Solving SDGE Models: New Algorithm for Sylvester Equation

Ondřej Kameník[1]

Abstract

This paper presents a new numerical algorithm for solving Sylvester equation involved in higher order perturbation method developed for solution of stochastic dynamic general equilibrium models. The new algorithm is better than methods used so far (esp. very popular doubling algorithm) in terms of computational time, memory consumption, and numerical stability.

Further, the paper applies the algorithm in a simulation of a large macroeconomy model providing a simple welfare analysis of a few monetary rules. The welfare analysis compares household's lifetime expected utility.

I wish to thank to Michel Juillard for his invaluable help during this project, for providing the source code of Dynare C++ prototype, and Douglas Laxton for providing a Dynare code of Czech–EU calibration of GEM. Many thanks also to my colleagues here in Czech National Bank.

A usual disclaimer applies.

[1] The Czech National Bank, Monetary and Statistical Department, E-mail: ondrej.kamenik@cnb.cz.

Nontechnical Summary

A perturbation approach to solve stochastic dynamic general equilibrium (SDGE) models seems very promising in comparison with other methods, since it allows handling of relatively large state space. According to [Jin and Judd, 2002] and [Juillard, 2003], the solution of k-order approximation consists of finding the non-stochastic steady state and then solving for k-order approximation around the steady state. The k-order approximation is found iteratively from the approximations of lower orders. The iterative process is started by standard solution of linear approximation ($k = 1$). The iterative step consists of two ordinary linear systems, and one linear Sylvester equation. While the two equations present no numerical problem, the Sylvester equation is a real computational challenge. Although size of input data of the three equations is more or less the same, the Sylvester equation entails solution of huge matrix whose size grows exponentialy with k and polynomially with number of predetermined variables.

The main result of this paper is developing the recursive algorithm for solving the Sylvester equation involved in the k-order approximations. The algorithm was implemented in C++ and tested. An efficiency of the recursive algorithm is shown in comparison with two methods used so far. These are a method due to [Bartels and Stewart, 1972] implemented by Michel Juillard, and doubling algorithm implemented also by the author of this paper. Since the application of the Bartels–Stewart approach in the context of SDGE models doesn't take into account a special structure of the Sylvester equation, it has quadratic complexity both in time and memory. This makes the method unusable for large models or high order approximations. When comparing with the doubling algorithm, the author's implementation of the doubling algorithm needs almost three times as much computational time and twice as much memory as the new recursive algorithm. The superiority of the new recursive approach is shown on the solution of second order approximation to Global Economy Model (GEM) calibrated for Czech Republic and Euro area [Laxton and Pesenti, 2003]. In this particulary difficult model, the doubling algorithm yields relative error of equation residual at order of 10^{-1}, and the new resursive method at order of 10^{-14}.

1. Introduction

We open the paper with a sketch of the paper [Juillard, 2003]. This should provide the reader with an intuition how the Sylvester equation is employed in k-order approximation of a non-linear SDGE model.

We are interested in discrete time models of the form[2]:

$$E_t\left(f(y^*_{t-1}, y_t, y^{**}_{t+1}, u_t)\right) = 0, \tag{1}$$

where y is a vector of n state variables, out of which y^* denotes m predetermined variables, and y^{**} denotes a $n-m$ vector of forward looking variables. (We count static variables as predetermined.) Further, u_t is a vector of stochastic shocks. For the sake of perturbation method, we write u_t as a result of σ scaling of η_t shocks, i.e.

$$u_t = \sigma\eta_t.$$

A dynamic equilibrium solution of this model is, in fact, a decision function of the following form:

$$y_t = g(y_{t-1}, u_t, \sigma),$$

which contains the decision rules for predetermined and forward looking variables, i.e.

$$\begin{aligned} y^*_t &= g^*(y^*_{t-1}, u_t, \sigma) \\ y^{**}_t &= g^{**}(y^*_{t-1}, u_t, \sigma) \end{aligned}$$

The original model can be then expressed as having the only reference to unknown future through serially uncorrelated shocks u_{t+1}, this is:

$$\begin{aligned} 0 &= E_t\left(F(y^*_{t-1}, u_t, \sigma, u_{t+1})\right) \\ &= E_t\left(f\left(y^*_{t-1}, g(y^*_{t-1}, u_t, \sigma), g^{**}\big(g^*(y^*_{t-1}, u_t, \sigma), u_{t+1}, \sigma\big), u_t\right)\right). \end{aligned} \tag{2}$$

Now let us suppose that we have already found the model's steady state $\bar{y}$, and the solution to the linear approximation. The latter includes Jacobians of g^*, and g^{**} with respect to y^*, and y^{**} resp. A goal of k-order perturbation method at this point is to obtain k order derivatives of g with respect to all y, u, σ, including all possible cross derivatives. In what follows, we provide an intuitive explanation of how the second (and higher) derivatives of g with respect to y are retrieved and how the Sylvester equation emerges.

For simplicity, we will do it for the second order ($k = 2$). However, exactly same ideas are applicable for steps of higher orders. First, it is not difficult (though technically demanding) to show that

$$[F_{y^*}]^i_{\alpha_1\alpha_2} = 0, \tag{3}$$

where i goes through all equations, and $[F_{y^*}]^i_{\alpha_1\alpha_2}$ is second derivative of i-th equation with respect to $y^*_{\alpha_1}$, and $y^*_{\alpha_2}$ [3] at the steady state $\bar{y}$.

[2] We use the same notation as in [Juillard, 2003].

[3] For bracketed expressions such as $[F_{y^*}]^i_{\alpha_1\alpha_2}$ the superscripts are used for equations, subscripts for variables.

Now using equation (2) we will rewrite the equation (3) in terms of derivatives of f, known first order derivatives of g, and unknown second order derivatives of g (these are three dimensional matrices for derivatives of g^* and g^{**}, i.e. $g^*_{y^{*2}}$, and $g^{**}_{y^{*2}}$). Since we only examine how the Sylvester equation comes out, we are interested only in terms involving unknowns $g^*_{y^{*2}}$, and $g^{**}_{y^{*2}}$. Such the terms are obtained as sum of derivatives with respect to arguments of f multiplied by second derivatives with respect to y^*. These are:

1) The derivative of f with respect to the first argument y^*_{t-1} is denoted as $f_{y^*_-}$. However, the second derivative of the argument $y^*_{t-1} = y^*_{t-1}$ is 0, so there is no contribution from the first argument.
2) The derivative of f with respect to the second argument y_t is denoted as f_y, which is block matrix $(f_{y^*}\ f_{y^{**}})$. The second derivative of $y_t = g(y^*_{t-1}, u_t, \sigma)$ with respect to y^*_{t-1} is a block three dimensional matrix $(g^*_{y^{*2}}\ g^{**}_{y^{*2}})^T$. So the overall contribution of the second argument is $[f_{y^*}]_\beta \cdot [g^*_{y^{*2}}]^\beta + [f_{y^{**}}]_\beta \cdot [g^{**}_{y^{*2}}]^\beta$[4]
3) The derivative of f with respect to the third argument y^{**}_{t+1} is $f_{y^{**}_+}$. The second derivative of $y^{**}_{t+1} = g^{**}(g^*(y^*_{t-1}, u_t, \sigma), u_{t+1}, \sigma)$ with respect to y^*_{t-1} is more complicated:
   a) First, it includes a first derivative of g^{**} with respect to the first argument multiplied by a second derivative of g^* with respect to y^*_{t-1}. This is $[g^{**}_{y^*}]_\beta \cdot [g^*_{y^{*2}}]^\beta$.
   b) Second, it includes a second derivative of g^{**} with respect to the first argument multiplied by corresponding combinations of first derivatives of g^*. In more detail, a second derivative $[g^{**}_{y^{*2}}]^\beta_{\alpha_1\alpha_2}$ must be multiplied by a sum of all combinations of products $[g^*_{y^*}]^{\alpha_1}_{\gamma_1}[g^*_{y^*}]^{\alpha_2}_{\gamma_2}$. In tensor notation, this is $[g^{**}_{y^{*2}}]_{\alpha_1\alpha_2} \cdot [g^*_{y^*}]^{\alpha_1} \cdot [g^*_{y^*}]^{\alpha_2}$.[5]
4) Since the fourth argument of f is u_t, there is no contribution to the terms in our interest.

Now note that in the equation (3) there are no other terms containing second derivatives (not only with respect to y^*_{t-1}) than those listed above, so the (3) can be rewritten as:

$$[f_{y^*}]_\beta \cdot [g^*_{y^{*2}}]^\beta + [f_{y^{**}}]_\beta \cdot [g^{**}_{y^{*2}}]^\beta + [f_{y^{**}_+} g^{**}_{y^*}]_\beta \cdot [g^*_{y^{*2}}]^\beta + [f_{y^{**}_+}]_\beta \cdot \left[[g^{**}_{y^{*2}}]_{\alpha_1\alpha_2} \cdot [g^*_{y^*}]^{\alpha_1} \cdot [g^*_{y^*}]^{\alpha_2}\right]^\beta = [D], \tag{4}$$

where D contains all other terms. The relation (4) is an equation of three dimensional matrices. With a careful handling of the last term, the equation can be put into two

[4] Here we use tensor product notation. For two dimensional A, and three dimensional B, we define $\left[[A]_\beta \cdot [B]^\beta\right]^\gamma_{\alpha_1\alpha_2} = \sum_\beta [A]^\gamma_\beta [B]^\beta_{\alpha_1\alpha_2}$

[5] Precisely, for three dimensional A, and two dimensional B, C we define

$$[[A]_{\alpha_1\alpha_2} \cdot [B]^{\alpha_1} \cdot [C]^{\alpha_2}]^\beta_{\gamma_1\gamma_2} = \sum_{\alpha_1,\alpha_2} [A]^\beta_{\alpha_1\alpha_2} [B]^{\alpha_1}_{\gamma_1} [C]^{\alpha_2}_{\gamma_2}$$

dimensional form, which is:[6]

$$f_{y^*} g^*_{y^{*2}} + f_{y^{**}} g^{**}_{y^{*2}} + f_{y^{**}_+} g^{**}_{y^*} g^*_{y^{*2}} + f_{y^{**}_+} g^{**}_{y^{*2}} \left(g^*_{y^*} \otimes g^*_{y^*} \right) = D$$

