národní
úložiště
šedé
literatury

Solving SDGE models

Kameník, Ondřej
2005

WORKING PAPER SERIES 10

2005

Ondřej Kameník:
Solving SDGE Models: A New Algorithm for the Sylvester Equation

CNB CZECH NATIONAL BANK

WORKING PAPER SERIES

Solving SDGE Models:
A New Algorithm for the Sylvester Equation

Ondřej Kameník

10/2005

CNB WORKING PAPER SERIES

The Working Paper Series of the Czech National Bank (CNB) is intended to disseminate the results of the CNB's research projects as well as the other research activities of both the staff of the CNB and collaborating outside contributors. The Series aims to present original research contributions relevant to central banks. It is refereed internationally. The referee process is managed by the CNB Research Department. The working papers are circulated to stimulate discussion. The views expressed are those of the authors and do not necessarily reflect the official views of the CNB.

Printed and distributed by the Czech National Bank. Available at http://www.cnb.cz.

Reviewed by: Michal Hlaváček (Czech National Bank)
Jiří Zelinka (Masaryk University, Brno)
Karel Žitný (Charles University, Prague)

Project Coordinator: Martin Cincibuch

Solving SDGE Models:

A New Algorithm for the Sylvester Equation

Ondřej Kameník*

Abstract

This paper presents a new numerical algorithm for solving the Sylvester equation involved in higher-order perturbation methods developed for solving stochastic dynamic general equilibrium models. The new algorithm surpasses other methods used so far (including the very popular doubling algorithm) in terms of computational time, memory consumption, and numerical stability.

JEL Codes: C63, C68.
Keywords: Dynamic general equilibrium, doubling algorithm, perturbation approach, recursive algorithm.

* Ondřej Kameník, Czech National Bank, (ondrej.kamenik@cnb.cz).
I wish to thank Michel Juillard for his invaluable help during this project and for providing the source code of the Dynare C++ prototype, Douglas Laxton for providing the Dynare code of the Czech–EU calibration of the GEM, and an anonymous referee for valuable comments. Any remaining errors are my own.

Nontechnical Summary

The perturbation approach to solving stochastic dynamic general equilibrium (SDGE) models seems very promising in comparison with other methods, since it is able to deal with a large state space. According to Jin and Judd (2002) and Juillard (2003), the solution of a k-order approximation consists in finding the non-stochastic steady state and then solving for a k-order approximation around the steady state. The k-order approximation is found iteratively from the lower-order approximations. The iterative process begins with a standard solution of a linear approximation ($k = 1$). The iterative step consists of a series of ordinary linear systems and a series of a linear Sylvester equation. While the ordinary linear equations present no numerical problems, the Sylvester equation is a computational challenge. Although the size of the input data of all the equations is similar, the Sylvester equation requires the solution of a large matrix whose size grows exponentially with k and polynomially with the number of predetermined variables.

The main purpose of this paper is to develop a recursive algorithm for solving the Sylvester equation involved in the k-order approximations. The algorithm is implemented in C++ and tested. The efficiency of the recursive algorithm is shown in comparison with two methods used so far: a method due to Bartels and Stewart (1972) implemented by Michel Juillard, and a doubling algorithm implemented by me. Since the application of the Bartels–Stewart approach in the context of SDGE models does not account for the 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. 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 illustrated with the solution to a second-order approximation to the Global Economy Model (GEM), calibrated for the Czech Republic and the euro area Laxton and Pesenti (2003). In this particularly difficult model, the doubling algorithm yields a relative error of the equation residual of order 10^{-1}, whereas the new recursive method is of order 10^{-14}.

1. Introduction

I examine discrete time models of the form[1]

$$E_t\left(f(y^*_{t-1}, y_t, y^{**}_{t+1}, u_t)\right) = 0, \tag{1.1}$$

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

$$u_t = \sigma\eta_t.$$

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

$$y_t = g(y^*_{t-1}, u_t, \sigma),$$

which contains the decision rules for the 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 the unknown future through serially uncorrelated shocks u_{t+1}; this is

$$\begin{aligned} 0 &= E_t\left(F(y^*_{t-1}, u_t, \sigma, u_{t+1})\right) \\ &= E_t\left(f\left(y^*_{t-1}, g(y^*_{t-1}, u_t, \sigma), g^{**}\big(g^*(y^*_{t-1}, u_t, \sigma), u_{t+1}, \sigma\big), u_t\right)\right). \end{aligned} \tag{1.2}$$

Now suppose that I 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 u. The goal of the k-order perturbation method at this point is to obtain the k-order derivatives of g with respect to y^*, u, and σ, including all possible cross derivatives.

