Applied Mathematics and Computation 435 (2022) 127458

Contents lists available at ScienceDirect

Applied Mathematics and Computation

journal homepage: www.elsevier.com/locate/amc

The dynamical functional particle method for multi-term linear matrix equations

Andrii Dmytryshyn[a], Massimiliano Fasi[b,*], Mårten Gulliksson[a]

[a] School of Science and Technology, Örebro University, Örebro, 701 82, Sweden
[b] Department of Computer Science, Durham University, Durham DH1 3LE, UK

ARTICLE INFO

Article history:
Received 1 March 2021
Revised 28 July 2022
Accepted 30 July 2022
Available online 14 August 2022

2010 MSC:
65F30
15A24

Keywords:
Linear matrix equation
Discrete functional particle method
Lyapunov equation
Sylvester equation
Generalized Sylvester equation

ABSTRACT

Recent years have seen a renewal of interest in multi-term linear matrix equations, as these have come to play a role in a number of important applications. Here, we consider the solution of such equations by means of the dynamical functional particle method, an iterative technique that relies on the numerical integration of a damped second order dynamical system. We develop a new algorithm for the solution of a large class of these equations, a class that includes, among others, all linear matrix equations with Hermitian positive definite or negative definite coefficients. In numerical experiments, our MATLAB implementation outperforms existing methods for the solution of multi-term Sylvester equations. For the Sylvester equation $AX + XB = C$, in particular, it can be faster and more accurate than the built-in implementation of the Bartels–Stewart algorithm, when A and B are well conditioned and have very different size.

© 2022 The Author(s). Published by Elsevier Inc.
This is an open access article under the CC BY license
(http://creativecommons.org/licenses/by/4.0/)

1. Introduction

The matrix equation $AX + XB = C$, where $A \in \mathbb{C}^{m\times m}$, $B \in \mathbb{C}^{n\times n}$, and $X, C \in \mathbb{C}^{m\times n}$, is named after Joseph J. Sylvester, who was the first to consider the homogeneous case [1]. Many similar equations appear in various branches of science and have been extensively studied. Here, we are interested in numerical algorithms for the solution of the linear matrix equation

$$\sum_{i=1}^{\ell} A_i X B_i = C, \qquad A_1, \dots, A_\ell \in \mathbb{C}^{m\times m}, \quad B_1, \dots, B_\ell \in \mathbb{C}^{n\times n}, \quad X, C \in \mathbb{C}^{m\times n}, \tag{1}$$

which is sometimes called the “generalized” Sylvester matrix equation [2], [3], although not all authors agree on this name.

In general, it is difficult to characterize the existence and uniqueness of solutions to (1) in terms of the coefficients A_i and B_i, but well-known results are available for special cases [4, Chap. 9, 10, and 11]. As suggested by Lancaster [5], by using the Kronecker product one can recast (1) as the $mn \times mn$ linear system

$$M\,\mathrm{vec}(X) = \mathrm{vec}(C), \qquad M := \sum_{i=1}^{\ell} \left(B_i^T \otimes A_i\right), \tag{2}$$

* Corresponding author.
E-mail addresses: andrii.dmytryshyn@oru.se (A. Dmytryshyn), massimiliano.fasi@durham.ac.uk (M. Fasi), marten.gulliksson@oru.se (M. Gulliksson).

https://doi.org/10.1016/j.amc.2022.127458
0096-3003/© 2022 The Author(s). Published by Elsevier Inc. This is an open access article under the CC BY license (http://creativecommons.org/licenses/by/4.0/)

where $\cdot \otimes \cdot$ and vec denote the Kronecker product and the operator that stacks the columns of a matrix into a column vector, respectively. In the following, we denote by $\mathrm{unvec}_{m,n}$ the operator that reshapes a vector of length mn into an $m \times n$ matrix so that $\mathrm{unvec}_{m,n}(\mathrm{vec}(X)) = X$ for $X \in \mathbb{C}^{m\times n}$.

