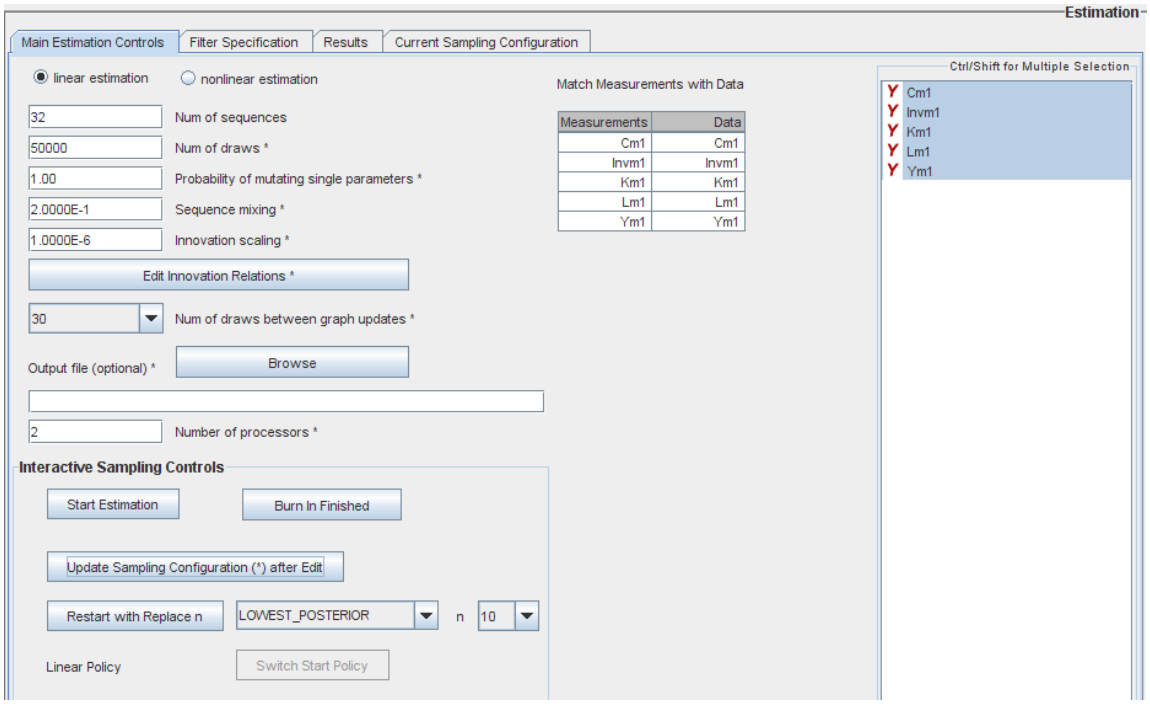

Fig. 5. Estimation specification

Another critical choice is J, the number of burn in draws. The strategy suggested by JBendge is to decide on it while looking at the generated sequences, see Figure (6), and on certain diagnostic tests during the running burn-in. The convergence tests are generated from the M parallel sequences. Brooks and Gelman (1998) proposed the multivariate potential scale reduction factor R as a diagnostic test. The general idea is to check within and between sequence variances and diagnose convergence if they are close to each other. The within-sequence-variance is the $D \times D$ matrix

$$W = \frac{1}{M(N-1)} \sum_{m=1}^{M} \sum_{n=1}^{N} (\hat{\theta}_{n,m} - \bar{\theta}_m)'(\hat{\theta}_{n,m} - \bar{\theta}_m)$$

where $\bar{\theta}_m = \frac{1}{N} \sum_{n=1}^{N} \hat{\theta}_{n,m}$ is the $1 \times D$ mean vector in sequence m. W is the