In order to handle higher dimensional objects like k-order derivatives, I use a tensor bracket notation. The upper one-dimensional index relates to the derived equation, and the lower multidimensional index relates to the variables. I also use the Einstein summation notation, i.e., $[[A]_\gamma \cdot [B]^\gamma]^\alpha_\beta = \sum_\gamma [A]^\alpha_\gamma [B]^\gamma_\beta$, and its more general variant $\left[[A]_{\gamma\ldots\gamma_n} \prod_{i=1}^n [B]^{\gamma_i}_{\beta_i}\right]^\alpha = \sum_{\gamma_1\ldots\gamma_n} [A]^\alpha_{\gamma_1\ldots\gamma_n} \prod_{i=1}^n [B]^{\gamma_i}_{\beta_i}$.

The Sylvester equations evolve when the derivatives $\left[g_{y^{*k}}\right]$ and $\left[g_{y^{*k-i}\sigma^i}\right]$ are retrieved. As shown in Juillard (2003), the derivatives $\left[g_{y^{*k}}\right]$ are given by

$$\left[F_{y^{*k}}\right]^i_{\alpha_1\ldots\alpha_k} = 0, \text{ for } i = 1,\ldots,n,\ \alpha_j = 1,\ldots,m,$$

[1] I use the same notation as in Juillard (2003).

which, after separating the unknowns takes the form:

$$[f_{y^*}]^i_\beta \cdot \left[g^*_{y^{*k}}\right]^\beta_{\alpha_1\dots\alpha_k} + [f_{y^{**}}]^i_\beta \cdot \left[g^{**}_{y^{*k}}\right]^\beta_{\alpha_1\dots\alpha_k} + \left[f_{y^{**}_+} g^{**}_{y^*}\right]^i_\beta \cdot \left[g^*_{y^{*k}}\right]^\beta_{\alpha_1\dots\alpha_k} + \left[f_{y^{**}_+}\right]^i_\beta \cdot \left[\left[g^{**}_{y^{*k}}\right]_{\gamma_1\dots\gamma_k} \prod_{j=1}^{k} \left[g^*_{y^*}\right]^{\gamma_j}_{\alpha_j}\right]^\beta = [D]^i_{\alpha_1\dots\alpha_k}. \tag{1.3}$$

If the higher dimensional tensors are unfolded column-wise, I obtain an equation of the form:

$$AX + BX\left(\otimes^k C\right) = D_k, \tag{1.4}$$

where

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

This equation presents a core for each k-order step of the perturbation approach. The derivatives $\left[g_{y^{*k-i}\sigma^i}\right]$ are obtained as a solution to the same type of Sylvester equation.

The remainder of the paper is organized as follows: The second section introduces the new recursive algorithm, solving equation (1.4), and analyzes memory consumption and computational complexity. The third section compares the recursive algorithm with the Bartels–Stewart approach and the doubling algorithm. It shows that the recursive algorithm is better than the two alternatives in terms of time, memory, and numerical stability. The conclusions are followed by three appendices. The first appendix describes the algorithm used for block diagonalizing a matrix. The second provides a formal description of the recursive algorithm. The third discusses the implementation of the Kronecker product used for the recursive and doubling algorithms.

2. The Recursive Algorithm

2.1 Complexity of the Problem

Before we start with a description of the algorithm, it is useful to assess the computational complexity of solving (1.4). Recall that n is the number of equations in the model (1.1), and m is the number of predetermined variables. A and B in (1.4) are (n, n), C is (m, m), and X and D_k are (n, m^k).

To see how large (1.4) can be, take the Czech EU calibration of the GEM as an example, Laxton and Pesenti (2003). It has $n = 244$ equations and 49 forward looking variables. After eliminating static variables from (1.4), one gets $m = 88$ predetermined dynamic variables. The following table shows the amount of memory needed for the storage of matrix D_k for several k's.

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

2.2 Preconditioning

The solution to (1.4) is obtained in three steps. First, a suitable linear transformation preconditioning the equation is found, then the transformed equation is solved, and third, an inverse transformation is applied to obtain the solution to the original system.

The preconditioning step consists in premultiplying by A^{-1} to obtain

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

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

$$\begin{aligned} Y + KY\left(\otimes^k F\right) &= \bar{D} && (2.5)\\ Y &= UX\left(\otimes^k V^T\right) && (2.6)\\ \bar{D} &= UA^{-1}D\left(\otimes^k V^T\right). \end{aligned}$$

When (2.2) is solved, the inverse of the linear transformation (2.2) is applied to recover X, i.e.,

$$X = U^T Y\left(\otimes^k V\right). \tag{2.7}$$

Two issues are noteworthy. The first is the existence and stability of inverse A^{-1}. Its existence is implied by the fact that, for the first order, $g_u = -A^{-1}f_u$. Its numerical stability is more complex. Basically, if the matrix A is poorly conditioned, the preconditioning step can introduce severe numerical errors. If this is the case, the model is most likely ill-stated, since the inverse of A is fundamental in solving the linear approximation. However, the condition number gives only the upper bound on the numerical errors. In practice, therefore, our implementation reports the residual relative size of (1.4). The numerical error of $A^{-1}B$ and $A^{-1}D$ need be checked only if the residual relative size is too large.