This can be rewritten as

$$\begin{pmatrix} f_{y^*} + f_{y^{**}_+} g^{**}_{y^*} & f_{y^{**}} \end{pmatrix} \begin{pmatrix} g^*_{y^{*2}} \\ g^{**}_{y^{*2}} \end{pmatrix} + \begin{pmatrix} 0 & f_{y^{**}_+} \end{pmatrix} \begin{pmatrix} g^*_{y^{*2}} \\ g^{**}_{y^{*2}} \end{pmatrix} \left(g^*_{y^*} \otimes g^*_{y^*} \right) = D$$

The same ideas made for the second order can be repeated for k-th order ending with an equation of the form:

$$AX + BX \left(\otimes^k C \right) = D_k, \tag{5}$$

where

$$\begin{aligned} A &= \begin{pmatrix} f_{y^*} + f_{y^{**}_+} g^{**}_{y^*} & f_{y^{**}} \end{pmatrix} \\ B &= \begin{pmatrix} 0 & f_{y^{**}_+} \end{pmatrix} \\ C &= g^*_{y^*} \\ X &= \begin{pmatrix} g^*_{y^{*k}} \\ g^{**}_{y^{*k}} \end{pmatrix} \text{ matrix of unknowns} \\ D_k &= \text{right hand side dependent on } k \end{aligned}$$

The remainder of the paper is organized as follows: The second chapter introduces the new recursive algorithm solving the equation (5), and analyzes memory consumption and computational complexity. The third chapter compares the recursive algorithm with Bartels–Stewart approach, and doubling algorithm. It shows that the recursive algorithm is better than the two alternatives in terms of time, memory, and numerical stability. The fourth chapter applies the recursive algorithm in a simulation of a large macro model. The aim of the simulations is a simple welfare analysis of two monetary rules. The first appendix describes in more detail an algorithm used for block diagonalization of a matrix. The second technical appendix provides a formal description of the recursive algorithm.

[6] We do not use brackets for ordinary two dimensional matrices.

2. The Recursive Algorithm

2.1. Complexity of the Problem

Before we start with a description of the algorithm, it is useful to asses the computational complexity of solving (5). Recall, that n is a number of equations in the model (1), and m is a number of predetermined variables. Size of matrix A in (5) is (n, n), B is also (n, n), C is (m, m), and finally X and D_k have size of (n, m^k).

To see how large the equation (5) can be, let us take the Czech EU calibration of GEM as an example, see [Laxton and Pesenti, 2003]. It has $n = 244$ equations, and 49 forward looking variables. After eliminating static variables from (5), we get $m = 88$ predetermined dynamic variables. The following table shows an amount of memory needed for storage of matrix D_k for a few k's.

Order k	Memory for D_k
2	14.4 MB
3	1.24 GB
4	109.0 GB

The equation (5) can be put in $Ax = b$ form by vectorizing:[7]

$$\left(I \otimes A + \left(\otimes^k C^T\right) \otimes B\right) \mathrm{vec}(X) = \mathrm{vec}(D_k)$$

Clearly, the problem cannot be attacked by Gausian elimination. As an illustration, consider estimated solution times for EarthSimulator supercomputer quoted in the following table:

Order k	Time for Gausian elimination
2	128 seconds
3	2 years 9 months

2.2. Preconditioning

The solution to (5) is obtained in three steps. First, a suitable linear transformation preconditioning the equation is found, then the transformed equation is solved, and third, the inverse transformation is applied to obtain the solution to the original system.

The preconditioning step consists of multiplying by A^{-1} from the left yielding:

$$X + A^{-1}BX\left(\otimes^k C\right) = A^{-1}D$$

[7] $\mathrm{vec}(X)$ is a vector of stacked columns of X

and finding real Schur decompositions $K = U(A^{-1}B)U^T$, and $F = VCV^T$ obtaining:

$$Y + KY\left(\otimes^k F\right) = \bar{D} \tag{6}$$

$$Y = UX\left(\otimes^k V^T\right) \tag{7}$$

$$\bar{D} = UA^{-1}D\left(\otimes^k V^T\right)$$

When (6) is solved, we apply an inverse of the linear transformation (7) to recover solution X, i.e:

$$X = U^T Y\left(\otimes^k V\right) \tag{8}$$

Two issues are worth noting here. The first is the existence and stability of inverse A^{-1}. The existence is implied by a fact that for the first order $g_u = -A^{-1}f_u$. The numerical stability is more complex. Basically, if the matrix A is poorly conditioned[8], the preconditioning step can introduce severe numerical errors. If this is the case, the model is most likely ill-stated, since inverse of A is fundamental for solution of the linear approximation. However, the condition number gives only an upper bound of numerical errors. In practice, therefore, our implementation reports a residual relative size of (5). The numerical error of $A^{-1}B$, and $A^{-1}D$ should be checked, only if the residual relative size is too big.

The second issue regards to the Schur decomposition of C. As it will become clear from the following section, the recursive algorithm will be boosted by large number of zero elements in the quasi-triangular matrix F. Additional zero elements can be introduced into F by sacrificing orthogonality of V, i.e. $C = VFV^{-1}$. However, the departure from the orthogonality can introduce large numerical errors, as a condition number of V is no more equal to 1 as is the case for orthogonal matrices. We implemented an algorithm using results of [Bavely and Stewart, 1979] and [Dongarra et al., 1992]. The algorithm is briefly sketched in *Appendix A*.

2.3. The Recursive Solution

Here we provide the core algorithm of the paper. It solves the equation (6), which can be vectorized as:

$$\left(I + \left(\otimes^k F^T \otimes K\right)\right)\mathrm{vec}(Y) = \mathrm{vec}(\bar{D}). \tag{9}$$

Although the algorithm works with the vectorized version, the matrix in (9) is not stored in memory. A basic idea of the recursive algorithm is to obtain the solution at k level with the solutions of the same problem (or similar) at $k-1$ level. In order to simplify the notation, let $F_{[k]}$ denote $\otimes^k F^T \otimes K$. For $k = 0$, $F_{[k]} = K$.

Recall, that both F and K are quasi-triangular matrices. In order to describe the recursive approach, suppose that the first eigenvalue of F is real, let us denote this as

[8] Condition number of a matrix is defined as a relative distance from the closest singular matrix

$r = F_{11}$. If we pick y as the first part of Y (it corresponds to the size of $F_{[k-1]}$ matrix), and d as the corresponding first part of $\bar{D}$, then y is a solution of

$$(I + r \cdot F_{[k-1]})y = d. \tag{10}$$

Note that this is pretty similar to the equation (9). If the first eigenvalue is complex, we pick the first two parts of Y, and the first two parts of $\bar{D}$. The first two parts of Y can be then obtained as a solution of

$$\left(I + \begin{pmatrix} \alpha & \beta_1 \\ -\beta_2 & \alpha \end{pmatrix} \otimes F_{[k-1]}\right) \begin{pmatrix} y_1 \\ y_2 \end{pmatrix} = \begin{pmatrix} d_1 \\ d_2 \end{pmatrix}, \tag{11}$$

where α, β_1, and β_2 constitute the first complex eigenvalue block.

When we obtain the solution of (10) or (11), we go through all non-zero elements of F^T below[9] the eigenvalue block and eliminate them by updating subsequent parts of matrix $\bar{D}$. More precisely, for the real eigenvalue we update

$$d_j \longleftarrow d_j - F_{1j} \cdot (F_{[k-1]})\, y \quad \text{for all } j = 2, \ldots, m$$

And for the complex eigenvalue we update

$$d_j \longleftarrow d_j - F_{1j} \cdot (F_{[k-1]})\, y_1 - F_{2j} \cdot (F_{[k-1]})\, y_2 \quad \text{for all } \; j = 3, \ldots, m$$

After the above elimination is performed, subsequent parts (or pair of parts) of Y can be found as a solution of equation of type (10) (or (11) resp.). In this way, the solution Y is obtained.

What remains to discuss is how equations for real (10) and complex (11) cases are solved. As noted before, (10) is a slight generalization of (6) ($r = 1$). It is clear, that solving equation (10) at $k-1$ level in the same manner as we solved the original equation (6) at k level, will produce problems of both types ((10) and (11)) at $k-2$, $k-3$, ... levels. For $k = 0$, the solution of (10) is easy since $I + rK$ is quasi triangular.

Now it suffices to show how (11) is solved. As *Lemma 1* in *Appendix B* claims, the solution of (11) is obtained as a solution of two equations of the form:[10]

$$\left(I + 2\alpha \cdot F_{[k-1]} + (\alpha^2 + \beta^2) \cdot F^2_{[k-1]}\right) y = d \tag{12}$$

Now note that the eigenvalues of F^2 are squares of eigenvalues of F, and since F is the Schur factor, the order of eigenvalues in F and F^2 is the same. Therefore, this equation can be solved in the very simmilar manner as we solved the equation (6). We simply go through all the diagonal blocks of F (coinciding with blocks of F^2), solving an appropriate equation followed by an elimination.

What are the appropriate equations which need to be solved during this process? For a real eigenvalue r of F (corresponding to r^2 of F^2), the equation takes the form:

$$\left(I + 2r\alpha \cdot F_{[k-2]} + r^2(\alpha^2 + \beta^2) \cdot F^2_{[k-2]}\right) y = d$$

[9] F^T is quasi lower triangular

[10] here $F^2_{[k]} = \otimes^k \left(F^T\right)^2 \otimes K^2$

In fact, this is exactly the same as (12), so we solve it by recursion. At the bottom of recursion ($k = 0$), we get an equation of the form $(I + \alpha K + (\alpha^2 + \beta^2)K^2)y = d$ which is easy to solve since $I + \alpha K + (\alpha^2 + \beta^2)K^2$ is quasi triangular.