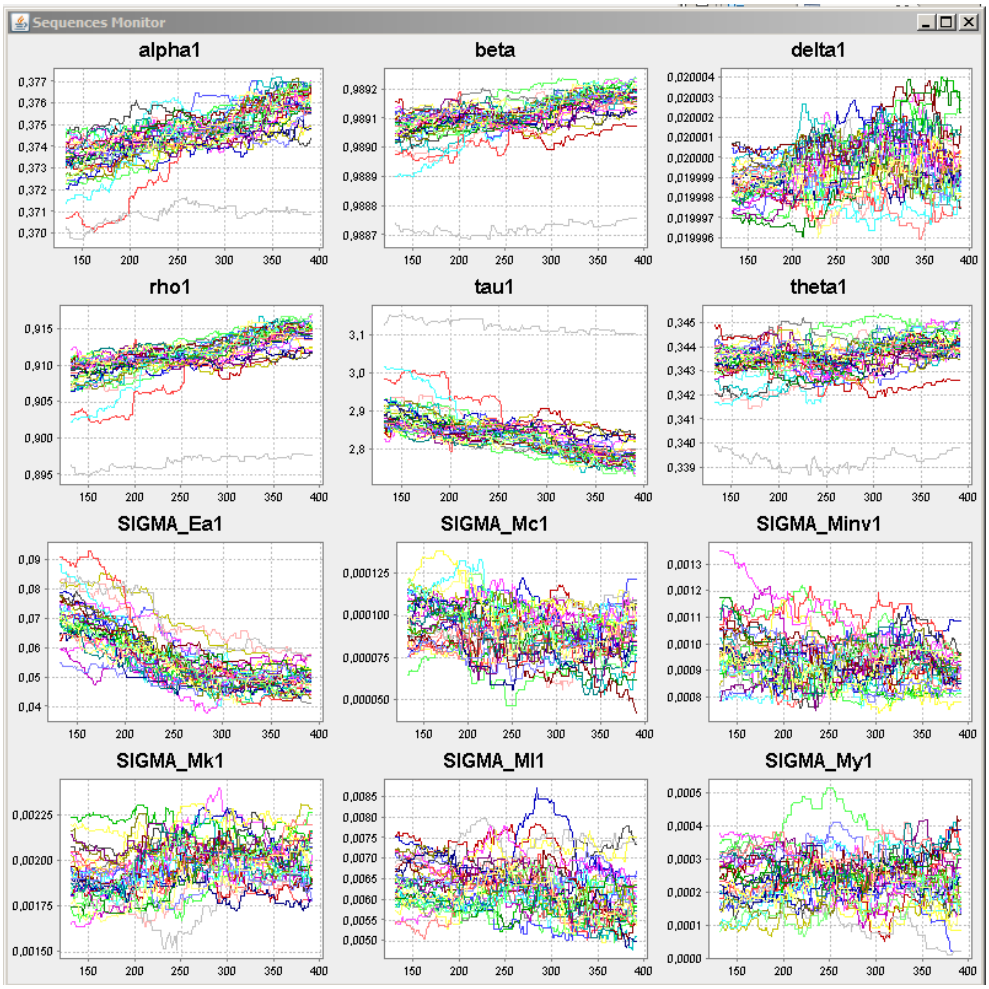

Fig. 6. Sequence monitor

mean of the variances in each sequence. The between-sequence-variance B/N is the $D \times D$ matrix

$$B/N = \frac{1}{M-1} \sum_{m=1}^{M} (\bar{\theta}_m - \bar{\theta})'(\bar{\theta}_m - \bar{\theta})$$

where $\bar{\theta} = \frac{1}{M} \sum_{m=1}^{M} \bar{\theta}_m$ is the $1 \times D$ mean of all draws. The combined variance can be estimated as

$$V = \frac{N-1}{N} W + (1 + \frac{1}{M}) B/N.$$

Convergence is detected for similar V and W. A distance measure is calculated by the multivariate potential scale reduction factor

$$R = \frac{N-1}{N} + \frac{M+1}{M} \lambda_{max} \qquad \text{where } \lambda_{max} = \max_a \frac{a'Va}{a'Wa}.$$

λ_{max} can be obtained by taking the largest absolute eigenvalue of $W^{-1}B/N$.

The following conditions for convergence should be checked: V and W should be similar and stabilize and R should be below 1.1.

In JBendge, these test statistics are calculated recursively after each draw and can be inspected with the *Sampling Monitor*, see Figure (7). Once the described conditions are met, users may interactively quit the burn-in process and start sampling by clicking the *Burn in finished* button, see Figure (5). The acceptance rate is displayed in the sampling monitor as well. To achieve a rate of about 30% one may adjust the parameter γ^{GE} accordingly. It should be increased if the rate is too high and vice versa.

A distinguishing feature of JBendge are the interactive controls during sampling. All settings that may be changed by the user without restarting the MH chain are tagged with the symbol '*'. The button *Update sampling configuration after edit* activates the changes. The effect of any changes can be seen in the updated monitor graphics. Although the graphics display in JBendge is very fast, too many graphics updates may negatively affect the sampling performance. Therefore, the update frequency may interactively be adjusted with the corresponding selection component.

4.2.1 The Metropolis-Hastings as a Global Optimizer

Experience of many sampling runs has shown that convergence occurs only if all sequences sample in regions close to the mode of the posterior distribution. This does not necessarily mean that the actual parameter draws are very close because it may happen that the posterior is rather flat in some directions. In any case, we found it a very useful strategy to use the parallel MH algorithm as a global optimizer in the first state of the burn-in process because it works similar to a genetic algorithm. ter Braak (2006) derives the same candidate generation process by analogy to the global evolutionary optimization algorithm, called differential evolution, by Storn and Price (1997).

The analogy to genetic algorithms also motivated the introduction of an interactive parameter p_m for the probability that single parameters are mutated. The candidate generation in the parallel MH algorithm is changed in the following way:

(1) generate the *potential* candidate: $\hat{\theta}^{**}_m = \hat{\theta}_{m,n-1} + \gamma^{GE}(\hat{\theta}_{m_1,n-1} - \hat{\theta}_{m_2,n-1}) + \epsilon$
(2) for all elements i of the candidate vector $\hat{\theta}^{*}_m$, do

$$\hat{\theta}^{*}_{m,i} = \begin{cases} \hat{\theta}^{**}_{m,i} & \text{if } U(0,1) \leq p_m \\ \hat{\theta}_{m,n-1,i} & \text{otherwise} \end{cases}$$