The second issue regards the Schur decomposition of C. As will become clear, the recursive algorithm benefits from a large number of zero elements in the quasi-triangular matrix F. Additional zero elements can be introduced into F by sacrificing the orthogonality of V, i.e., $C = VFV^{-1}$. However, this departure from orthogonality can introduce large numerical errors, since the condition number of V is no longer equal to 1 as for orthogonal matrices. In order to control condition number growth in the similarity decomposition $c = VFV^{-1}$, the results of Bavely and Stewart (1979) and Dongarra et al. (1992) are used and the algorithm is briefly sketched in *Appendix A*.

2.3 The Recursive Solution

This section gives the core algorithm to solve equation (2.2), which can be vectorized as

$$\left(I + \left(\otimes^k F^T \otimes K\right)\right)\text{vec}(Y) = \text{vec}(\bar{D}). \tag{2.8}$$

Although the algorithm works with the vectorized version, the matrix in (2.8) is not stored in the memory. The basic idea of the recursive algorithm is to obtain the solution at level k with the solutions for the same problem (or similar) at level $k-1$. 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. To describe the recursive approach, suppose that the first eigenvalue of F is real; let us denote this as $r = F_{11}$. If we pick y as the first part of Y (it corresponds to the size of the $F_{[k-1]}$ matrix) and d as the corresponding first part of $\bar{D}$, then y is a solution of

$$(I + r \cdot F_{[k-1]})y = d. \tag{2.9}$$

Note the similarity to (2.8). If the first eigenvalue is complex, I pick the first two parts of Y and the first two parts of $\bar{D}$. The first two parts of Y can then be obtained as a solution of

$$\left(I + \begin{pmatrix} \alpha & \beta_1 \\ -\beta_2 & \alpha \end{pmatrix} \otimes F_{[k-1]}\right) \begin{pmatrix} y_1 \\ y_2 \end{pmatrix} = \begin{pmatrix} d_1 \\ d_2 \end{pmatrix}, \tag{2.10}$$

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

When obtaining the solution of (2.9) or (2.10), I go through all non-zero elements of F^T below the eigenvalue block (since F^T is quasi lower triangular) and eliminate them by updating subsequent parts of matrix $\bar{D}$. More precisely, for the real eigenvalue I 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 I 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 eliminations are performed, subsequent parts of Y can be found as a solution to an equation of type (2.9) (or (2.10) respectively). In this way, the solution Y is obtained.

What remains to be discussed is how the equations for the real (2.9) and complex (2.10) cases are solved. As noted before, (2.9) is a slight generalization of (2.2) ($r = 1$). It is clear that solving (2.9) at level $k-1$ in the same way as the original equation (2.2) at level k will result in cases of both types ((2.9) and (2.10)) at levels $k-2$, $k-3$, ... For $k = 0$, the solution of (2.9) is easy since $I + rK$ is quasi triangular.

Now it suffices to show how (2.10) is solved. As *Lemma 1* in *Appendix B* claims, the solution of (2.10) is obtained as that for two equations of the form

$$\left(I + 2\alpha \cdot F_{[k-1]} + (\alpha^2 + \beta^2) \cdot F^2_{[k-1]}\right) y = d, \tag{2.11}$$

where $F^2_{[k]}$ denotes $\otimes^k \left(F^T\right)^2 \otimes K^2$. Note that the eigenvalues of F^2 are the squares of the 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 a manner very similar to that of equation (2.2). I simply go through all the diagonal blocks of F (coinciding with blocks of F^2), solving the appropriate equation, followed by an elimination.

What are the appropriate equations 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

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

In fact, this is exactly the same as (2.11), so I solve it by recursion. At the bottom of the recursion ($k = 0$), I 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 a 2×2 block of F^T

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

one obtains an equation of the form

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

Lemma 2 in *Appendix B* claims that this type of equation can be transformed to the solution of two systems of two serial equations of the type (2.11). In this way, the recursion is closed. A 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 the input/output vector, the algorithm needs memory only for temporary vectors allocated on the recursion's stack. These are the modified right-hand sides in *Lemma 1* and *Lemma 2* in *Appendix B*. The memory consumption peak is 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 memory consumption, numerical complexity is not that simple. Calculating the number of flops for a solution of the appropriate recursive formulas yields a fairly complicated result. Without making further assumptions, it is not possible to deduce the dominating terms. The following table shows the flops complexity for the extreme assumptions of a fully and moderately filled matrix F. The latter means that there are pm non-zero elements in matrix F, where $p \ll m$. In an implementation, one does not solve for the derivatives of the decision rules with respect to static variables, because they are zero. Therefore they may be excluded. Let n_1 denote the number of forward looking variables, $n_1 \leq n - m$.