For a complex eigenvalue of F corresponding to 2×2 block of F^T

$$\begin{pmatrix} \gamma & \delta_1 \\ -\delta_2 & \gamma \end{pmatrix}$$

we come to an equation of the form:

$$\left(I + 2\alpha \begin{pmatrix} \gamma & \delta_1 \\ -\delta_2 & \gamma \end{pmatrix} \otimes F_{[k-2]} + (\alpha^2 + \beta^2) \begin{pmatrix} \gamma & \delta_1 \\ -\delta_2 & \gamma \end{pmatrix}^2 \otimes F_{[k-2]}^2 \right) \begin{pmatrix} y_1 \\ y_2 \end{pmatrix} = \begin{pmatrix} d_1 \\ d_2 \end{pmatrix}$$

The *Lemma 2* in *Appendix B* claims that this type of equation can be transformed to solution of two systems of two serial equations of the type (12). In this way, the recursion is closed.

More formal account is given in *Appendix B*.

2.4. Memory Consumption

The algorithm given above can be implemented so that its input is overwritten by its output. Therefore, besides storage for input/output vector, the algorithm needs memory only for temporary vectors allocated on the recursion's stack. These are modified right hand sides in *Lemma 1* and *Lemma 2* in *Appendix B*. The peak of memory consumption will be reached at the bottom of a recursion having all complex eigenvalues in higher recursion levels. The maximum is:

$$\sum_{i=0}^{k} nm^i = n\frac{m^{k+1} - 1}{m - 1}.$$

2.5. Numerical Complexity

Unlike the memory consumption, the numerical complexity is not that simple. If calculating number of flops, a solution of appropriate recursive formulas yields a fairly complicated result. Without making any further assumptions, it is not possible to deduce dominating terms. The following table shows the flops complexity for extreme assumptions of fully and moderatelly filled matrix F. The latter means that there are pm non-zero elements in matrix F, where $p \ll m$. Note that n_1 is a number of forward looking variables.[11]

[11] In an implementation, we do not solve for the derivatives of the decision rules with respect to static variables, because they are zero. Therefore they are excluded and thus $n_1 \leq n - m$.

Assumption	Flops complexity
no real eig., full F	$\Theta(m^k nn_1) + \Theta(m^{2k-2}nn_1) + \Theta(m^{k+1}n)$
no real eig., moderate F	$\Theta(m^k nn_1)$
all real eig., full F	$\Theta(km^{k+1}n)$
all real eig., moderate F	$\Theta(m^k nn_1) + \Theta(kpm^k n)$

The computational time can be divided into three parts. The first portion is spent in solution of bottom quasi triangular systems, the second is time spent in eliminations, and the third is time needed for right hand side modifications in *Lemma 1* and *Lemma 2* of *Appendix B*. For cases with all eigenvalues complex, the right hand side modification term dominates overall complexity. For full F, the term $\Theta(m^{2k-2}nn_1)$ dominates for $k > 2$, and for moderate F, the term $\Theta(m^k nn_1)$ is comparable with the term for quasi triangular systems. This assertion is justified by profiling the code which shows that aproximately half of computational time is spent by right hand side modifications. If there are no complex eigenvalues in F, the time is in most cases dominated by eliminations.

2.6. Complex Schur Modification

Having studied the computational complexity, a natural question arises. Why we do not precondition the equation (5) using a complex Schur decomposition obtaining strictly triangular (though complex) matrices K, and F in (6)? Then, all the calculations would be much more easier, since the system (6) is triangular. There would be no necessity of the right hand side modifications whose complexity dominates the overall complexity for a general case.

However, this "trick" will not improve overall computational time. The reason for this is that the true complexity lays in preconditioning (7), and recovering (8). The complexity of these steps is $\Theta(m^{2k}nn_1)$. The time of these steps can be four times longer than time of the same calculations handling only real numbers.

3. Comparison with Other Solution Methods

3.1. Bartels–Stewart Approach

The approach suggested by [Bartels and Stewart, 1972] is applicable for solving of a more general equation than (6), i.e.

$$X + SXT = D, \tag{13}$$

where S and T are square matrices and T (without loss of generality) is a block upper triangular with $m \times m$ square blocks. Note that T is not required to have a form of Kronecker product as in (6). Let T_{ij} denote a block of T, and let X and D be partitioned according to column partitioning of T. The main idea of this approach is to decompose the equation (13) to a number of smaller Sylvester equations as follows:

$$\begin{aligned} X_1 &= D_1 - SX_1T_{11} \\ X_j &= D_j - S\sum_{k=1}^{j-1} X_jT_{kj} - SX_jT_{jj} \quad \text{for } j = 2, \ldots, m \end{aligned}$$

Note that the both equations take the form of (13) as the sum in the second equation is known.[12] Also note that if T is a real Schur factor, then each T_{jj} is either 1×1, or 2×2.

Here we consider an implementation of this method as described in [Juillard, 2003]. Recall the equation (5) which solves for k-th derivatives of policy rules g with respect to state variables y. The derivatives is a multidimensional symmetric matrix, and when unfolded columnwise, some columns are repeated in both, X and D_k. When the repeated columns are deleted, the resulting matrices $\hat{X}$, and $\hat{D}_k$ have $(m+k-1)!/(k!(m-1)!)$ columns. These deletions must also be projected in the Kronecker power $\otimes^k C$ giving a matrix C_k not having a form of Kronecker product any more. So we come to the equation

$$A\hat{X} + B\hat{X}C_k = \hat{D}_k \tag{14}$$

This system is first preconditioned in a similar manner as for the recursive algorithm. We obtain quasi triangular $F_k = V_kC_kV_k^T$ and the equation takes the form:

$$\hat{Y} + K\hat{Y}F_k = \hat{E}_k \tag{15}$$

This is solved in the fashion described above.

First, let us compare the memory consumption of both algorithms. Recall that required memory for the recursive algorithm is approximately nm^k. In order to make the following expressions more intuitive, we will compare the memory requirements relative to the size of real algorithm input, which is $n(m+k-1)!/(k!(m-1)!)$.[13] In our framework, k is much smaller than m, so this can be approximated as $nm^k/k!$. The relative memory consumption

[12] Notice that our recursive algorithm fits to this pattern. T is block partitioned according to the top level of Kronecker power $\otimes^k F$, and the smaller Sylvester equations are solved recursively with help of *Lemma 1* and *Lemma 2* of *Appendix B*

[13] This is a size of $\hat{D}_k$ matrix, we neglect size of A, B, and C matrices.

of the recursive algorithm is then $k!$. The Bartels–Stewart algorithm allocates memory for full matrix V_k and quasi triangular matrix F_k . The required memory for these matrices is approximately $\frac{3}{2}(m^k/k!)^2$, and if divided by the size of input, we get $\frac{3}{2}m^k/(nk!)$.

For a fixed order k, the recursive algorithm is linear in memory consumption, while the Bartels–Stewart algorithm is polynomial of k-th order with respect to m. The implications of this conclusion are clear when the above results are applied on the GEM (recall $n = 244$, $m = 88$) as shown in the following table:

	absolute	relative
Recursive $k = 2$	14.58 MB	2.00
Bartels–Stewart $k = 2$	175.5 MB	24.07
Recursive $k = 3$	1.25 GB	5.86
Bartels–Stewart $k = 3$	154.24 GB	722.21

When estimating number of flops needed for Bartels–Stewart algorithm, we have to add flops from Schur decomposition of C_k and solution of (15). The Schur real decomposition has cubic complexity with respect to size of the matrix, so we get $O(m^{3k}/(k!)^3)$.[14] Supposing that F_k is not block diagonalized, we get number of flops of Bartels-Stewart algorithm of $\Theta(nn_1m^k/k!) + \Theta(nm^{2k}/(k!)^2)$. If not even considering the severe complexity of the large Schur decomposition, the Bartels-Stewart approach is still worse by m^2/n_1 than the recursive algorithm, whose worst case dominating flops term is $\Theta(nn_1m^{2k-2})$

There are a few variants of the Bartels–Stewart approach. One of them avoids the multiplication of (14) by A^{-1} in the preconditioning using generalized Schur decomposition of A, and B. Another uses Hessenberg decomposition of $A^{-1}B$ instead of Schur decomposition in the preconditioning. This method is known as Hessemberg–Schur algorithm, and can be found in [Golub and Loan, 1996] and [Anderson et al., 1996].

3.2. Doubling Algorithm

The most popular algorithm for solving equation (6) is a doubling algorithm (see [Anderson et al., 1996]). It exploits a property that all eigenvalues of both $A^{-1}B$ and C have modulus smaller than one. This implies that matrix $F_{[k]} = \otimes^k F^T \otimes K$ in equation (9) has all eigenvalues with modulus less than one and thus $(F_{[k]})^i$ converges to 0 as $i \to \infty$. Now we can use a classical result of linear algebra:

$$(I + F_{[k]})^{-1} = \sum_{i=0}^{\infty} (-1)^i (F_{[k]})^i$$

and the sum can be written as a product, i.e.

$$(I + F_{[k]})^{-1} = (I - F_{[k]}) \prod_{j=1}^{\infty} (I + (F_{[k]})^{2^j})$$

[14] Here we write O instead of Θ to express that the complexity can be better, for instance C_k might be already in the Schur form by some lucky accident.

Notice, that each multiplication in the product doubles a number of summands in the sum; hence the name "doubling" algorithm.

When these ideas are applied in solving (9), we get the following iterative process:

$$\begin{aligned} y_0 &= \mathrm{vec}(\bar{D}_k) \\ M_1 &= F_{[k]} \\ y_1 &= y_0 - M_1 y_0 \\ M_j &= M_{j-1} M_{j-1} \quad \text{for } j = 2, \dots \\ y_j &= y_{j-1} + M_j y_{j-1} \quad \text{for } j = 2, \dots \end{aligned}$$

This iterative process is stopped, when $M_j y_{j-1}$ is smaller than a user given tolerance, which means that subsequent y_j will not change more than the tolerance. In the doubling algorithm implementation we do not calculate M_j but we calculate iteratively $F_j = F^{2^j}$, and $K_j = K^{2^j}$, and then evaluate $M_j y_{j-1} = (\otimes^k F_j^T \otimes K_j) y_{j-1}$.

Let us look at the memory requirements. At each step, the algorithm updates a current solution by addition. This implies that two copies of the current solution are needed. This means that the memory consumption is $2nm^k$. However, if F has been successfully block diagonalized, a careful implementation can update appropriate part of the solution vector block by block reusing previously allocated memory. So, if $m_1 \leq m$ denotes a size of the largest diagonal block of F, then the memory consumption is $(1 + m_1/m)nm^k$.