If $p_m = 1$ the original MH algorithm is replicated. The purpose of this algo-

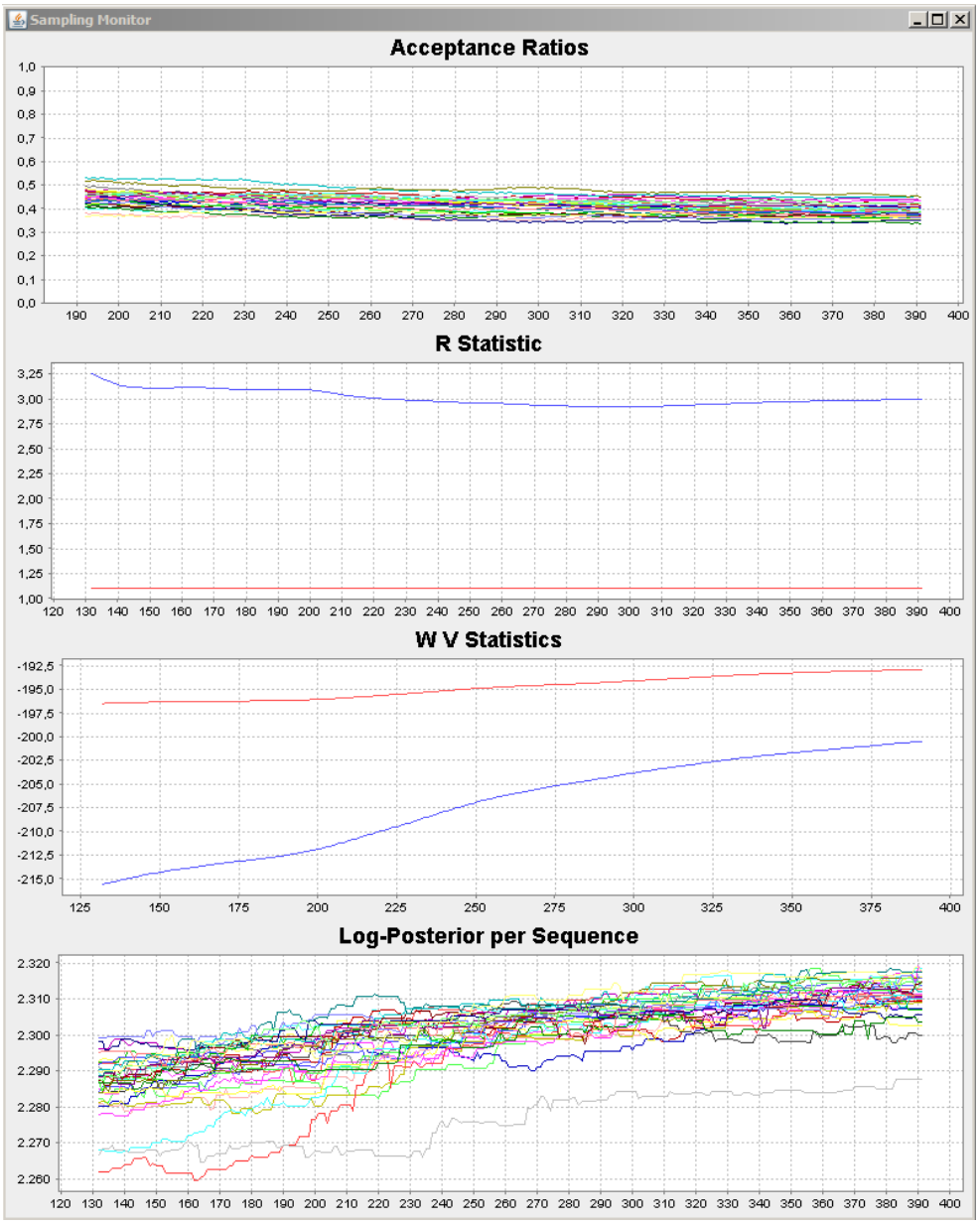

Fig. 7. Sampling monitor

rithm is to allow a more flexible candidate generation where not all elements of $\hat{\theta}$ need to change. If the single elements of $\hat{\theta}$ are viewed as its genes, then p_m steers the probability that the genes mutate. If $p_m = 1$ the additional random number draws are not needed which results in a notably better performance. However, it has been found that setting p_m to, say, 0.5 can help finding the mode of the posterior when the original MH seems to get stuck.

Another interactive feature of the parallel MH algorithm implemented in JBendge is the possibility to restart the Markov chain by replacing those n sequences either with the lowest posterior or the lowest acceptance rate. Restarting means that all sequences take their current value as the starting

value, except those n sequences with the lowest criterion value. These sequences attain their starting values from the remaining $M - n$ sequences with the highest criterion values. If n is larger than $M/2$ then the replacement values are recycled.

Restarting the MH algorithm can occasionally be very helpful to speed up the burn-in process and to replace sequences that are far away from the others. Theoretically, those sequences should converge as well but it may take a very long time.

4.3 Nonlinear Estimation

If the estimation of the structural parameters is based on the nonlinear solution, all settings previously set in the solution specification, like the states grid bounds and the approximation and integration levels, are inherited.

The model solving policy functions $x(s)$ can be used to substitute the policy variables in the state transition equations. Augmented with a measurement equation, we obtain the model's empirical implication for the observables in terms of a nonlinear state space model

$$\begin{aligned} s_t &= g(s_{t-1}, x(s_{t-1}), e_t) = g(s_{t-1}, e_t) \\ y_t &= m(s_t, x(s_{t-1})) + \epsilon_t = m(s_t) + \epsilon_t. \end{aligned}$$

It is well known that the standard Kalman filter should not be used to evaluate the likelihood of the nonlinear state space system because its optimality properties rest on the assumption of normality and linearity, which are both violated in this case. In JBendge there are five filters implemented which are eligible for nonlinear state space models

(1) Extended Kalman filter
(2) Particle filter
(3) Smolyak Kalman filter
(4) Smolyak Gaussian Sum filter
(5) Particle Smolyak Kalman filter

The first two of these filters are standard in the literature but with obvious shortcomings, see for example Arulampalam et al. (2002). The Extended Kalman filter applies the Kalman filter on a linearized version of the state space model which may result in biased estimates for the required moments. The Particle filter, on the other hand, estimates and updates the complete posterior density of the unobserved states represented by a sample of the states, called particles. This is done by a costly sequential importance sampling with

an inaccurate but simple to implement proposal density. This filter is computationally costly because the proposal density does not use the available information from the current observation and may therefore generate many useless draws. Moreover, as a Monte Carlo approach it does not use any information, like the smoothness, of the involved functions.

The remaining three filters have been proposed in Winschel and Krätzig (2008). All of them have in common that they extend existing filters with the Smolyak Gaussian quadrature.