Assumption	Flops complexity
no real eigenvalues, full F	$\Theta(m^k n n_1) + \Theta(m^{2k-2} n n_1) + \Theta(m^{k+1} n)$
no real eigenvalues, moderate F	$\Theta(m^k n n_1)$
all eigenvalues real, full F	$\Theta(k m^{k+1} n)$
all eigenvalues real, moderate F	$\Theta(m^k n n_1) + \Theta(k p m^k n)$

The computational time can be divided into three parts. The first portion deals with the solution of bottom quasi triangular systems, the second with the eliminations, and the third with right-hand side modifications of *Lemma 1* and *Lemma 2* of *Appendix B*. When all the eigenvalues are complex, the right-hand side modification term dominates the 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 approximately half of the computational time is[sjv1] consumed in the right-hand side modifications. If there are no complex eigenvalues in F, the time is dominated in most cases by the eliminations.

2.6 Complex Schur Modification

Having studied the computational complexity, a natural question arises. Why we do not precondition equation (1.4) using a complex Schur decomposition giving strictly triangular (though complex) matrices K, and F in (2.2)? Then, all the calculations would be much easier, since the system (2.2) is triangular. There would be no need for the right-hand side modifications, whose complexity dominates the overall complexity for the general case.

However, this "trick" will not improve the overall computational time, because the true complexity lies in preconditioning (2.2) and recovering (2.7). The complexity of these steps is $\Theta(m^{2k}nn_1)$. The time for these steps can be four times longer than that for the same calculations dealing only with 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 a more general equation than (2.2), i.e.,

$$X + SXT = D, \tag{3.12}$$

where S and T are square matrices and T (without loss of generality) is block upper triangular with $m \times m$ square blocks. Note that T is not required to have the form of a Kronecker product as in (2.2). Let T_{ij} denote a block of T, and let X and D be partitioned according to a column partitioning of T. The main idea of this approach is to decompose the equation (3.12) into 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 both equations take the form of (3.12), as the sum in the second equation is known. 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 equation (1.4), which solves for the k-th derivatives of the policy rules g with respect to the state variables y. The derivatives make a multidimensional symmetric matrix, and when unfolded column-wise, 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, which no longer has the form of a Kronecker product. Thus, one obtains

$$A\hat{X} + B\hat{X}C_k = \hat{D}_k \tag{3.13}$$

This system is first preconditioned in a manner similar to that for the recursive algorithm. We obtain a quasi triangular $F_k = V_k C_k V_k^T$, and the equation takes the form

$$\hat{Y} + K\hat{Y}F_k = \hat{E}_k. \tag{3.14}$$

This is solved in the fashion described above.

First, let us compare the memory consumption of the two algorithms. Recall that the memory required for the recursive algorithm is approximately nm^k. To make the following expressions more intuitive, I compare the memory requirements relative to the size of the real algorithm input, which is $n(m+k-1)!/(k!(m-1)!)$.[2] In our framework, k is much smaller than m, so this can be approximated as $nm^k/k!$. The relative memory consumption of the recursive algorithm is then $k!$. The Bartels–Stewart algorithm allocates memory for the full matrix V_k and the quasi triangular matrix F_k. The memory required for these matrices is approximately $\frac{3}{2}(m^k/k!)^2$, and if we divide by the size of the 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 to 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 the number of flops needed for the Bartels–Stewart algorithm, we have to add the flops from the Schur decomposition of C_k and the solution of (3.14). The Schur real decomposition has cubic complexity with respect to the size of the matrix, so we get $O(m^{3k}/(k!)^3)$.

[3] Supposing that F_k is not block diagonalized, we get a number of flops for the Bartels-Stewart algorithm of $\Theta(nn_1m^k/k!) + \Theta(nm^{2k}/(k!)^2)$. Even if we abstract from 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 in (3.13) by A^{-1} in the preconditioning using the generalized Schur decomposition of A

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

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

and B. Another uses the Hessenberg decomposition of $A^{-1}B$ instead of the Schur decomposition in the preconditioning. This method is known as the Hessenberg–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 (2.2) is the doubling algorithm – see Anderson et al. (1996). The doubling algorithm is formulated in a vectorized form here for ease of comparison with the recursive algorithm. The choice of formulation method clearly has no impact on the implementation efficiency.

The doubling algorithm exploits the 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 (2.8) has all its 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.,

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

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

When these ideas are applied in solving (2.8), one gets 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 $M_j y_{j-1}$ becomes smaller than some user given tolerance, which means that subsequent y_j will not change more than the tolerance. In the doubling algorithm, we do not calculate M_j but calculate $F_j = F^{2^j}$ and $K_j = K^{2^j}$ iteratively, and then evaluate $M_j y_{j-1} = (\otimes^k F_j^T \otimes K_j) y_{j-1}$. The implementation of this operation is described in *Appendix C*.