The linear system (2) has a unique solution if and only if $\det M \neq 0$, that is, if the coefficient matrix M has full rank. When $\det M = 0$, on the other hand, the matrix equation (1) has either infinitely many solutions, if M and the augmented matrix $[M\,\mathrm{vec}(C)] \in \mathbb{C}^{mn\times(mn+1)}$ have same rank, or no solution, if they do not.

Some special cases of (1) are particularly important and have been extensively studied in the literature. The most obvious example is that of a linear system with multiple right-hand sides. Here we will discuss, in particular, the following:

- the two-sided linear equation [6]
$$AXB = C, \tag{3}$$
- the generalized inverse [6]
$$AXA = A, \tag{4}$$
- the continuous-time Lyapunov equation [7], [8]
$$AX + XA^* = C, \tag{5}$$
- the discrete-time Lyapunov equation [8], [9]
$$AXA^* - X = C, \tag{6}$$
- the Sylvester equation [1], [10]
$$AX + XB = C, \tag{7}$$
- the discrete-time Sylvester equation [9], [11]
$$AXB + X = C, \tag{8}$$
- a number of other linear matrix equations—or systems thereof—for which the umbrella expression “generalized Sylvester equations” is sometimes used [12, App. A].

A couple of decades ago, the matrix equation (1) was considered to be mainly of theoretical interest [13, sect. 16.5]. In recent years, however, it has come to play an important role in a variety of applications [9] such as, for example, the numerical study of certain bilinear dynamical systems [7], [14], [15, sect. 2.2], or uncertainty quantification in PDEs with random inputs [16], [17], [18]. In many of these applications, the matrix coefficients show a special structure, and, in particular, are often Hermitian and positive definite or semi-definite.

Here we adapt the dynamical functional particle method [19] to the numerical solution of (1). The algorithm we develop requires the eigenvalues of the coefficient matrix M in (2) be real and have same sign. The class of equations that satisfy this requirements includes, among others, all matrix equations of the form (4) with Hermitian positive or negative definite coefficients. The general technique we develop can be tailored to the special cases (3)–(8) so to reduce computational cost and, therefore, execution time.

In its most elementary form, the algorithm we propose involves only matrix multiplications, and a more refined variant requires also the solution of linear systems. This simplicity translates into ease of implementation: matrix multiplication and solution of linear systems are among the most basic matrix operations, and are therefore available in most libraries for linear algebra. This makes the new approach particularly suitable for computational environments in which only a few linear algebra kernels are typically available. The technique developed by Bartels and Stewart for the solution of (5) and (7), for example, requires the Schur factorization of the coefficients of the matrix equation [10]. This factorization is computed by means of Francis’s double-shift QR algorithm [20], also known as QR algorithm [21], [22], which is one of the most complex algorithms in matrix computation [23, Chap. 7] and is often not available, for example, in massively parallel and in multiprecision or mixed-precision computing environments [24]. The new algorithms can readily be implemented in such frameworks, as multiprecision libraries typically feature routines for computing matrix products and solving linear systems.

Moreover, the iterative nature of the new methods allows for a finer control over the accuracy of the computed solution: the user can choose to stop the iteration as soon as the target precision has been reached, which is not possible when using the Bartels–Stewart algorithm. Ours is not the first iterative technique for solving matrix equations of the form (1). Ding and Chen have developed several gradient-based algorithms [25] for the solution of such equations, and these algorithms can be adapted to the case of coupled Sylvester equations [26]. The alternating-directional-implicit (ADI) iterations [27] can also be used to solve Sylvester equations.

The next section summarizes the main features of the dynamical functional particle method, upon which our technique for the solution of (1) is built. In view of the equivalent formulation (2), particular emphasis is given to the theoretical aspects specific to the solution of linear systems. In Section 3, we describe how the method can be adapted to the solution of matrix equations of the form (1) and discuss how the algorithm can be tailored to the solution of (3)–(8). In Section 4, we compare our implementation of the new algorithms with the built-in MATLAB function `sylvester` and with an implementation of the gradient-based iterative algorithms of Ding and Chen [25], [26]. We conclude the paper by drawing our conclusions and outlining possible directions for future work.