The Smolyak Kalman filter assumes that the prediction $p(s_t|y_{1:t-1})$ and posterior densities $p(s_t|y_{1:t})$ are Gaussian. We therefore need to approximate only the first two moments of the densities and can then use the Kalman update in the filtering step. The moments needed for the Kalman step can be calculated as expected values of nonlinearly transformed random variables. Hence, we can use a deterministic Smolyak Gaussian quadrature for the approximation of these moments. This filter is very fast but the price may be an undesirably high approximation error.

The Smolyak particle filter improves the particle filter by combining it with the Smolyak Kalman filter. We use the posterior densities obtained by the Smolyak Kalman filter, represented by the deterministically integrated first two moments of the states, as a proposal density for the importance sampling of the particle filter. This procedure incorporates the latest information obtained from the data. It combines the advantages of both filters, the accurate but slow sampling particle filter and the potentially inaccurate but fast deterministic filter. It also uses a Gaussian assumption in one step.

Finally, the Smolyak Gaussian sum filter applies a Gaussian sum approximation (Kotecha and Djuric, 2003) of the involved densities. The filter is again very fast and purely deterministic. It effectively runs several Smolyak Kalman filters in parallel.

We have found little differences between these nonlinear filters for the nonlinear structural estimation, except from the extended Kalman filter.

Figure (8) shows the user interface for the filter specification of the estimation. All available filters can be selected from a drop down menu and the corresponding parameters may be set. For the quadrature based filters this involves setting the integration order for the integrations over the state and measurement errors. Typical values are 2 or 3. Particle filters also allow to set to number of particles to use. This figure should be much lower for the Particle Smolyak Kalman filter (1000-5000 particles) than for the standard Particle filter (20000-80000 particles). The Gaussian sum filter also allows to set the number of Gaussians involved in the approximation of the density, a typical value is 20.

The filter specification GUI also allows to evaluate the filters at certain values for the structural parameters. Furthermore, it is possible to do the filter evaluation at a grid over a single selected parameter, see the text output in Figure (8). This feature is helpful to investigate the shape of the likelihood in certain directions.

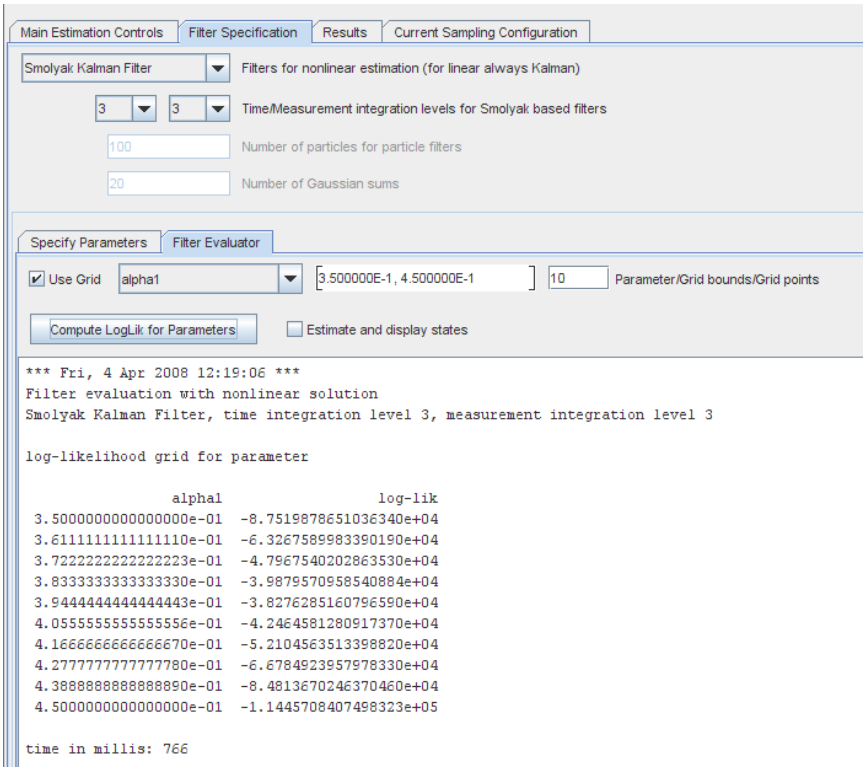

Fig. 8. Filter evaluator

4.4 Estimation output

The output of the estimation are the generated sequences which can be stored sequentially to a file. JBendge summarizes the sampled distributions in histogram plots for each parameter, see Figure (9). Furthermore, it prints textual output with the means, standard deviations, minima, maxima as well as the realizations corresponding to the maximum likelihood and maximum posterior values.

5 General Features

JBendge inherits many useful features from the underlying software framework JStatCom. Any software that is built with that framework can automat-