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

It is not difficult to calculate the number of flops for the doubling algorithm. Let m_f denote the number of non-zero elements in F. The number of flops per iteration is $\Theta(nn_1m_f^k)$. If ν denotes the number of iterations, the overall complexity is $\Theta(\nu nn_1m_f^k)$. Recall that if we take the most unfavorable case for the recursive algorithm, where F has all complex eigenvalues and $m_f = m^2/2$, the dominating term in the flops complexity is $\Theta(nn_1m_f^{k-1})$, which is νm_f times better than the doubling algorithm.

The following table shows the computational times (in seconds) for three models. The first is the GEM of Laxton and Pesenti (2003), and the other two are optimal portfolio problems with 12 and 4 assets respectively. The table also shows how many doubling steps were needed to obtain the size of the $M_{j+1}y_j$ contribution less than the tolerance 10^{-30}.

	recursive	doubling
GEM (k=2)	6.2	16.2 (9 steps)
Portfolio 12 (k=3)	6.9	19.7 (12 steps)
Portfolio 4 (k=5)	15.5	26.2 (8 steps)

Since the doubling algorithm employs very efficient matrix–matrix products (see *Appendix C*), its performance is much better than the number of flops derived above. However, it is still worse than the recursive algorithm. The time of the recursive algorithm does not depend on the data, but that of the doubling algorithm does. The closer the maximum eigenvalue size is to 1, the more steps are needed for the doubling algorithm to converge.

One more issue should be brought in. Consider the case where the input/output vector vec($\bar{D}_k$) of (2.8) is so large that it cannot fit into the physical memory. In this situation, the number of vector touches is often more critical for performance than flops. By a vector touch I mean a transport of a data chunk between processor and memory. Obviously, a vector touch can be very costly if all the data cannot fit into the memory.

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

A much more serious (and often overlooked) issue is the roundoff properties of the doubling algorithm. The algorithm employs square powers of matrices applied repeatedly. Recall that the forward componentwise error E of such an operation satisfies $|E| \leq \frac{n\mathbf{u}}{1-n\mathbf{u}}|A|\cdot|A|$, where n is the 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 the large 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 the residual matrix R of equation (1.4). The Czech-EU calibration of the GEM of Laxton and Pesenti (2003) suffers from this problem. The following table shows the relative residual sizes in a few norms of the doubling and recursive algorithms for the second-order simulation.

	recursive	doubling
$\|R\|_1/\|D_k\|_1$	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

A further analysis of this particular case shows that the matrix F is a significant source of roundoff error. The maximum eigenvalue size is 0.9508, not an unusual value for models of this type. However, the sizes of matrices $|F^{2^j}| \cdot |F^{2^j}|$ remain high for $j = 1, \dots, 5$. This indicates high rounding errors in the evaluations of the powers of F. The recursive algorithm does not have this problem, since it evaluates only the second powers of the matrices. This shows that the doubling algorithm is unusable for this problem.

4. Conclusions

A core step toward a higher-order approximation to the solution of the SGDE model is a Sylvester equation. I present a new recursive algorithm for the Sylvester equation that exploits a particular structure appearing in the context of the perturbation method. I have also compared the algorithm with the Bartels–Stewart approach and the doubling algorithm. The new recursive algorithm is seen to be better in terms of computational time, memory usage and numerical stability. It has also been shown that the doubling algorithm can completely fail to provide a satisfactory solution, due to roundoff errors. Extreme caution, therefore, should be exercised before using the doubling algorithm.

A. Block Diagonalization Algorithm

I describe here the algorithm used for block diagonalizing the matrix C. It employs a similarity transformation $C = VFV^{-1}$, where F is a block diagonal matrix with quasi-triangular blocks. The smaller the block sizes, the more zeros there are above the 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 I 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 the large S and by the insufficiently separated R and T. The latter can be improved by a different eigenvalue ordering. The eigenvalue reordering can be done by an orthogonal similarity transformation that does not worsen the condition number of the resulting V.

Intuitively, an eigenvalue reordering, besides separating R and T, also changes the size of S, so it is very difficult to algorithmically predict the size of Q. Because it is not feasible to try all possible orderings, the heuristics due to Bavely and Stewart (1979) is used. I take the first eigenvalue (1×1 block for real, 2×2 for complex) as matrix R, and T as the rest. Then I calculate Q. If the greatest absolute value of Q's elements is less than a user-given threshold, I break the matrix and carry on with T. Otherwise, I select a suitable eigenvalue from T and incorporate it into R by eigenvalue reordering. Another attempt to break the matrix is then made. The eigenvalue suitable to be selected is the one whose size is closest to the average eigenvalue size of matrix R, since it is likely that such an eigenvalue is guilty of 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 my implementation, the error of the similarity transformation $C = VFV^{-1}$ is reported to provide feedback on the threshold to the user.