2. The dynamical functional particle method

Edvardsson, Gulliksson, and Persson originally introduced the dynamical functional particle method as a technique for the solution of boundary value problems arising in quantum chemistry [19], but the use of this technique has since been considered for a number of applications including, for example, constrained optimization and linear eigenvalue problems [28], or linear least squares and ill-posed linear systems [29].

Our iterative method for the solution of (1) builds upon the corresponding method for the solution of linear systems of equations [30], which we now briefly recall. Let $G \in \mathbb{C}^{n\times n}$ be a matrix with positive real eigenvalues, let $b \in \mathbb{C}^n$, and let $x : \mathbb{R}^+ \to \mathbb{C}^n$, where $\mathbb{R}^+$ denotes the closed positive real axis, be a function of the dummy time variable t. The method described in this section is equally applicable if G has only negative real eigenvalues, in which case it suffices to consider the matrix $-G$.

In order to solve the linear system

$$Gy = b, \tag{9}$$

consider the second order dynamical system

$$\ddot{x}(t) + \mu\,\dot{x}(t) = b - Gx(t), \qquad \mu > 0, \tag{10}$$

which can equivalently be written as the first order system

$$\begin{cases} \dot{x}(t) = v(t), \\ \dot{v}(t) = -\mu v(t) + b - Gx(t). \end{cases} \tag{11}$$

The system (11) can be integrated efficiently in time using the symplectic Euler algorithm [31, Chap. VI]. For $\Delta t > 0$, we obtain

$$\begin{cases} x_{k+1} = x_k + \Delta t\, v_{k+1}, \\ v_{k+1} = v_k - \Delta t\, \mu\, v_k + \Delta t\,(b - Gx_k), \end{cases} \tag{12}$$

which can be rewritten in matrix form as

$$w_{k+1} = Bw_k + c, \qquad B = \begin{bmatrix} I - \Delta t^2 G & \Delta t\,(1-\Delta t\,\mu)\,I \\ -\Delta t\, G & (1-\Delta t\,\mu)\,I \end{bmatrix}, \qquad c = \begin{bmatrix} \Delta t^2\, b \\ \Delta t\, b \end{bmatrix}, \tag{13}$$

where $w_k = [x_k\ v_k]^T$. For the initial conditions, $x(0)$ is initialized to a random vector and $\dot{x}(0)$ is set to 0 for simplicity. It is easy to see that if (12) converges, then the vector $\widetilde{y} := x(t)$ tends to a solution to (9) as $t \to 0$.

The convergence of the symplectic scheme (12) is governed by the damping coefficient $\mu > 0$, which appears in the dissipation term, and by the discretization step Δt. In order to ensure fast convergence of the method, one must choose Δt and μ so that $\|B\|_2$ is smaller than 1 and, in fact, as small as possible. The optimal choice of these two parameters, see [30], is

$$\mu^* = \frac{2\sqrt{\lambda_{\min}(G)\lambda_{\max}(G)}}{\sqrt{\lambda_{\min}(G)} + \sqrt{\lambda_{\max}(G)}}, \qquad \Delta t^* = \frac{2}{\sqrt{\lambda_{\min}(G)} + \sqrt{\lambda_{\max}(G)}}, \tag{14}$$

where $\lambda_{\min}(G)$ and $\lambda_{\max}(G)$ denote the smallest and largest eigenvalue of the matrix G, respectively. The parameters in (14) are the solution to the optimization problem

$$\min_{\Delta t, \mu}\ \max_{1\le i\le 2n} |\lambda_i(B)|,$$

where $\lambda_1(B) \ge \lambda_2(B) \ge \cdots \ge \lambda_{2n}(B)$ are the eigenvalues of the matrix B in (13). In [29, Cor. 1], this method using optimal parameters is shown to be stable and convergent for positive definite (and, therefore, negative definite) matrices.