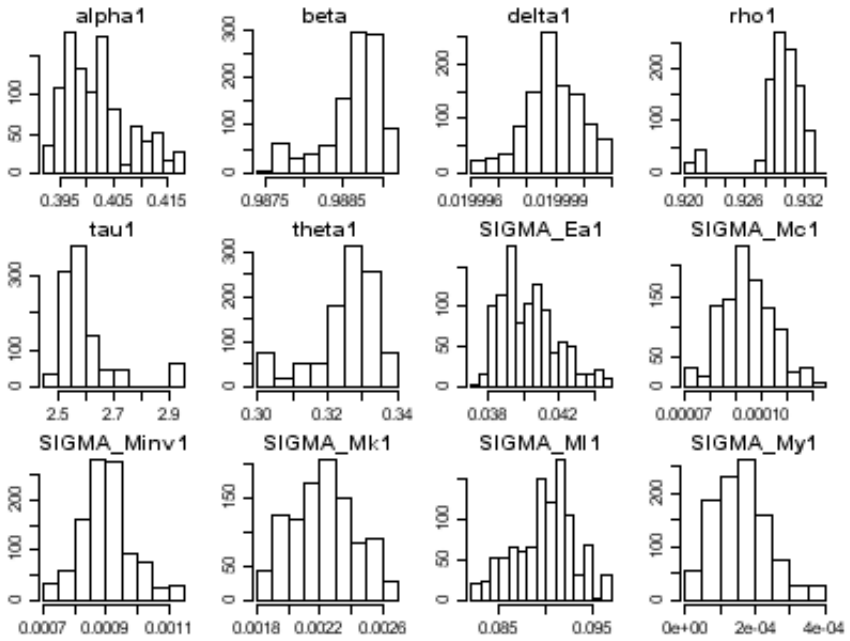

Fig. 9. Estimation output

ically use the data import and export functions, the overall application frame, project management, the symbol control, the time series calculator and other subsystems, see Krätzig (2006) for a detailed description. Most inherited functions can be changed or simply switched off. Framework driven development avoids reinventing the wheel, however, the framework must fit well in the problem domain (Deutsch, 1989).

5.1 Project Management

JBendge provides a project management system that stores all relevant model settings together with the time series data in a single project file with the default suffix `.jsc`. Project files are just gzipped XML files, hence, they are human-readable after unzipping. However, it is in general not convenient to edit them manually. For that purpose JBendge offers a simple ASCII format that allows to specify all relevant model settings in a single file, see Section 5.2.

A JSC project file in JBendge represents a state of the system. There can only be one project loaded at the time, but it is easy to switch between different projects. This behavior is different from other programs, for example Eviews, where multiple workfiles can be loaded. However, we believe that this behavior simplifies user interaction.

At startup, JBendge restores its last state. The corresponding project is au-

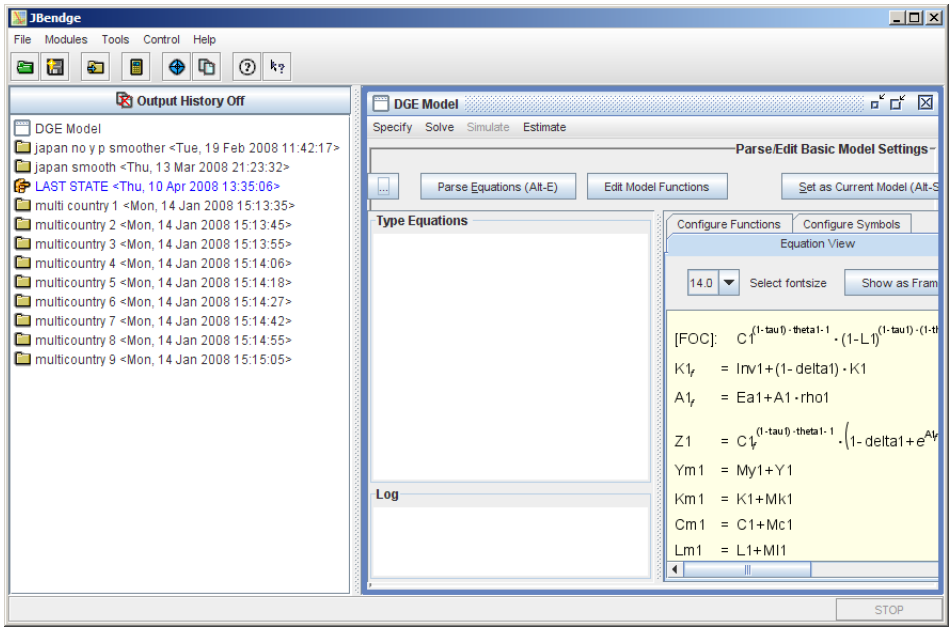

Fig. 10. JBendge project explorer

tomatically generated in the user's home directory and named *Last State*. Whenever a state should be permanently saved, either a new project should be created for it or the current project settings are overwritten. Figure (10) shows the main window of JBendge with the project and module explorer panel on the left. Switching between projects just requires a mouse click. JSC project files make it especially convenient to share models together with specific settings and data. It is even possible to simply drag JSC files with the mouse and drop them over the explorer panel.

5.2 Model Definition File

An important feature is the option to specify all relevant equations and parameters in a single file that is easy to create. This supports the typical usage scenario where the model equations are automatically generated by a computer-algebra system, like Mathematica or Maple. Listing (6) displays the respective file format for JBendge. It consists of sections that are separated by identifiers with a leading `$` sign. Inside the sections the parser rules described earlier apply. For example, in the `$ModelSpec` section all model equations are recognized according to the rules described in Section 2.3. The following identifiers can be used:

- `$ModelSpec` - model equations
- `$Parameters` - parameter values
- `$ShockDist` - shock distributions
- `$SteadyStatesAnalytic` - analytic steady states

- `$SteadyStateStartVals` - start values for root finder used by steady state solver
- `$Priors` - prior distributions for estimation
- `$InitDraws` - distributions for initial draws of estimation
- `$StatesGridBounds` - grid bounds for the states, required by nonlinear solution.

The identifiers are case insensitive and can be in any order. Except the `$ModelSpec` section, all other sections are optional. The model specification is required because it is used internally to instantiate an object of the class *DGEModel*, without which the other settings cannot be meaningfully interpreted. Furthermore, the model specification must adhere to the rules with which JBendge automatically recognizes variable and function types.

5.3 Symbol Control

Another JStatCom feature that is available in JBendge is the symbol control system. Figure (11) shows the GUI that displays all internally used data objects. Users may select symbols to display their contents and to export them in various formats. In the displayed example the coefficient matrices for the linear solution are displayed. Those can be exported, for example, to a binary Matlab `.m` file.