A comment must be made here regarding eigenvalue swapping. As discussed in Dongarra et al. (1992), swapping ill-conditioned eigenvalues (or two close ones) may not be possible with respect to computer precision. In this case, such eigenvalues are brought into R together, avoiding their unstable swaps. Another difficulty pointed out by Dongarra et al. (1992) is that a swap can turn an ill-conditioned complex eigenvalue into two real ones. In the extreme, swapping two complex eigenvalues can yield four real ones. In such cases, the algorithm proceeds normally, but the implementation must be able to recognize it.

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 for*

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

then it can be obtained as solution of

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

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 for*

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

it can be obtained as

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

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 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

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

The product is a diagonal consisting 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

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

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 I describe the recursive algorithm, solving (2.2) in more technical detail. I define three functions (which recursively call each other) of which $\mathrm{vec}(Y) = \mathbf{solv1}(1, \mathrm{vec}(\widehat{D}), k)$ provides the solution Y.

In the following text, m denotes the dimension of F and n that of K. Let m_c be the number of complex eigenvalue pairs of F, and m_r the number of real eigenvalues; thus $m = m_r + 2m_c$. F_j denotes the j-th diagonal block of F^T, this is a 1×1 or 2×2 matrix, depending on the j-th eigenvalue of F for $j = 1, \dots, m_c + m_r$. For a fixed j, let $\bar{\jmath}$ denote the index of the first column of F_j in F^T. Finally, whenever I 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 the $\bar{\jmath}$-th partition of y (similarly for $d_{\bar{\jmath}}$). If the 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 $\mathbf{solv1}$.

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

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

The function proceeds as follows:

1. If $k = 0$, the equation is solved directly; K is an upper quasi-triangular matrix, so this is easy.
2. If $k > 0$, then go through all diagonal blocks F_j for $j = 1, \dots, m_r + m_c$ and perform
   (a) If $F_j = (f_{\bar{\jmath}\bar{\jmath}}) = (f)$, return $y_j = y_{\bar{\jmath}} = \mathbf{solv1}(rf, d_{\bar{\jmath}}, k-1)$. Then precalculate $z = r \cdot F_{[k-1]} y_j$, and eliminate elements below F_j. That is, for each $\bar{\imath} = \bar{\jmath}+1, \dots, m$, put
   $$d_{\bar{\imath}} = d_{\bar{\imath}} - f_{\bar{\jmath}\bar{\imath}} z.$$
   (b) If $F_j = \begin{pmatrix} \alpha & \beta_1 \\ -\beta_2 & \alpha \end{pmatrix}$, return $y_j = \mathbf{solv2}(r\alpha, r\beta_1, r\beta_2, d_j, k-1)$. Then 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. That is, for each $\bar{\imath} = \bar{\jmath}+2, \dots, m$ 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 proceeds 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 go through all the diagonal blocks F_j, for $j = 1, \ldots, m_r + m_c$, and perform:
   (a) If $F_j = (f_{\overline{jj}}) = (f)$, then the 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 one can have $y_j = y_{\bar{j}} = \mathbf{solv2p}(f\alpha, f^2\beta^2, d_j, k-1)$.
   To eliminate the elements below F_j 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. That is, for all $\bar{\imath} = \bar{\jmath} + 1, \ldots, m$ put

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

   where $g_{\overline{\jmath\imath}}$ denotes the 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 the 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**, one must calculate $\widehat{d}_{\bar{\jmath}}$, and $\widehat{d}_{\bar{\jmath}+1}$, and a_1, b_1, a_2, b_2 to obtain

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

To eliminate elements below F_j, first 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 go through all $\bar{\imath} = \bar{\jmath} + 2, \ldots, m$ and put

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

C. Implementation of Kronecker Product

The performance of the doubling algorithm and partly of the recursive algorithm depends on how the operation $P \cdot (Q_1 \otimes \ldots \otimes Q_k)$ is implemented. This appendix describes the algorithm used in non-vectorized form. Everything that follows can be equivalently written in vectorized form; the only reason for choosing the non-vectorized form is notational convenience.

The implementation depends on the matrix memory storage mode. I assume here a column major storage mode, that is, the matrix is stored column by column in the memory. Moreover, I assume that there is no skip between columns, in other words, the leading dimension is equal to the number of rows. This assumption implies that one can reshape a matrix by stacking subsequent columns into new columns (or breaking a column into new columns) at zero costs.