Also it is not difficult to calculate number of flops of the doubling algorithm. Let m_f denote a number of non-zero elements in F. Number of flops per one iteration is of $\Theta(nn_1 m_f^k)$. If ν denotes a number of iterations, the overall complexity is $\Theta(\nu nn_1 m_f^k)$. Recall that if we take the most unfavourable case for the recursive algorithm, which is F with all eigenvalues complex and $m_f = m^2/2$, the dominating term of the flops complexity is $\Theta(nn_1 m_f^{k-1})$, which is νm_f times better than the doubling algorithm.

One more issue should be brought in. Consider a case for which the input/output vector $\mathrm{vec}(\bar{D}_k)$ of (9) is so long, that it cannot fit into physical memory. In these situations, a number of vector touches is often more critical for performance than flops. By a vector touch we mean a transport of a data chunk between processor and memory. Obviously, a vector touch can be very costly if all the data don't fit into memory.

It is easy to assert, that one iteration of the doubling algorithm has approximately the same number of vector touches as whole recursive algorithm since the data is traversed in a similar manner. Therefore, for really large problems which do not fit into memory, one can expect that the doubling algorithm will need ν times more disk loads and stores.

Much more serious (and often overlooked) issue is roundoff properties of the doubling algorithm. The algorithm employs square powers of matrices applied repeatedly. Recall that a roundoff error of such an operation satisfies $\|E\|_1 \leq n\mathbf{u}\|A\|_1^2 + O(\mathbf{u}^2)$, where n is a matrix dimension and $\mathbf{u}$ is the computer unit roundoff. If there is a significant rounding error in the first step of the algorithm, and high elements causing this error are far from the diagonal, then further steps can magnify this error before these elements are cut down. This problem can be discovered by evaluating residual matrix R of the equation (5). The

Czech-EU calibration of GEM [Laxton and Pesenti, 2003] suffers from this problem. The following table shows the computational time and relative residual sizes in a few norms of doubling and recursive algorithm for the second order simulation.

	recursive	doubling
Time (seconds)	9.5	29.3
$\|R\|_1/\|D_k\|_1$	5.635×10^{-15}	0.2394
$\|R\|_\infty/\|D_k\|_\infty$	1.045×10^{-13}	0.1458
$\|R\|_F/\|D_k\|_F$	1.366×10^{-14}	0.1956
$\|\operatorname{vec}(R)\|_1/\|\operatorname{vec}(D_k)\|_1$	2.408×10^{-14}	0.1088
$\|\operatorname{vec}(R)\|_\infty/\|\operatorname{vec}(D_k)\|_\infty$	2.419×10^{-14}	0.3911

The table shows that the doubling algorithm is unusable for this problem.

3.3. A Further Research

The Sylvester equation is a part of a recursive step towards k-order approximation of the decision rules. Its input data consist of the first derivatives of system equations (1), the first derivatives of policy rules g, and finally of a combination of the results of the previous steps $1 \dots k-1$. All inputs are subject to numerical errors, which in turn are propagated to subsequent orders of the approximation.

A challenging task of the further research would be a development of a perturbation theory for the Sylvester equation of the form (5). A preliminary examination of the problem indicates that obtaining normwise error bounds is not much more simple than obtaining componentwise bounds.

Having this error perturbation theory in hand, one would be able to estimate the numerical error bounds of the whole process.

4. Welfare Analysis Application

In this chapter we apply the second order perturbation method for a simple welfare analysis of two different monetary rules in GEM [Laxton and Pesenti, 2003]. The analysis simply compares lifetime utility expectations of households for the different parametrizations of monetary policy rule. Obviously, the linear approximation around the deterministic steady state is not sufficient for this task. This is because we need to simulate how uncertainty coming from exogeneous shocks is diffused through other variables, and how households respond. The monetary policy rule influences the way how the uncertainty is propagated through the economy.

For the simulation exercises we use a two country version of GEM, calibration for Euro area (Foreign country) and Czech Republic (Home country). We examine monetary rules and their welfare implications in the Home country.

4.1. Household Optimization in GEM

Here we provide a sketch of the household optimization in GEM. More details, the model description, and its parametrization is given in [Laxton and Pesenti, 2003].

The Home households[15] maximize lifetime expected utility

$$\mathcal{W}_t(j) = E_t \sum_{\tau=t}^{\infty} \beta^{\tau-t} \left(U(C_\tau(j)) - V(l_\tau(j)) \right),$$

where j indexes Home households. The utility of consumption is

$$U(C_t(j)) = Z_{U,t} \frac{(C_t(j) - bC_{t-1})^{1-\sigma} - 1}{1-\sigma}$$

where C_{t-1} is per-capita Home consumption, b is a habit persistence parameter, and $Z_{U,t}$ is a common preference shock. The disutility of labor is given by

$$V(l_t(j)) = Z_{V,t} \frac{l_t(j)^{1+\zeta}}{1+\zeta}$$

where $Z_{V,t}$ is a shock to labor disutility.

A Home household maximizes the lifetime expected utility $\mathcal{W}_t$ by choosing holdings of bonds denominated in home and foreign currency, money holdings, investment, consumption, and setting wages. Its budget constraint include money, bond yields, capital returns, yield of land, wages, consumption, investment, and non-distortionary taxes. Consumption, capital accumulation, wage setting, and foreign bond trading are subject to adjustment costs.

[15] The same formulas apply for Foreign country

4.2. Examined Monetary Rules

The monetary rules in the Home country simulated below are inflation forecast based (IFB) rule, and a similar rule employing stabilization of exchange rate. A general form of the both used rules is:

$$(1+i_{t+1})^4 - 1 = \omega_i((1+i_t)^4 - 1) + (1-\omega_i)E_t\left(\beta^{-4}\frac{P_{t+1}}{P_{t-3}}\right) + \omega_1 E_t\left(\frac{P_{t+1}}{P_{t-3}} - \Pi_{t+1}\right) + \omega_2 y_{gap} + \omega_3\left(\frac{\mathcal{E}^4_{t+1}}{\mathcal{E}^4_t} - 1\right)$$

where i_t is the interest rate, P_{t+1}/P_{t-3} is year-on-year CPI inflation one period into the future, Π_t is year-on-year CPI inflation target, y_{gap} is GDP gap, and $\mathcal{E}_t$ is a nominal exchange rate (units of Home currency per one unit of Foreign currency). We choose $\omega_i = 0.93$, $\omega_1 = 0.39$, and $\omega_2 = 0.26$. The first rule (referred as pure IFB) doesn't involve the exchange rate, $\omega_3 = 0$, the second rule reacts quite aggresively having $\omega_3 = 10$.

4.3. Simulation Results

We took the GEM calibration described in [Laxton and Pesenti, 2003] with a few modifications. Firstly, we modified the habit persistence parameter b. We changed it from $b = 0.95$ to $b = 0.3$ in the Home country. The reason is that $b = 0.95$ makes agents more value a positive change in consumption than its level. Thus in the stochastic steady state, the level of consumption doesn't matter for the lifetime expected utility.

Secondly, we changed a steady state value for stochastic process of the labour disutility preference shocks $Z_{V,t}$. We changed it from $\mu(Z_V) = 11$ to $\mu(Z_V) = 2$ leaving its counterpart, steady state of the consumption utility preference shocks, at $\mu(Z_U) = 1$. This was done also for Foreign country. In this way we put more weight on consumption in the utility function since this setting gives more realistic stochastic steady state of labour.

Thirdly, we changed the intertemporal elasticity of substitution σ from $\sigma = 1/3$. We simulate two cases, $\sigma = 2/3$, and $\sigma = 2$.

The lifetime expected utility $\mathcal{W}_t$ of the representative household was simulated by adding a new equation for a non-predetermined variable $\mathcal{W}_t$, i.e.

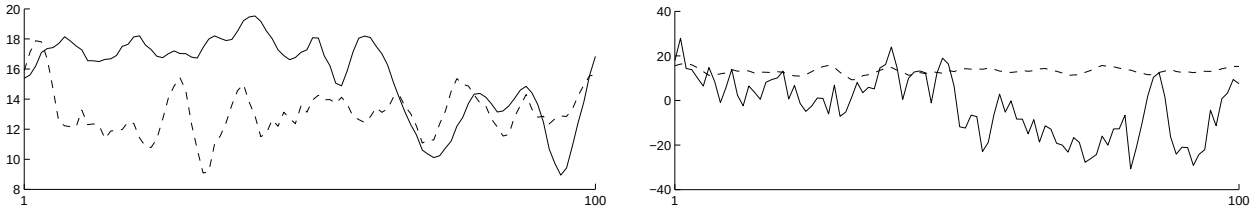
$$\mathcal{W}_t = U(C_t) - V(l_t) + E_t(\mathcal{W}_{t+1})$$

The simulations presented below are stochastic and were started at the deterministic steady state (the first period is dropped). The results draw a transition from an economy with no uncertainty (deterministic steady state) to a world with uncertainty (stochastic steady state). However, for our analysis the stochastic steady state is more important than the path of the transition.

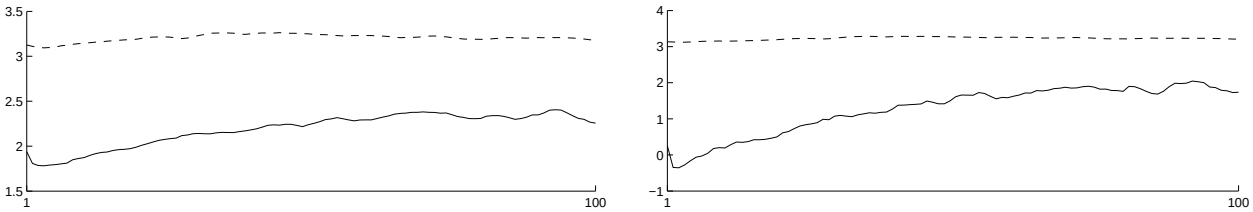
The following graphs display simulation results of the two rules for $\sigma = 2$. The graphs in the left column correspond to the pure IFB rule, right column corresponds to the rule

involving the exchange rate. Where applicable, a solid line is used for the Home country, dashed for the Foreign country.