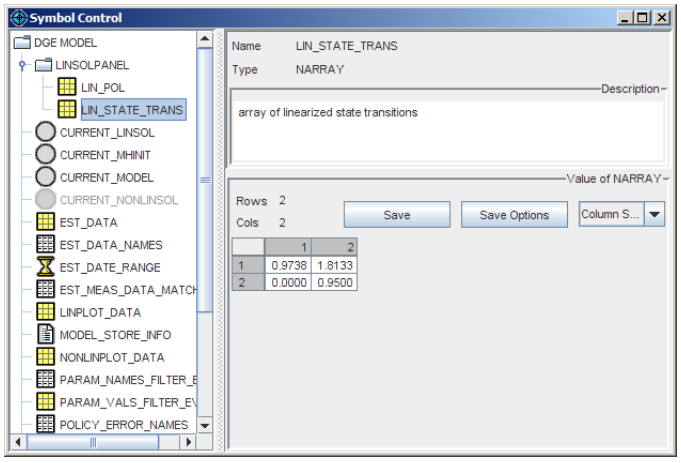

Fig. 11. Symbol control showing parts of linear solution

5.4 Scripting

The drawback of using a GUI application is that it only provides features that have been prepared by the developers. There is a lack of flexibility as compared to pure scripts. It is clear that researchers will always require the possibility to

```
$ModelSpec
C1^((1 - tau1) * theta1 - 1) * (1 - L1)^((1 - tau1) * (1 -
    theta1)) - beta * Z1;
K1_f = Inv1 + (1 - delta1) * K1;
A1_f = Ea1 + A1 * rho1;
...

$Parameters
alpha1=0.40;
beta=0.99;
...

$ShockDist
Ea1: NORMAL, MEAN=0.0, SIGMA=0.035;
Mk1: NORMAL, MEAN=0.0, SIGMA=8.66E-4;
...

$SteadyStatesAnalytic
L1 := (alpha1 - 1) * (1 + beta * (delta1 - 1)) * theta1 * (
    alpha1 * theta1 - 1 + beta * (1 + (alpha1 - 1) * delta1 -
     alpha1 * theta1))^(-1);
A1 := 0;
...

$SteadyStateStartVals
A1=0;
K1=23;
L1=0.3121;

$Priors
A1: NORMAL, MEAN=0.0, SIGMA=0.1;
alpha1: UNIFORM, LBOUND=0.0, UBOUND=1.0;
...

$InitDraws
alpha1: UNIFORM, LBOUND=0.390459312, UBOUND=0.390459312;
beta: UNIFORM, LBOUND=0.9875197840000001, UBOUND
    =0.9875197840000001;
...

$StatesGridBounds
A1=-0.3,0.3;
K1=20,30;
```

Listing 6. Structure of an example model definition file

use the system in new and unanticipated ways. Currently, JBendge addresses this issue by providing a Java interface to the users. This might prevent users trained in other languages from using it. However, it is much easier to do the scripting with JBendge than to write Fortran code, which is also a typical tool in this problem domain.

Furthermore, scripting with JBendge only requires to write rather basic Java code that is easy to learn. Listing (7) shows the code to load a model definition file, to find the steady state and the linear and the nonlinear solutions of that model. The Euler errors are calculated and displayed as well. Using Java can even be very convenient if one uses a modern integrated development environment, like for example Eclipse. Such a tool highlights syntax errors, does code indentation and syntax coloring, the compilation is done automatically after every change and typing is supported by automatic completions, to name just a few benefits not typically available in statistical packages. Figure (12) shows the Java editor from the Eclipse IDE. The mistyped call `settStatesGridBounds` is underlined and the box shows all available methods of the class `NonLinearSolution`. This should demonstrate that scripting with Java can be convenient and relatively easy. Furthermore, all required tools are open source, which allows to step through single lines of code with the debugger and to analyze numerical traps in every detail. It is also possible to use JBendge as a library callable from other languages via the available Java interfaces. Hence, JBendge is a versatile modeling environment that can be used with and without its GUI interface.

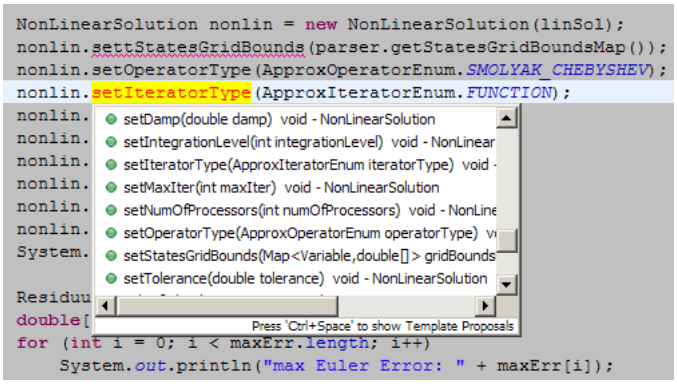

Fig. 12. Eclipse IDE code editor

6 Conclusion

We have presented a new and comprehensive software package that is especially useful for solving and estimating DSGE models. Its estimation procedures are defined in terms of linear or nonlinear state space models and are therefore useful for a much larger class of models. Many of the implemented

```
ModelParser parser = new ModelParser();
parser.parseFromFile(new File("modelparse.txt"));
DGEModel model = parser.getAssembledModel();

SteadyState ss = new SteadyState(model);
ss.solve(parser.getStartingValsSSMap());
System.out.println(ss.getSteadyStateMap());

LinearSolution linSol = new LinearSolution(ss);
linSol.solve();

NonLinearSolution nonlin = new NonLinearSolution(linSol);
nonlin.setStatesGridBounds(parser.getStatesGridBoundsMap());
nonlin.setOperatorType(ApproxOperatorEnum.SMOLYAK_CHEBYSHEV);
nonlin.setIteratorType(ApproxIteratorEnum.FUNCTION);
nonlin.setApproxLevel(2);
nonlin.setIntegrationLevel(2);
nonlin.setMaxIter(2000);
nonlin.setTolerance(1e-4);
nonlin.initialize();
nonlin.solve();
System.out.println(nonlin.report());

Residuum res = nonlin.estimateResiduum(1000);
double[] maxErr = res.getMaxEulerError();
for (int i = 0; i < maxErr.length; i++)
    System.out.println("max Euler Error: " + maxErr[i]);
```