Using a fundamental property of the Kronecker product, the product operation can be written as

$$P \cdot (Q_1 \otimes \ldots \otimes Q_k) = P \cdot (Q_1 \otimes I) \cdot \ldots \cdot (I \otimes Q_i \otimes I) \cdot \ldots \cdot (I \otimes Q_k),$$

where the identity matrices (all written as I) have different dimensions. One can choose a different order of matrices on the right- hand side, implying different dimensions of the identity matrices.

The algorithm consists of k steps. Each step performs one of the three operations. These have the form: $B \cdot (A \otimes I)$, $B \cdot (I \otimes A \otimes I)$, and $B \cdot (I \otimes A)$. Their implementations are:

1. $B \cdot (A \otimes I)$: Let m be the number of rows of B, (n, p) be the dimensions of A, and q be the dimensions of I. It is easy to see that

$$B \cdot (A \otimes I) = \text{reshape}(\text{reshape}(B, mq, n) \cdot A, m, pq),$$

   where the operator reshape(M, r, s) reshapes the data of matrix M to r rows and s columns. This step is, in fact, one matrix–matrix product.

2. $B \cdot (I \otimes A)$: Let m, n, p, q be the same as in the previous paragraph. If B is partitioned as $B = [B_1 \quad B_2 \quad \dots \quad B_q]$, then the product is

$$B \cdot (I \otimes A) = [B_1 A \quad B_2 A \quad \dots \quad B_q A].$$

   This step corresponds to q matrix–matrix products.

3. $B \cdot (I \otimes A \otimes I)$: Let m, n, p be the same as in the first paragraph, and let q be the dimension of the first identity matrix and r that of the second one. Then, in a similar way as in the previous paragraph, one gets

$$B \cdot (I \otimes A \otimes I) = [B_1 \cdot (A \otimes I) \quad B_2 \cdot (A \otimes I) \quad \dots \quad B_q \cdot (A \otimes I)].$$

   Each of the partition products corresponds to one matrix–matrix product, thus this step corresponds to q matrix–matrix products.

References

ANDERSON, E. W. ET AL. (1996): *Handbook of Computational Economics*, volume 1 of, chapter Mechanics of Forming and Estimating Dynamic Linear Economies. Elsevier Science.

BARTELS, R. H. AND G. W. STEWART (1972): "Solution of the Equation $AX + XB = C$." *Comm. ACM*, 15:820–26.

BAVELY, C. A. AND G. W. STEWART (1979): "An Algorithm for Computing Reducing Subspaces by Block Diagonalization." *SIAM Journal on Numerical Analysis*, 16(2): 359–367.

DONGARRA, J. J., HAMMARLING, S., AND J. H. WILKINSON (1992): "Numerical Considerations in Computing Invariant Subspaces." *SIAM Journal on Matrix Analysis and Applications*, 13(1):145–161.

GOLUB, G. H. AND C. F. V.LOAN (1996): *Matrix Computations*. The Johns Hopkins University Press, Baltimore, Maryland, Third edition.

HIGHAM, N. J. (2002): *Accuracy and Stability of Numerical Algorithms*. SIAM, Second edition.

JIN, H.-H. AND K. JUDD (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 P. PESENTI (2003): "Monetary Rules for Small, Open, Emerging Economies." *NBER Working Paper 9568*. (http://www.nber.org/papers/9568)

CNB WORKING PAPER SERIES

10/2005	Ondřej Kameník:	*Solving SDGE models: A new algorithm for the sylvester equation*
9/2005	Roman Šustek:	*Plant-level nonconvexities and the monetary transmission mechanism*
8/2005	Roman Horváth:	*Exchange rate variability, pressures and optimum currency area criteria: Implications for the central and eastern european countries*
7/2005	Balázs Égert: Luboš Komárek	*Foreign exchange interventions and interest rate policy in the Czech Republic: Hand in glove?*
6/2005	Anca Podpiera: Jiří Podpiera	*Deteriorating cost efficiency in commercial banks signals an increasing risk of failure*
5/2005	Luboš Komárek: Martin Melecký	*The behavioural equilibrium exchange rate of the Czech koruna*
4/2005	Kateřina Arnoštová: Jaromír Hurník	*The monetary transmission mechanism in the Czech Republic (evidence from VAR analysis)*
3/2005	Vladimír Benáček: Jiří Podpiera Ladislav Prokop	*Determining factors of Czech foreign trade: A cross-section time series perspective*
2/2005	Kamil Galuščák: Daniel Münich	*Structural and cyclical unemployment: What can we derive from the matching function?*
1/2005	Ivan Babouček: Martin Jančar	*Effects of macroeconomic shocks to the quality of the aggregate loan portfolio*