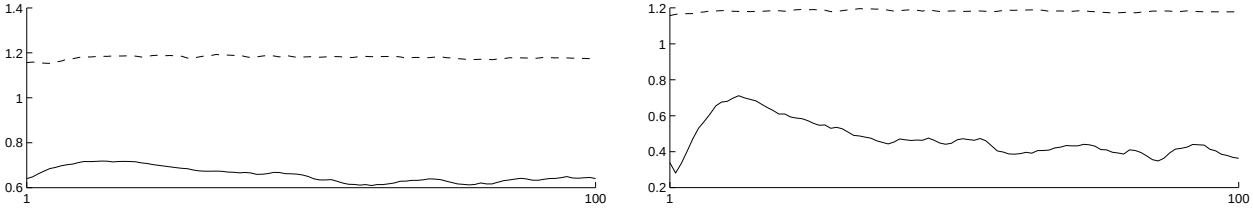
Nominal Interest Rate



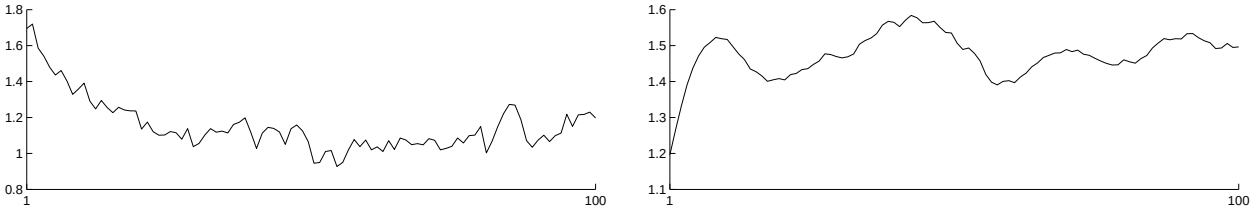
Consumption



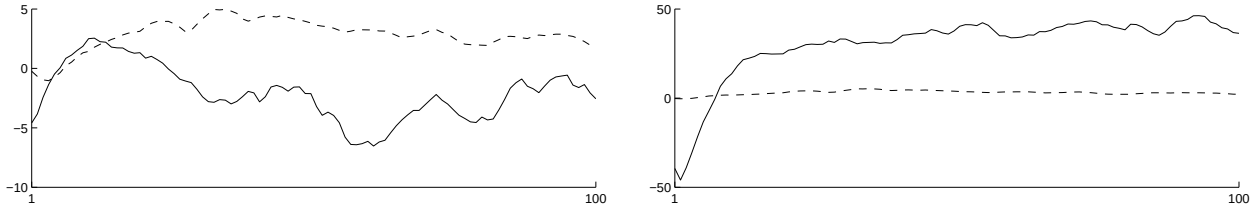
Labour



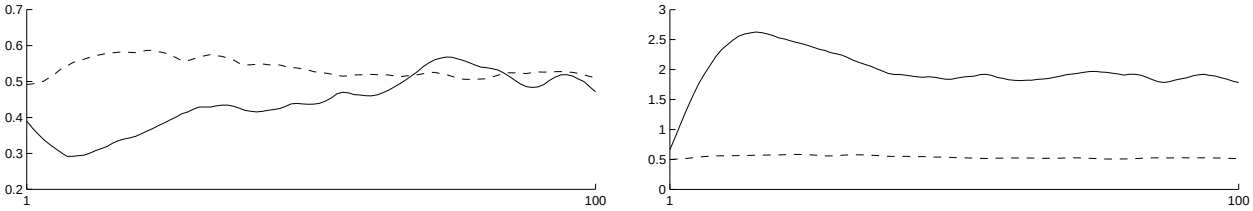
Real Exchange Rate



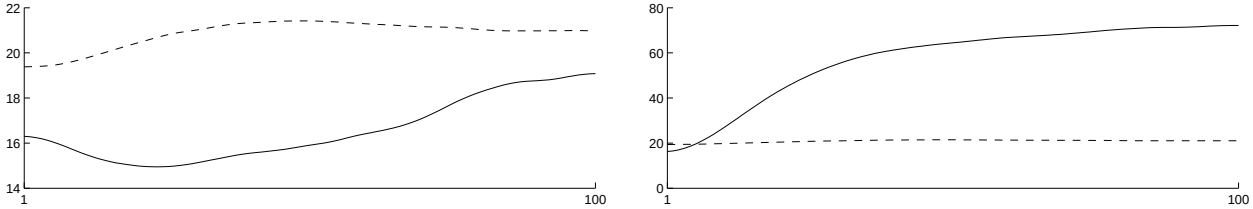
GDP Gap



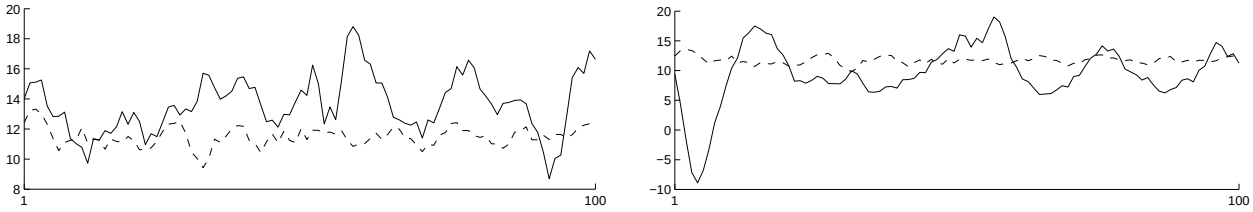
Investment



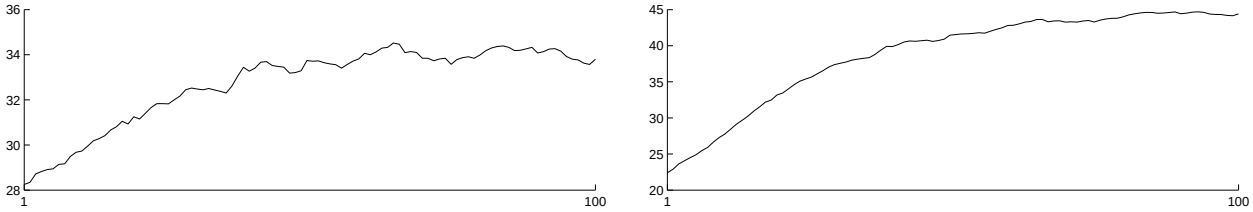
Total Capital Stock



CPI Inflation y-o-y

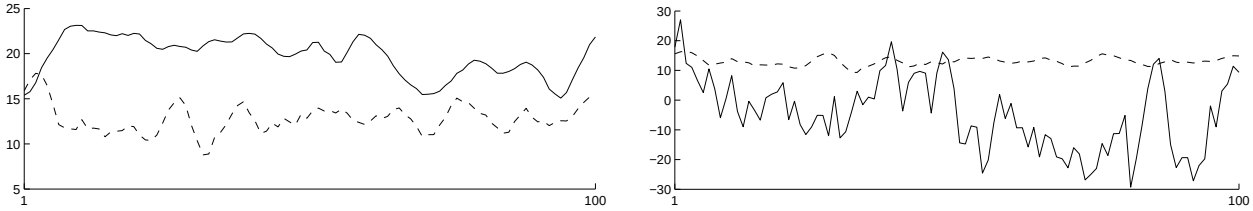


Lifetime Expected Utility of Home Households

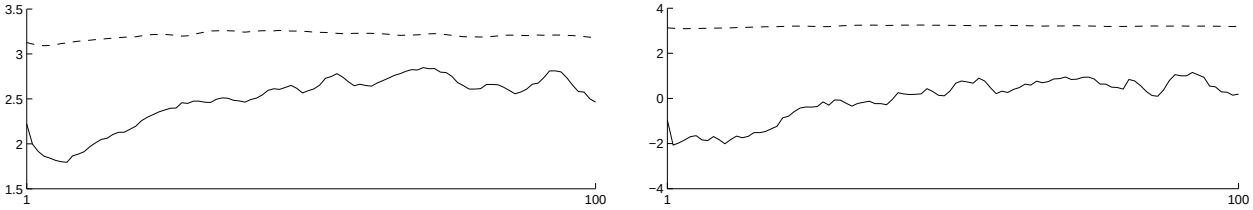


The following are results for $\sigma = 2/3$.