If we substitute the values in (14) into B in (13) and assume, without loss of generality, that $b = 0$, then we can show that

$$\|w_{k+1}\|_2 \le \|B\|_2 \|w_k\|_2 \le \frac{\sqrt{\kappa_2(G)} - 1}{\sqrt{\kappa_2(G)} + 1}, \tag{15}$$

where $\kappa_2(G) = \|G\|_2 \, \|G^{-1}\|_2$ is the 2-norm condition number of G. We refer the interested reader to the proof of [29, Cor. 1] for the details. We remark that, in the worst case, this technique has the same convergence rate as the conjugate gradient method.

We note that other approaches based on dynamical systems typically aim to minimize the potential

$$V(t) = \frac{x(t)^T G x(t)}{2} - x(t)^T b, \tag{16}$$

and we refer the reader to [29] and references therein for the details. By using (11), on the other hand, the dynamical functional particle method instead aims to decrease the energy

$$E(t) = \frac{x(t)^T G x(t)}{2} - x(t)^T b + \frac{v(t)^T v(t)}{2},$$

which complements the potential (16) with an additional kinetic term. Then by using symplectic methods, which are particularly efficient for systems of the form (11), we can outperform existing methods, as shown in Section 4.

3. Solving the multi-term linear equation

The dynamical functional particle method for the solution of the matrix equation (1) can readily be formulated by using the alternative expression (2). For efficiency's sake, the coefficient matrix M is never explicitly computed, and the matrix–vector product Gx in (12) is performed implicitly, as we now explain.

Let $\widetilde{X}_0 \in \mathbb{C}^{mn}$ be a vector with randomly generated entries, and let $\widetilde{V}_0 \in \mathbb{C}^{mn}$ be a vector of length mn with all entries set to 0. We can apply the symplectic Euler scheme in (12) to (2) and write

$$\begin{cases} \widetilde{R}_k = \operatorname{vec}(C) - M \operatorname{vec}\left(\widetilde{X}_k\right), \\ \widetilde{V}_{k+1} = \widetilde{V}_k + \Delta t \cdot \left(\widetilde{R}_k - \mu \widetilde{V}_k\right), \\ \widetilde{X}_{k+1} = \widetilde{X}_k + \Delta t \cdot \widetilde{V}_{k+1}, \end{cases} \tag{17}$$

where the matrix M is the sum of Kronecker products defined in (2). The approximate solution to (1) at step k' will be $\operatorname{unvec}_{m,n}(\widetilde{X}_{k'})$.

The iterative scheme (17) is not practical, as each step requires the evaluation of the matrix–vector product $M \operatorname{vec}(X)$. As M is a matrix of order mn, this matrix computation requires $2m^2n^2 + o(m^2n^2)$ floating-point operations (flops), and can become unduly expensive even for moderate values of m and n. The computation of the residual R_k, however, can equivalently be written as

$$\widetilde{R}_k = \operatorname{vec}\left(C - \sum_{i=1}^{\ell} A_i X_k B_i\right), \tag{18}$$

so that one step of the iteration requires only $2\ell(m^2n + mn^2) + o\big(\ell(m^2n + mn^2)\big)$ flops.

In view of this observation, we can rewrite (17) in the more natural form

$$\begin{cases} R_k = C - \sum_{i=1}^{\ell} A_i X_k B_i, \\ V_{k+1} = V_k + \Delta t \cdot \left(R_k - \mu V_k\right), \\ X_{k+1} = X_k + \Delta t \cdot V_{k+1}, \end{cases} \tag{19}$$

where $X_k, V_k, R_k \in \mathbb{C}^{m \times n}$.

In order to obtain the optimal damping and time step for this scheme, we still need an efficient way of estimating $\lambda_{\max}(M)$ and $\lambda_{\min}(M)$, which are real by assumption. This can be done efficiently by using methods such as the subspace iteration or polynomial Krylov methods [32]. These iterative algorithms approximate an eigenvector corresponding to the dominant eigenvalue by performing, at each iteration, only matrix–vector products, which can be implemented implicitly without ever forming the matrix M. More refined techniques, such as the inverse iteration or methods based the extended Krylov subspace [33], can be used when an efficient routine for solving linear systems having M as coefficient is available.