Listing 7. Java example

methods are unique and so far not found in any other tool. The interactive MH sampler offers much better usability than existing samplers which have to be restarted when parameters need to be changed.

JBendge takes a new path in econometric software development as it is not built with a specialized statistical package but uses the mainstream programming language Java and is strictly object-oriented. The advantages of that approach are that modern elaborate development tools may be used and that a wealth of additional language features and libraries is available. Furthermore, the software can run on literally any operating system without changes. Another important aspect is that concurrent and grid computing is supported by libraries that are mature and well supported. This comes at the price of slightly worse scripting capabilities and more verbose numerical code.

We argue that software development in economics can exploit significant gains from applying current software engineering principles. These help to build and maintain larger systems with better quality. This holds especially for

complex systems with many interacting components and algorithms of which many have a quite natural abstraction as objects. It is not always easy to follow these principles in specialized languages because they often lack certain important features or tool support. For example, among the most vital quality engineering tools are automated unit tests which check all features of a project after changes have been made (Beck, 1999). But this needs to be supported by tools that make the process of creating, maintaining and executing these tests sufficiently convenient. Otherwise they would always be skipped and loose their value.

We also want to stress the community aspect of the open source software paradigm. As JBendge is hosted at sourceforge, a big provider for such projects that offers many essential features to users and developers, it is ideally suited for online collaboration. This spans from bug reports and informed critique over detailed feature requests to active development. Due to the modular approach of JBendge, it is possible to clearly separate development tasks. A remarkably successful project that bears many similarities to JBendge is Bioclipse (Spjuth et al., 2007). It is based on a mature Java framework for inheriting basic functionality and is organized as an open source project. By using a similar organizational setup we hope to achieve a significant level of collaboration also with JBendge.

References

Arulampalam, S., Maskell, S., Gordon, N., Clapp, T., 2002. Tutorial on particle filters for on-line non-linear/non-gaussian bayesian tracking. IEEE Transactions on Signal Processing 50 (2), 174–188.

Beck, K., 1999. Extreme Programming Explained: Embrace Change, 1st Edition. Addison-Wesley.

Brooks, S. P., Gelman, A., 1998. General methods for monitoring convergence of iterative simulations. Journal of Computational & Graphical Statistics 7 (4), 434–455.

Bungartz, H.-J., Griebel, M., December 2004. Projection methods for solving aggregate growth models. Acta Numerica 13, 1–123.

Chib, S., Greenberg, E., 1995. Understanding the metropolis-hastings algorithm. The American Statistician 49 (4), 327–335.

Deutsch, L. P., 1989. Design reuse and frameworks in the Smalltalk-80 system. In: Biggerstaff, T. J., Perlis, A. J. (Eds.), Software Reusability, Volume II: Applications and Experience. Addison-Wesley, Reading, MA, pp. 57–71.

Gamma, E., Helm, R., Johnson, R., Vlissides, J., 1995. Design Patterns: Elements of Reusable Object-Oriented Software. Addison-Wesley, Reading, MA.

Judd, K. L., December 1992. Projection methods for solving aggregate growth models. Journal of Economic Theory 58 (2), 410–452.

Juillard, M., 1996. Dynare: A program for the resolution and simulation of dynamic models with forward variables through the use of a relaxation algorithm. Discussion paper, CEPREMAP.

Kalman, R. E., 1960. A new approach to linear filtering and prediction problems. Transactions of the ASME–Journal of Basic Engineering 82 (Series D), 35–45.

Klein, P., 2000. Using the generalized schur form to solve a multivariate linear rational expectations model. Journal of Economic Dynamics and Control 24, 1405–1423.

Kotecha, J., Djuric, P., 2003. Gaussian sum particle filter. IEEE Transactions on Signal Processing 51, 2602–2612.

Krätzig, M., 2006. A software framework for data analysis. Computational Statistics and Data Analysis 52 (2), 618–34.

Miranda, M. J., Fackler, P. L., 2002. Applied Computational Economics and Finance. MIT Press, Cambridge, MA.

Press, W. H., Teukolsky, S. A., Vetterling, W. T., Flannery, B. P., 2007. Numerical Recipes: The Art of Scientific Computing, 3rd Edition. Cambridge University Press, Cambridge.

Spjuth, O., Helmus, T., Willighagen, E., Kuhn, S., Eklund, M., Wagener, J., Rust, P. M., Steinbeck, C., Wikberg, J., 2007. Bioclipse: An open source workbench for chemo- and bioinformatics. BMC Bioinformatics 8 (1).

Storn, R., Price, K., 1997. Differential evolution - a simple and efficient heuristic for global optimization over continuous spaces. Journal of Global Optimisation, 341–359.

ter Braak, 2006. A markov chain monte carlo version of the genetic algorithm differential evolution: easy bayesian computing for real parameter spaces. Statistics and Computing 16 (3), 239–249.