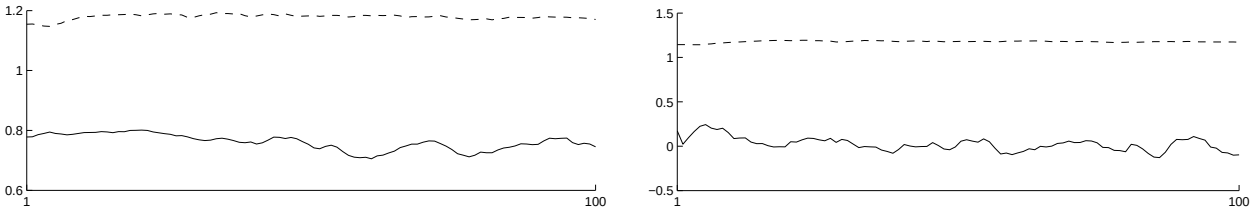
Nominal Interest Rate



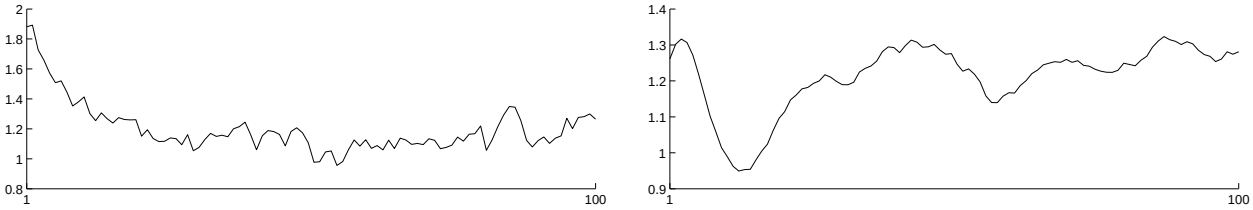
Consumption



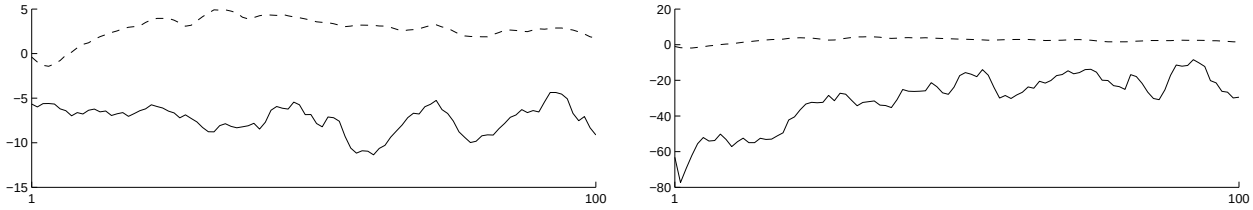
Labour



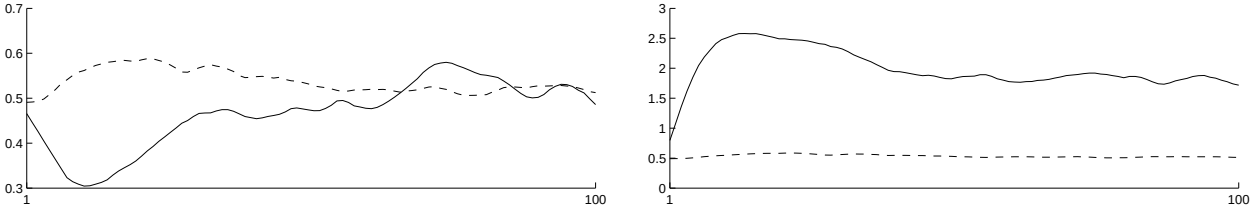
Real Exchange Rate

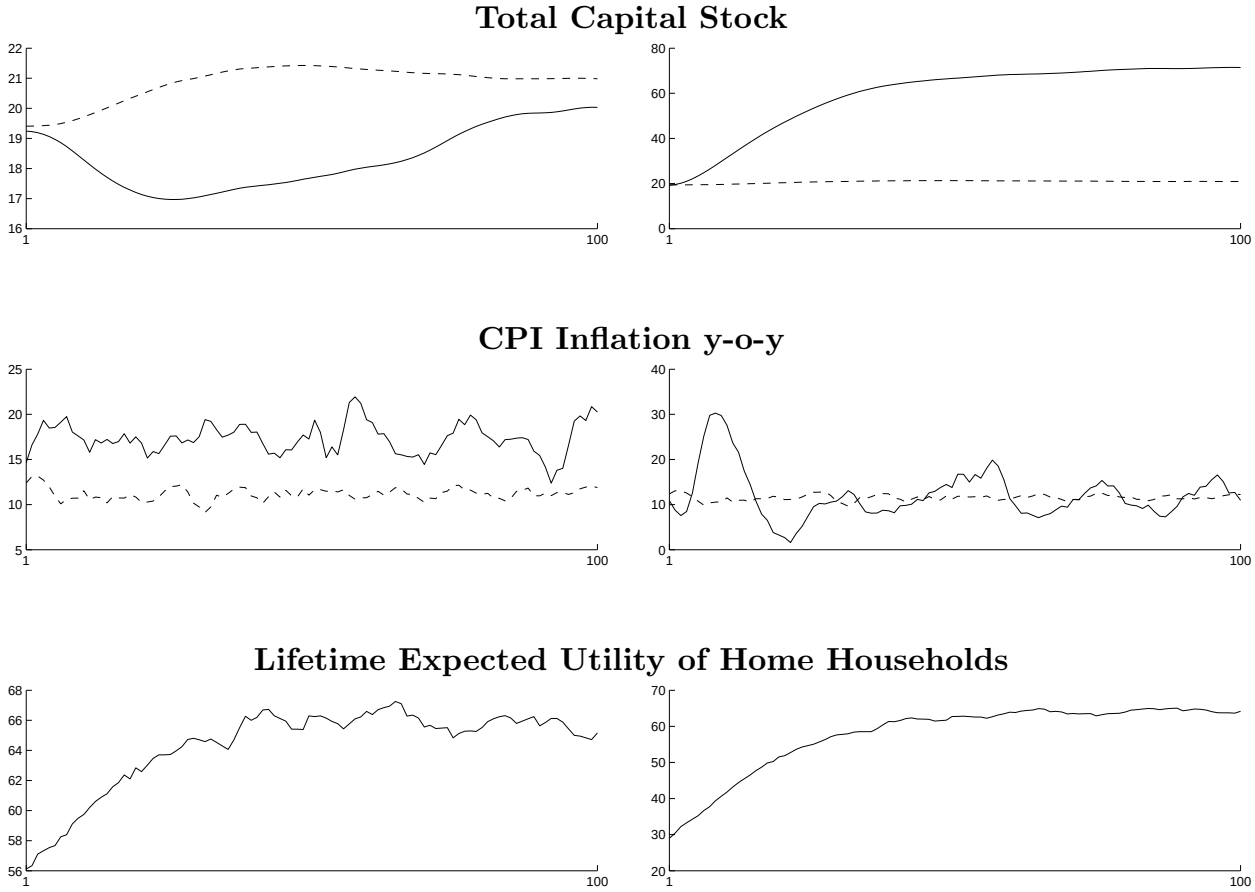


GDP Gap



Investment





4.4. Used Software Tools

For the simulations we used C++ Dynare prototype, currently under intensive development. Its main aim is to allow simulations of higher orders and/or larger models. It is almost compatible with its well known Matlab counterpart, Matlab Dynare (see [Collard and Juillard, 2003]). The C++ source code was provided by Michel Juillard. For the solution of the Sylvester equation we used the author's C++ implementation of the recursive algorithm.

For some parametrizations, the C++ prototype failed to solve deterministic steady state. In these cases, Matlab Dynare was used to calculate the steady state.

4.5. Conclusions and Open Questions

In terms of initial lifetime expected utility, the IFB rule performs better in both cases. For $\sigma = 2$, the initial value of the lifetime expected utility $\mathcal{W}_1$ is 28 for pure IFB which is greater than 25 for the exchange rate rule. For $\sigma = 2/3$, the initial value $\mathcal{W}_1$ is 56 for pure IFB against 30 for exchange rate rule.

However, if the households are given enough time to accommodate, the IFB rule performs better only for economy with $\sigma = 2/3$, yielding stochastic steady state of $\mathcal{W}_t$ equal to 66 for pure IFB and 60 for the exchange rate rule. If $\sigma = 2$, than the stochastic steady state of $\mathcal{W}_t$ is 34 for pure IFB, and 44 for the exchange rate rule.

Why the pure IFB rule performs better than the exchange rate rule for $\sigma = 2/3$, and worse for $\sigma = 2$? It is not trivial to find an answer to this, and the question is left open.

More problems evolve if one wants to calibrate a model with respect to the stochastic steady state. So far, a common practice is to calibrate a model and check that the model yields a reasonable determinsitic steady state. A typical example is a calibration of the weight between consumption utility and labour disutility, it is chosen so that the labor in the deterministic steady state is (for instance) 1/4 of the total available time. In this way, a deterministic steady version of (1) with desired values of some variables at deterministic steady state gives a constraint on the model parameters. On the other hand, if one wants to calibrate a model with respect to some desired values of variables at stochastic steady state (which is a correct way), he faces much more difficult problem, since the equation (1) involves integration over all stochastic shocks. Moreover, the variances of the shocks enter to the problem as additional parameters, which cannot be set independently on calibration of the model parameters.

In our view, a correct non-linear SDGE simulation cannot be effectively done without a computation tool helping to calculate the stochastic steady state and calibrate the model with respect to the desired values of some state variables at the stochastic steady state.

Appendix A. Block Diagonalization Algorithm

A goal of the following is to give a description of the algorithm used for block diagonalization of matrix C. This is a similarity transformation $C = VFV^{-1}$, where F is a block diagonal matrix with quasi-triangular blocks. The less the sizes of blocks are, the more zeros are above diagonal in F.

The initial step of the block diagonalization algorithm is the real Schur decomposition $C = V_1 F_1 V_1^T$.

For the following steps, consider a similarity transformation of a block triangular matrix:

$$\begin{pmatrix} I & Q \\ 0 & I \end{pmatrix} \begin{pmatrix} R & S \\ 0 & T \end{pmatrix} \begin{pmatrix} I & -Q \\ 0 & I \end{pmatrix} = \begin{pmatrix} R & S + QT - RQ \\ 0 & T \end{pmatrix}.$$

If a solution Q to the Sylvester equation $S = RQ - QT$ is found, then the above similarity transformation breaks the block triangular matrix into a block diagonal matrix. Therefore, the following steps $F_i = V_{i+1} F_{i+1} V_{i+1}^{-1}$ correspond to the above equation. In general, at each step we choose a diagonal block of F_i and break it by solving the above Sylvester equation for Q. Using Q we form V_{i+1} (and V_{i+1}^{-1}), and when the process is finished, V is a product of all V_i (V^{-1} likewise). However, Q as the solution of $S = RQ - QT$ can be very large, making V_{i+1}, and V_{i+1}^{-1} ill conditioned. The large size of Q is implied by large S, and by insufficiently separated R, and T. The latter can be improved by a different eigenvalue ordering. An eigenvalue reordering can be done by orthogonal similarity transformation not worsening the condition number of resulting V.

Intuitivelly, an eigenvalue reordering besides the separation of R and T also changes size of S, so it is very diffucult to algorithmically predict the size of Q. It is not feasible to try all possible orderings, that's why we use a heuristics due to [Bavely and Stewart, 1979]. We take the first eigenvalue[16] as matrix R, and T as the rest. Then we calculate Q. If the greatest absolute value of Q's elements is less then a user given threshold, we break the matrix, and carry on with T. Otherwise, we select a suitable eigenvalue from T, and incorporate it to R by the eigenvalue reordering. Another attempt to break the matrix is made then. The suitable eigenvalue is selected so that its size would be closest to the average eigenvalue size [17] of matrix R, since it is likely that such an eigenalue is guilty for the bad separation.