To the best of our knowledge, no such routine exists when $\ell > 2$, but the extreme eigenvalues of M can be computed efficiently for some special cases. For the one-term and two-term linear equations in (3)–(8) we can use the following well-know result.

Proposition 3.1 ([34], Thm. 4.2.12 and Ex. 19, p. 251)**.** *Let $\lambda_1,\ldots, \lambda_m \in \mathbb{C}$ be the eigenvalues of $A \in \mathbb{C}^{m\times m}$ and let $\xi_1, \ldots, \xi_n \in \mathbb{C}$ be the eigenvalues of $B \in \mathbb{C}^{n\times n}$. Then for $i = 1,\ldots, m$ and $j = 1,\ldots, n$, the mn eigenvalues of $A \otimes B$ and $A \otimes I_n + I_m \otimes B$ have the form $\lambda_i \xi_j$ and $\lambda_i + \xi_j$, respectively.*

By combining this result with the fact that a matrix and its transpose have the same characteristic polynomial and thus the same eigenvalues, we can obtain formulae for the extreme eigenvalues of M which only require knowledge of the extreme eigenvalues of the coefficient matrices appearing on the left-hand side of the matrix equation. We summarize the results for the cases of interest in Table 1.

Table 1
Spectrum of the Kronecker form of some special cases of (1).

Matrix equation	Coefficient M	$\lambda_{\min}(M)$	$\lambda_{\max}(M)$
$AXB = C$	$B^T \otimes A$	$\lambda_{\min}(A)\lambda_{\min}(B)$	$\lambda_{\max}(A)\lambda_{\max}(B)$
$AXA = C$	$A^T \otimes A$	$\lambda_{\min}(A)^2$	$\lambda_{\max}(A)^2$
$AX + XA^* = C$	$I_m \otimes A + \bar{A} \otimes I_m$	$2\lambda_{\min}(A)$	$2\lambda_{\max}(A)$
$AXA^* - X = C$	$\bar{A} \otimes A - I_m \otimes I_m$	$\lambda_{\min}(A)^2 - 1$	$\lambda_{\max}(A)^2 - 1$
$AX + XB = C$	$I_m \otimes A + B^T \otimes I_m$	$\lambda_{\min}(A) + \lambda_{\min}(B)$	$\lambda_{\max}(A) + \lambda_{\max}(B)$
$AXB + X = C$	$B^T \otimes A + I_m \otimes I_m$	$\lambda_{\min}(A)\lambda_{\min}(B) + 1$	$\lambda_{\max}(A)\lambda_{\max}(B) + 1$

We note that the approximations

$$\lambda_{\min}(M) \approx \sum_{i=1}^{\ell} \lambda_{\min}(B_i^T \otimes A_i), \qquad \lambda_{\max}(M) \approx \sum_{i=1}^{\ell} \lambda_{\max}(B_i^T \otimes A_i), \tag{20}$$

do not require the explicit computation of the matrix M.

The method discussed in this section inherits the convergence and stability behavior of the algorithm for linear systems described in Section 2. The following result is a direct consequence of [29, Cor. 1].

Theorem 3.1 (Convergence)**.** *Let $X_0, V_0 \in \mathbb{C}^{m\times n}$ be a matrix with randomly generated entries and the zero matrix, respectively, let $X_* \in \mathbb{C}^{m\times n}$ be the matrix that satisfies (1), and let M be the coefficient matrix of the linear system (2). Then the scheme (19) with*

$$\mu = \frac{2\sqrt{\lambda_{\min}(M)\lambda_{\max}(M)}}{\sqrt{\lambda_{\min}(M)} + \sqrt{\lambda_{\max}(M)}}, \qquad \Delta t = \frac{2}{\sqrt{\lambda_{\min}(M)} + \sqrt{\lambda_{\max}(M)}},$$

is stable and

$$\lim_{n\to\infty} \frac{\|X_{n+1} - X_*\|_2}{\|X_n - X_*\|_2} = \frac{\sqrt{\kappa_2(M)} - 1}{\sqrt{\kappa_2(M)} + 1} =: \eta. \tag{21}$$

Therefore, the convergence of the dynamical functional particle method for the matrix equation (1) is linear, and the convergence rate η in (21) depends only on the conditioning of the matrix M in (2): the convergence will be fastest when $\kappa_2(M) \approx 1$, in which case η is proportional to $\kappa_2(M) - 1$, and will slow down as $\kappa_2(M)$ grows, since $\eta \to 1$ as $\kappa_2(M) \to \infty$.