10/2004	Aleš Bulíř: Kateřina Šmídková	*Exchange rates in the new EU accession countries: What have we learned from the forerunners*
9/2004	Martin Cincibuch: Jiří Podpiera	*Beyond Balassa-Samuelson: Real appreciation in tradables in transition countries*
8/2004	Jaromír Beneš: David Vávra	*Eigenvalue decomposition of time series with application to the Czech business cycle*
7/2004	Vladislav Flek, ed.:	*Anatomy of the Czech labour market: From over-employment to under-employment in ten years?*
6/2004	Narcisa Kadlčáková: Joerg Keplinger	*Credit risk and bank lending in the Czech Republic*
5/2004	Petr Král:	*Identification and measurement of relationships concerning inflow of FDI: The case of the Czech Republic*
4/2004	Jiří Podpiera:	*Consumers, consumer prices and the Czech business cycle identification*
3/2004	Anca Pruteanu:	*The role of banks in the Czech monetary policy transmission mechanism*
2/2004	Ian Babetskii:	*EU enlargement and endogeneity of some OCA criteria: Evidence from the CEECs*
1/2004	Alexis Derviz: Jiří Podpiera	*Predicting bank CAMELS and S&P ratings: The case of the Czech Republic*

12/2003	Tibor Hlédik:	*Modelling the second-round effects of supply-side shocks on inflation*
11/2003	Luboš Komárek: Zdeněk Čech Roman Horváth	*ERM II membership – the view of the accession countries*
10/2003	Luboš Komárek: Zdeněk Čech Roman Horváth	*Optimum currency area indices – how close is the Czech Republic to the eurozone?*
9/2003	Alexis Derviz: Narcisa Kadlčáková Lucie Kobzová	*Credit risk, systemic uncertainties and economic capital requirements for an artificial bank loan portfolio*
8/2003	Tomáš Holub: Martin Čihák	*Price convergence: What can the Balassa–Samuelson model tell us?*
7/2003	Vladimír Bezděk: Kamil Dybczak Aleš Krejdl	*Czech fiscal policy: Introductory analysis*
6/2003	Alexis Derviz:	*FOREX microstructure, invisible price determinants, and the central bank's understanding of exchange rate formation*
5/2003	Aleš Bulíř:	*Some exchange rates are more stable than others: Short-run evidence from transition countries*
4/2003	Alexis Derviz:	*Components of the Czech koruna risk premium in a multiple-dealer FX market*
3/2003	Vladimír Benáček: Ladislav Prokop Jan Á. Víšek	*Determining factors of the Czech foreign trade balance: Structural Issues in Trade Creation*
2/2003	Martin Čihák: Tomáš Holub	*Price convergence to the EU: What do the 1999 ICP data tell us?*
1/2003	Kamil Galuščák: Daniel Münich	*Microfoundations of the wage inflation in the Czech Republic*
4/2002	Vladislav Flek: Lenka Marková Jiří Podpiera	*Sectoral productivity and real exchange rate appreciation: Much ado about nothing?*
3/2002	Kateřina Šmídková: Ray Barrell Dawn Holland	*Estimates of fundamental real exchange rates for the five EU pre-accession countries*
2/2002	Martin Hlušek:	*Estimating market probabilities of future interest rate changes*
1/2002	Viktor Kotlán:	*Monetary policy and the term spread in a macro model of a small open economy*

CNB RESEARCH AND POLICY NOTES

4/2005	Vít Bárta:	*Fulfilment of the Maastricht inflation criterion by the Czech Republic: Potential costs and policy options*
3/2005	Helena Sůvová: Eva Kozelková David Zeman Jaroslava Bauerová	*Eligibility of external credit assessment institutions*
2/2005	Martin Čihák: Jaroslav Heřmánek	*Stress testing the Czech banking system: Where are we? Where are we going?*
1/2005	David Navrátil: Viktor Kotlán	*The CNB's policy decisions – Are they priced in by the markets?*
4/2004	Aleš Bulíř:	*External and fiscal sustainability of the Czech economy: A quick look through the IMF's night-vision goggles*
3/2004	Martin Čihák:	*Designing stress tests for the Czech banking system*
2/2004	Martin Čihák:	*Stress testing: A review of key concepts*
1/2004	Tomáš Holub:	*Foreign exchange interventions under inflation targeting: The Czech experience*
2/2003	Kateřina Šmídková:	*Targeting inflation under uncertainty: Policy makers' perspective*
1/2003	Michal Skořepa: Viktor Kotlán	*Inflation targeting: To forecast or to simulate?*

CNB ECONOMIC RESEARCH BULLETIN

November 2005	*Financial stability*
May 2005	*Potential output*
October 2004	*Fiscal issues*
May 2004	*Inflation targeting*
December 2003	*Equilibrium exchange rate*

Czech National Bank
Economic Research Department
Na Příkopě 28, 115 03 Praha 1
Czech Republic
phone: +420 2 244 12 321
fax: +420 2 244 14 278
http://www.cnb.cz
e-mail: research@cnb.cz