It is difficult to link the user given threshold for element size in Q with the condition number of the resulting V. Therefore, in our implementation, the error of the similarity transformation $C = VFV^{-1}$ is reported giving a feedback on the threshold to the user.

A comment must be made here regarding the eigenvalue swapping. As it is discussed in [Dongarra et al., 1992], swapping of ill conditioned eigenvalues or two close eigenvalues can turn out not possible with respect to computer precision. In this case, such eigenvalues are brought into R together avoiding their unstable swaps. Another difficulty pointed by [Dongarra et al., 1992] is that a swap can turn an ill conditioned complex eigenvalue into

[16] real eigenvalues correspond to 1×1 matrices, complex ones to 2×2

[17] complex pairs are counted only once in the average

two real. Extremely, a swap of two complex eigenvalues can yield four real eigenvalues. In such cases, the algorithm proceeds normally, but the implementation must be able to recognize it.

Appendix B. The Recursive Algorithm in Detail

Lemma 1. *For any $n \times n$ matrix A and $\beta_1\beta_2 > 0$, if there is exactly one solution of*

$$\left(I_2 \otimes I_n + \begin{pmatrix} \alpha & \beta_1 \\ -\beta_2 & \alpha \end{pmatrix} \otimes A\right) \begin{pmatrix} y_1 \\ y_2 \end{pmatrix} = \begin{pmatrix} d_1 \\ d_2 \end{pmatrix},$$

then it can be obtained as solution of

$$\left(I_n + 2\alpha A + (\alpha^2 + \beta^2)A^2\right) y_1 = \widehat{d_1}$$
$$\left(I_n + 2\alpha A + (\alpha^2 + \beta^2)A^2\right) y_2 = \widehat{d_2}$$

where $\beta = \sqrt{\beta_1\beta_2}$, and

$$\begin{pmatrix} \widehat{d_1} \\ \widehat{d_2} \end{pmatrix} = \left(I_2 \otimes I_n + \begin{pmatrix} \alpha & -\beta_1 \\ \beta_2 & \alpha \end{pmatrix} \otimes A\right) \begin{pmatrix} d_1 \\ d_2 \end{pmatrix}$$

Proof. Since

$$\begin{pmatrix} \alpha & \beta_1 \\ -\beta_2 & \alpha \end{pmatrix} \begin{pmatrix} \alpha & -\beta_1 \\ \beta_2 & \alpha \end{pmatrix} = \begin{pmatrix} \alpha & -\beta_1 \\ \beta_2 & \alpha \end{pmatrix} \begin{pmatrix} \alpha & \beta_1 \\ -\beta_2 & \alpha \end{pmatrix} = \begin{pmatrix} \alpha^2 + \beta^2 & 0 \\ 0 & \alpha^2 + \beta^2 \end{pmatrix},$$

it is easy to see that if the equation is multiplied by

$$I_2 \otimes I_n + \begin{pmatrix} \alpha & -\beta_1 \\ \beta_2 & \alpha \end{pmatrix} \otimes A$$

we obtain the result. We only need to prove that the matrix is regular. But this is clear because matrix

$$\begin{pmatrix} \alpha & -\beta_1 \\ \beta_2 & \alpha \end{pmatrix}$$

collapses an eigenvalue of A to -1 iff the matrix

$$\begin{pmatrix} \alpha & \beta_1 \\ -\beta_2 & \alpha \end{pmatrix}$$

does. □

Lemma 2. *For any $n \times n$ matrix A and $\delta_1\delta_2 > 0$, if there is exactly one solution of*

$$\left(I_2 \otimes I_n + 2\alpha \begin{pmatrix} \gamma & \delta_1 \\ -\delta_2 & \gamma \end{pmatrix} \otimes A + (\alpha^2 + \beta^2) \begin{pmatrix} \gamma & \delta_1 \\ -\delta_2 & \gamma \end{pmatrix}^2 \otimes A^2\right) \begin{pmatrix} y_1 \\ y_2 \end{pmatrix} = \begin{pmatrix} d_1 \\ d_2 \end{pmatrix}$$

it can be obtained as

$$\left(I_n + 2a_1A + (a_1^2 + b_1^2)A^2\right)\left(I_n + 2a_2A + (a_2^2 + b_2^2)A^2\right) y_1 = \widehat{d_1}$$
$$\left(I_n + 2a_1A + (a_1^2 + b_1^2)A^2\right)\left(I_n + 2a_2A + (a_2^2 + b_2^2)A^2\right) y_2 = \widehat{d_2},$$

where

$$\begin{pmatrix} \widehat{d_1} \\ \widehat{d_2} \end{pmatrix} = \left(I_2 \otimes I_n + 2\alpha \begin{pmatrix} \gamma & -\delta_1 \\ \delta_2 & \gamma \end{pmatrix} \otimes A + (\alpha^2 + \beta^2) \begin{pmatrix} \gamma & -\delta_1 \\ \delta_2 & \gamma \end{pmatrix}^2 \otimes A^2 \right) \begin{pmatrix} d_1 \\ d_2 \end{pmatrix}$$

and

$$\begin{aligned} a_1 &= \alpha\gamma - \beta\delta \\ b_1 &= \alpha\delta + \gamma\beta \\ a_2 &= \alpha\gamma + \beta\delta \\ b_2 &= \alpha\delta - \gamma\beta \\ \delta &= \sqrt{\delta_1\delta_2} \end{aligned}$$

Proof. The matrix can be written as

$$\left(I_2 \otimes I_n + (\alpha + \mathrm{i}\beta) \begin{pmatrix} \gamma & \delta_1 \\ -\delta_2 & \gamma \end{pmatrix} \otimes A \right) \left(I_2 \otimes I_n + (\alpha - \mathrm{i}\beta) \begin{pmatrix} \gamma & \delta_1 \\ -\delta_2 & \gamma \end{pmatrix} \otimes A \right).$$

Note that the both matrices are regular since their product is regular. For the same reason as in the previous proof, the following matrix is also regular

$$\left(I_2 \otimes I_n + (\alpha + \mathrm{i}\beta) \begin{pmatrix} \gamma & -\delta_1 \\ \delta_2 & \gamma \end{pmatrix} \otimes A \right) \left(I_2 \otimes I_n + (\alpha - \mathrm{i}\beta) \begin{pmatrix} \gamma & -\delta_1 \\ \delta_2 & \gamma \end{pmatrix} \otimes A \right),$$

and we may multiply the equation by this matrix obtaining $\widehat{d_1}$ and $\widehat{d_2}$. Note that the four matrices commute, that is why we can write the whole product as

$$\begin{aligned} &\left(I_2 \otimes I_n + (\alpha + \mathrm{i}\beta) \begin{pmatrix} \gamma & \delta_1 \\ -\delta_2 & \gamma \end{pmatrix} \otimes A \right) \cdot \left(I_2 \otimes I_n + (\alpha + \mathrm{i}\beta) \begin{pmatrix} \gamma & -\delta_1 \\ \delta_2 & \gamma \end{pmatrix} \otimes A \right) \cdot \\ &\left(I_2 \otimes I_n + (\alpha - \mathrm{i}\beta) \begin{pmatrix} \gamma & \delta_1 \\ -\delta_2 & \gamma \end{pmatrix} \otimes A \right) \cdot \left(I_2 \otimes I_n + (\alpha - \mathrm{i}\beta) \begin{pmatrix} \gamma & -\delta_1 \\ \delta_2 & \gamma \end{pmatrix} \otimes A \right) = \\ &\quad \left(I_2 \otimes I_n + 2(\alpha + \mathrm{i}\beta) \begin{pmatrix} \gamma & 0 \\ 0 & \gamma \end{pmatrix} \otimes A + (\alpha + \mathrm{i}\beta)^2 \begin{pmatrix} \gamma^2 + \delta^2 & 0 \\ 0 & \gamma^2 + \delta^2 \end{pmatrix} \otimes A^2 \right) \cdot \\ &\quad \left(I_2 \otimes I_n + 2(\alpha - \mathrm{i}\beta) \begin{pmatrix} \gamma & 0 \\ 0 & \gamma \end{pmatrix} \otimes A + (\alpha - \mathrm{i}\beta)^2 \begin{pmatrix} \gamma^2 + \delta^2 & 0 \\ 0 & \gamma^2 + \delta^2 \end{pmatrix} \otimes A^2 \right) \end{aligned}$$

The product is a diagonal consiting of two $n \times n$ blocks, which are the same. The block can be rewritten as product:

$$\begin{aligned} &(I_n + (\alpha + \mathrm{i}\beta)(\gamma + \mathrm{i}\delta)A) \cdot (I_n + (\alpha + \mathrm{i}\beta)(\gamma - \mathrm{i}\delta)A) \cdot \\ &(I_n + (\alpha - \mathrm{i}\beta)(\gamma + \mathrm{i}\delta)A) \cdot (I_n + (\alpha - \mathrm{i}\beta)(\gamma - \mathrm{i}\delta)A) \end{aligned}$$

and after reordering

$$\begin{aligned}&(I_n + (\alpha + \mathrm{i}\beta)(\gamma + \mathrm{i}\delta)A) \cdot (I_n + (\alpha - \mathrm{i}\beta)(\gamma - \mathrm{i}\delta)A) \cdot \\ &(I_n + (\alpha + \mathrm{i}\beta)(\gamma - \mathrm{i}\delta)A) \cdot (I_n + (\alpha - \mathrm{i}\beta)(\gamma + \mathrm{i}\delta)A) = \\ &\qquad (I_n + 2(\alpha\gamma - \beta\delta)A + (\alpha^2 + \beta^2)(\gamma^2 + \delta^2)A^2) \cdot \\ &\qquad (I_n + 2(\alpha\gamma + \beta\delta)A + (\alpha^2 + \beta^2)(\gamma^2 + \delta^2)A^2)\end{aligned}$$

Now it suffices to compare $a_1 = \alpha\gamma - \beta\delta$ and verify that

$$\begin{aligned}b_1^2 &= (\alpha^2 + \beta^2)(\gamma^2 + \delta^2) - a_1^2 = \\ &= \alpha^2\gamma^2 + \beta^2\gamma^2 + \alpha^2\beta^2 + \beta^2\delta^2 - (\alpha\gamma)^2 + 2\alpha\beta\gamma\delta - (\beta\delta)^2 = \\ &= (\beta\gamma)^2 + (\alpha\beta)^2 + 2\alpha\beta\gamma\delta = \\ &= (\beta\gamma + \alpha\beta)^2\end{aligned}$$

For b_2 it is done in the same way. □

Here we describe the recursive algorithm solving (6) in more technical detail. We define three functions (which call recursively each other) of which $\mathrm{vec}(Y) = \mathbf{solv1}(1, \mathrm{vec}(\widehat{D}), k)$ provides the solution Y.