4. Numerical experiments

Now we compare the algorithm discussed in Section 3 with existing methods for the solution of the matrix equation (1) and of special cases thereof. The results in this section were obtained by running the experiments in MATLAB 9.11.0 (R2021b) Update 1, on a machine equipped with 32 GiB of RAM and an AMD Ryzen 7 PRO 5850U running at 1.9 GHz. In order to facilitate the reusability of our work and the replicability of our results [35], the code used to produce the plots in this section is available on GitHub.[1] We compared the following five implementations.

- `solve_gen_kron` solves the linear system (2) by explicitly constructing the matrix M and then using the MATLAB backslash operator.
- `solve_gen_dfpm_opt` solves (1) by using the dynamical functional particle method described in Section 3 with optimal damping and time step computed according to (14) for $G = M$. The quantities $\lambda_{\min}(M)$ and $\lambda_{\max}(M)$ are computed by constructing the matrix M explicitly, with the exception of the special cases in Table 1.
- `solve_gen_dfpm_app` solves (1) by using the dynamical functional particle method described in Section 3 with parameters chosen according to (14) for $G = M$. The extreme eigenvalues of M are estimated as in (20).
- `solve_gen_gbia` solves (1) by using the gradient based iterative algorithms developed by Ding and Chen [25], [26].
- `sylvester` solves the continuous-time Lyapunov equation (5) and the Sylvester equation (7) by means of the MATLAB function `sylvester`, which implements the algorithm of Bartels and Stewart [10].

For the iterative algorithms, we set the maximum number of iterations to 50,000 and keep iterating until the 1-norm relative residual, defined for an equation of the form (1) as

$$\frac{\left\| \sum_{i=1}^{\ell} A_i X_k B_i - C \right\|_1}{\left(\sum_{i=1}^{\ell} \|A_i\|_1 \|B_i\|_1\right) \|X_k\|_1 + \|C\|_1},$$

[1] https://github.com/mfasi/dfpm-gen-sylvester

gets below $2^3 \cdot u$, where $u = 2^{-53} \approx 1.11 \times 10^{-16}$ denotes the unit roundoff of binary64 arithmetic [36], known as "double precision" arithmetic in previous revisions of the IEEE 754 standard [37], [38]. The extreme eigenvalues of M are estimated by means of the MATLAB function `eigs`.

In our tests, the superscript notation $A^{(\eta)} \in \mathbb{R}^{m\times m}$ denotes the real non-symmetric matrix generated as

$$A^{(\eta)} = PD^{(\eta)}P^{-1}, \tag{22}$$

where $P \in \mathbb{R}^{m\times m}$ is such that $\kappa_2(P) = 2$ and is generated using the `randsvdfast` function [39], whereas $D^{(\eta)}$ is a diagonal matrix with extreme eigenvalues $\sqrt{\eta}^{-1}$ and $\sqrt{\eta}$ and remaining diagonal elements uniformly distributed in $\left[\sqrt{\eta}^{-1}, \sqrt{\eta}\right]$. This choice of P and $D^{(\eta)}$ ensures that $\kappa_2\left(A^{(\eta)}\right) \le 4\eta$, and in practice provides a matrix $A^{(\eta)}$ such that $\kappa_2\left(A^{(\eta)}\right) \approx \eta$. We stress that the matrix $A^{(\eta)}$ is diagonalizable by construction; this fact is exploited only by the `sylvester` algorithm, but does not make a difference for any of the iterative algorithms we consider in these experiments.