Winschel, V., Krätzig, M., 2008. Solving, estimating and selecting nonlinear dynamic models without the curse of dimensionality. SFB 649 Discussion Paper 2008-018.
URL http://sfb649.wiwi.hu-berlin.de/papers/pdf/SFB649DP2008-018.pdf

SFB 649 Discussion Paper Series 2008

For a complete list of Discussion Papers published by the SFB 649, please visit http://sfb649.wiwi.hu-berlin.de.

001 "Testing Monotonicity of Pricing Kernels" by Yuri Golubev, Wolfgang Härdle and Roman Timonfeev, January 2008.
002 "Adaptive pointwise estimation in time-inhomogeneous time-series models" by Pavel Cizek, Wolfgang Härdle and Vladimir Spokoiny, January 2008.
003 "The Bayesian Additive Classification Tree Applied to Credit Risk Modelling" by Junni L. Zhang and Wolfgang Härdle, January 2008.
004 "Independent Component Analysis Via Copula Techniques" by Ray-Bing Chen, Meihui Guo, Wolfgang Härdle and Shih-Feng Huang, January 2008.
005 "The Default Risk of Firms Examined with Smooth Support Vector Machines" by Wolfgang Härdle, Yuh-Jye Lee, Dorothea Schäfer and Yi-Ren Yeh, January 2008.
006 "Value-at-Risk and Expected Shortfall when there is long range dependence" by Wolfgang Härdle and Julius Mungo, Januray 2008.
007 "A Consistent Nonparametric Test for Causality in Quantile" by Kiho Jeong and Wolfgang Härdle, January 2008.
008 "Do Legal Standards Affect Ethical Concerns of Consumers?" by Dirk Engelmann and Dorothea Kübler, January 2008.
009 "Recursive Portfolio Selection with Decision Trees" by Anton Andriyashin, Wolfgang Härdle and Roman Timofeev, January 2008.
010 "Do Public Banks have a Competitive Advantage?" by Astrid Matthey, January 2008.
011 "Don't aim too high: the potential costs of high aspirations" by Astrid Matthey and Nadja Dwenger, January 2008.
012 "Visualizing exploratory factor analysis models" by Sigbert Klinke and Cornelia Wagner, January 2008.
013 "House Prices and Replacement Cost: A Micro-Level Analysis" by Rainer Schulz and Axel Werwatz, January 2008.
014 "Support Vector Regression Based GARCH Model with Application to Forecasting Volatility of Financial Returns" by Shiyi Chen, Kiho Jeong and Wolfgang Härdle, January 2008.
015 "Structural Constant Conditional Correlation" by Enzo Weber, January 2008.
016 "Estimating Investment Equations in Imperfect Capital Markets" by Silke Hüttel, Oliver Mußhoff, Martin Odening and Nataliya Zinych, January 2008.
017 "Adaptive Forecasting of the EURIBOR Swap Term Structure" by Oliver Blaskowitz and Helmut Herwatz, January 2008.
018 "Solving, Estimating and Selecting Nonlinear Dynamic Models without the Curse of Dimensionality" by Viktor Winschel and Markus Krätzig, February 2008.
019 "The Accuracy of Long-term Real Estate Valuations" by Rainer Schulz, Markus Staiber, Martin Wersing and Axel Werwatz, February 2008.
020 "The Impact of International Outsourcing on Labour Market Dynamics in Germany" by Ronald Bachmann and Sebastian Braun, February 2008.
021 "Preferences for Collective versus Individualised Wage Setting" by Tito Boeri and Michael C. Burda, February 2008.

SFB 649, Spandauer Straße 1, D-10178 Berlin
http://sfb649.wiwi.hu-berlin.de

This research was supported by the Deutsche Forschungsgemeinschaft through the SFB 649 "Economic Risk".

022 "Lumpy Labor Adjustment as a Propagation Mechanism of Business Cycles" by Fang Yao, February 2008.
023 "Family Management, Family Ownership and Downsizing: Evidence from S&P 500 Firms" by Jörn Hendrich Block, February 2008.
024 "Skill Specific Unemployment with Imperfect Substitution of Skills" by Runli Xie, March 2008.
025 "Price Adjustment to News with Uncertain Precision" by Nikolaus Hautsch, Dieter Hess and Christoph Müller, March 2008.
026 "Information and Beliefs in a Repeated Normal-form Game" by Dietmar Fehr, Dorothea Kübler and David Danz, March 2008.
027 "The Stochastic Fluctuation of the Quantile Regression Curve" by Wolfgang Härdle and Song Song, March 2008.
028 "Are stewardship and valuation usefulness compatible or alternative objectives of financial accounting?" by Joachim Gassen, March 2008.
029 "Genetic Codes of Mergers, Post Merger Technology Evolution and Why Mergers Fail" by Alexander Cuntz, April 2008.
030 "Using R, LaTeX and Wiki for an Arabic e-learning platform" by Taleb Ahmad, Wolfgang Härdle, Sigbert Klinke and Shafeeqah Al Awadhi, April 2008.
031 "Beyond the business cycle – factors driving aggregate mortality rates" by Katja Hanewald, April 2008.
032 "Against All Odds? National Sentiment and Wagering on European Football" by Sebastian Braun and Michael Kvasnicka, April 2008.
033 "Are CEOs in Family Firms Paid Like Bureaucrats? Evidence from Bayesian and Frequentist Analyses" by Jörn Hendrich Block, April 2008.
034 "JBendge: An Object-Oriented System for Solving, Estimating and Selecting Nonlinear Dynamic Models" by Viktor Winschel and Markus Krätzig, April 2008.

SFB 649, Spandauer Straße 1, D-10178 Berlin
http://sfb649.wiwi.hu-berlin.de

This research was supported by the Deutsche Forschungsgemeinschaft through the SFB 649 "Economic Risk".