In the following text, we retain m denoting dimension of F, n is dimension of K. Let m_c be a number of complex eigenvalues pairs of F, and m_r a number of real eigenvalues, thus $m = m_r + 2m_c$. Additionally, F_j will denote j-th diagonal block of F^T, this is 1×1 or 2×2 matrix depending on j-th eigenvalue of F for $j = 1, \ldots, m_c + m_r$. For a fixed j, let $\bar{\jmath}$ denote an index of the first column of F_j in F^T. Finally, whenever we write something like $(I_{m^k} \otimes I_n + r \cdot F_{[k]})y = d$, y and d denote column vectors of appropriate dimensions, and $y_{\bar{\jmath}}$ is $\bar{\jmath}$-th partition of y (similarly for $d_{\bar{\jmath}}$). If j-th eigenvalue is real, y_j denotes $y_{\bar{\jmath}}$, and if it is complex, y_j denotes a double size vector of stacked vectors $y_{\bar{\jmath}}$, and $y_{\bar{\jmath}+1}$. Similarly for d.

Function solv1.

The function $y = \mathbf{solv1}(r, d, k)$ solves equation

$$(I_{m^k} \otimes I_n + r \cdot F_{[k]})\, y = d.$$

The function proceedes as follows:

1) If $k = 0$, the equation is solved directly, K is upper quasi-triangular matrix, so this is easy.
2) If $k > 0$, then we go through all diagonal blocks F_j for $j = 1, \ldots, m_r + m_c$ and perform:
   a) If $F_j = (f_{\bar{\jmath}\bar{\jmath}}) = (f)$, then we return $y_j = y_{\bar{\jmath}} = \mathbf{solv1}(rf, d_{\bar{\jmath}}, k-1)$. Then we precalculate $z = r \cdot F_{[k-1]} y_j$, and eliminate elements below F_j. This is, for each $\bar{\imath} = \bar{\jmath} + 1, \ldots, m$, we put
   $$d_{\bar{\imath}} = d_{\bar{\imath}} - f_{\bar{\jmath}\bar{\imath}} z$$

b) If $F_j = \begin{pmatrix} \alpha & \beta_1 \\ -\beta_2 & \alpha \end{pmatrix}$, we return $y_j = \mathbf{solv2}(r\alpha, r\beta_1, r\beta_2, d_j, k-1)$. Then we precalculate

$$z_1 = r \cdot F_{[k-1]} y_{\bar{\jmath}}, \quad \text{and } z_2 = r \cdot F_{[k-1]} y_{\bar{\jmath}+1}$$

and eliminate elements below F_j. This is, for each $\bar{\imath} = \bar{\jmath} + 2, \ldots, m$ we put

$$d_{\bar{\imath}} = d_{\bar{\imath}} - f_{\bar{\jmath}\bar{\imath}} z_1 - f_{\bar{\jmath}+1\bar{\imath}} z_2$$

Function solv2.

The function $y = \mathbf{solv2}(\alpha, \beta_1, \beta_2, d, k)$ solves equation

$$\left(I_2 \otimes I_{m^k} \otimes I_n + \begin{pmatrix} \alpha & \beta_1 \\ -\beta_2 & \alpha \end{pmatrix} \otimes F_{[k]} \right) y = d$$

According to **Lemma 1** the function returns

$$y = \begin{pmatrix} \mathbf{solv2p}(\alpha, \beta_1\beta_2, \widehat{d}_1, k) \\ \mathbf{solv2p}(\alpha, \beta_1\beta_2, \widehat{d}_2, k) \end{pmatrix},$$

where $\widehat{d}_1$, and $\widehat{d}_2$ are partitions of $\widehat{d}$ from the lemma.

Function solv2p.

The function $y = \mathbf{solv2p}(\alpha, \beta^2, d, k)$ solves equation

$$\left(I_{m^k} \otimes I_n + 2\alpha F_{[k]} + (\alpha^2 + \beta^2) F_{[k]}^2 \right) y = d$$

The function proceedes as follows:

1) If $k = 0$, the matrix $I_n + 2\alpha K + (\alpha^2 + \beta^2) K^2$ is calculated and the solution is obtained directly.
2) If $k > 0$, note that the diagonal blocks of F^{2T} are of the form F_j^2, since if the F^T is block partitioned according to diagonal blocks, then it is lower triangular. So we go through all the diagonal blocks F_j for $j = 1, \ldots, m_r + m_c$ and perform:

   a) If $F_j = (f_{\bar{\jmath}\bar{\jmath}}) = (f)$ then j-th diagonal block of

   $$I_{m^k} \otimes I_n + 2\alpha \cdot F_{[k]} + (\alpha^2 + \beta^2) \cdot F_{[k]}^2$$

   takes the form

   $$I_{m^{k-1}} \otimes I_n + 2\alpha f \cdot F_{[k-1]} + (\alpha^2 + \beta^2) f^2 \cdot F_{[k-1]}^2$$

   and we can put $y_j = y_{\bar{\jmath}} = \mathbf{solv2p}(f\alpha, f^2\beta^2, d_j, k-1)$.

   Then we need to eliminate guys below F_j. We precalculate

   $$z = 2\alpha \cdot F_{[k-1]} y_j, \quad \text{and } w = (\alpha^2 + \beta^2) \cdot F_{[k-1]}^2 y_j$$

and eliminate, this is for all $\bar{\imath} = \bar{\jmath} + 1, \dots, m$ we put

$$d_{\bar{\imath}} = d_{\bar{\imath}} - f_{\bar{\jmath}\bar{\imath}} z - g_{\bar{\jmath}\bar{\imath}} w,$$

where $g_{\bar{\jmath}\bar{\imath}}$ denotes an element of F^{2T} at position $(\bar{\imath}, \bar{\jmath})$.

b) If $F_j = \begin{pmatrix} \gamma & \delta_1 \\ -\delta_2 & \gamma \end{pmatrix}$, then j-th diagonal block of

$$I_{m^k} \otimes I_n + 2\alpha \cdot F_{[k]} + (\alpha^2 + \beta^2) \cdot F_{[k]}^2$$

takes the form

$$I_{m^{k-1}} \otimes I_n + 2\alpha \begin{pmatrix} \gamma & \delta_1 \\ -\delta_2 & \gamma \end{pmatrix} F_{[k-1]} + (\alpha^2 + \beta^2) \begin{pmatrix} \gamma & \delta_1 \\ -\delta_2 & \gamma \end{pmatrix}^2 F_{[k-1]}^2$$

According to **Lemma 2**, we need to calculate $\widehat{d_{\bar{\jmath}}}$, and $\widehat{d_{\bar{\jmath}+1}}$, and a_1, b_1, a_2, b_2. Then we obtain

$$y_{\bar{\jmath}} = \mathbf{solv2p}(a_1, b_1^2, \mathbf{solv2p}(a_2, b_2^2, \widehat{d_{\bar{\jmath}}}, k-1), k-1)$$
$$y_{\bar{\jmath}+1} = \mathbf{solv2p}(a_1, b_1^2, \mathbf{solv2p}(a_2, b_2^2, \widehat{d_{\bar{\jmath}+1}}, k-1), k-1)$$

Now we need to eliminate guys below F_j. First, we precalculate

$$z_1 = 2\alpha \cdot F_{[k-1]} y_{\bar{\jmath}} \quad z_2 = 2\alpha \cdot F_{[k-1]} y_{\bar{\jmath}+1}$$
$$w_1 = (\alpha^2 + \beta^2) \cdot F_{[k-1]}^2 y_{\bar{\jmath}} \quad w_2 = (\alpha^2 + \beta^2) \cdot F_{[k-1]}^2 y_{\bar{\jmath}+1}$$

Then we go through all $\bar{\imath} = \bar{\jmath} + 2, \dots, m$ and put:

$$d_{\bar{\imath}} = d_{\bar{\imath}} - f_{\bar{\jmath}\bar{\imath}} z_1 - f_{\bar{\jmath}+1\bar{\imath}} z_2 - g_{\bar{\jmath}\bar{\imath}} w_1 - g_{\bar{\jmath}+1\bar{\imath}} w_2$$

5. References

Anderson, E. W. et al. (1996). *Handbook of Computational Economics*, volume 1, chapter Mechanics of Forming and Estimating Dynamic Linear Economies. Elsevier Science.

Bartels, R. H. and Stewart, G. W. (1972). Solution of the equation $ax+xb = c$. *Comm. ACM*, 15:820–26.

Bavely, C. A. and Stewart, G. W. (1979). An algorithm for computing reducing subspaces by block diagonalization. *SIAM Journal on Numerical Analysis*, 16(2):359–367.

Collard, F. and Juillard, M. (2003). Stochastic simulations with DYNARE. A practical guide. http://www.cepremap.cnrs.fr/~michel/dynare/guide.pdf, DYNARE Home Page http://www.cepremap.cnrs.fr/~michel/dynare.

Dongarra, J. J., Hammarling, S., and Wilkinson, J. H. (1992). Numerical considerations in computing invariant subspaces. *SIAM Journal on Matrix Analysis and Applications*, 13(1):145–161.

Golub, G. H. and Loan, C. F. V. (1996). *Matrix Computations*. The Johns Hopkins University Press, Baltimore, Maryland, third edition.

Jin, H.-H. and Judd, K. (2002). Perturbation methods for general dynamic stochastic models. Unpublished manuscript.

Juillard, M. (2003). Solving stochastic dynamic equilibrium models: A k-order perturbation approach. http://www.stanford.edu/groups/SITE/Juillard.pdf.

Laxton, D. and Pesenti, P. (2003). Monetary rules for small, open, emerging economies. *NBER Working Paper 9568*. http://www.nber.org/papers/9568.