4.1. The Sylvester equation

In this first experiment we consider the solution of the Sylvester equation

$$A^{(\eta)}X - XB^{(\eta)} = C, \tag{23}$$

where $A^{(\eta)} \in \mathbb{R}^{m\times m}$ and $B^{(\eta)} \in \mathbb{R}^{n\times n}$ are as in (22) and $C \in \mathbb{R}^{m\times n}$ is generated using a matrix $X \in \mathbb{R}^{m\times n}$ with entries from the Gaussian distribution.

In Figure 1 we report the time required by `sylvester` and `solve_gen_dfpm_opt` to solve (23) and the forward error of the solutions computed by the two algorithms. In the plots we consider two moderate values of η, namely 10 (top row) and 100 (bottom row), and we fix $n = 500$ while allowing m to vary between 2 and 500. Especially for smaller values of m, this choice leads to equations with one coefficient much smaller than the other. In this setting, it would be appropriate to consider the Schur–Hessenberg algorithm of Golub, Nash, and Van Loan [40], which requires the computation of the Schur form of only one of the matrix coefficients. We did not include this method in our comparison since, as far as we are aware, no LAPACK-style implementation of this algorithm exists, and a naive MATLAB implementation would not offer a fair comparison.

For both values of η, `solve_gen_dfpm_opt` is more accurate than `sylvester` for all test matrices. The forward error of `solve_gen_dfpm_opt` is of the order of $\kappa_1(M)u$ for $\eta = 10$ and about one order of magnitude smaller than the accuracy reference for $\eta = 100$.

We now discuss the timings of the two algorithms. As `solve_gen_dfpm_opt` is an iterative algorithm, its execution time depends not only on the computational cost of a single iteration, but also on the total number of steps needed to achieve convergence.

As discussed above, the most expensive operation in `solve_gen_dfpm_opt` is the computation of the residual, thus each iteration asymptotically requires $2(m^2n + mn^2)$ flops. Therefore, we should expect the timings of this algorithm to grow with the order of $A^{(\eta)}$ and $B^{(\eta)}$, which is confirmed by the plots in the left column of Figure 1.

The total number of iterations required by `solve_gen_dfpm_opt` is proportional to the conditioning of the matrix M in (2), which in turn roughly depends on the parameter η. This is consistent with previous findings in the literature on the dynamical functional particle method for linear systems with positive definite coefficients. In our experiments, the algorithm required between 52 and 63 iterations for the matrices in the top row and between 195 and 887 for those in the bottom row.

In our experimental setup, for $\eta = 10$ `solve_gen_dfpm_opt` is faster than `sylvester` when the order of $A^{(\eta)}$ is up to 40% of that of $B^{(\eta)}$, a percentage that reduces to about 10% when $\eta = 100$. For larger values of η, `solve_gen_dfpm_opt` is typically slower but still more accurate than `sylvester`. Similar results obtained for larger values of n suggest that `solve_gen_dfpm_opt` is marginally—but consistently—more accurate than `sylvester`, but is faster than the latter only when the coefficients of the matrix equation (23) are well conditioned and differ considerably in size.

4.2. The multi-term linear equation

Now we consider the solution of the more general matrix equation

$$\sum_{i=1}^{5} A_i^{(\eta)} X B_i^{(\eta)} = C. \tag{24}$$

In our experiments, the $m \times m$ matrices $A_1^{(\eta)},\dots, A_5^{(\eta)}$ are simultaneously diagonalizable, and so are the $m \times m$ matrices $B_1^{(\eta)},\dots, B_5^{(\eta)}$, although the eigenvectors of $A_i^{(\eta)}$ and $B_i^{(\eta)}$ are different, in general. This is to ensure that the matrix M in (2) has positive real eigenvalues. As in the previous experiment, the matrix $C \in \mathbb{R}^{m\times m}$ is computed by using a matrix $X \in \mathbb{R}^{m\times m}$ with entries from the Gaussian distribution.

In Figure 2 we compare `solve_gen_kron`, `solve_gen_dfpm_opt`, `solve_gen_dfpm_app`, and `solve_gen_gbia` for the solution of (24) as m varies. As we are mainly interested in well-conditioned matrices, we consider once again the two cases $\eta = 10$ and $\eta = 100$.

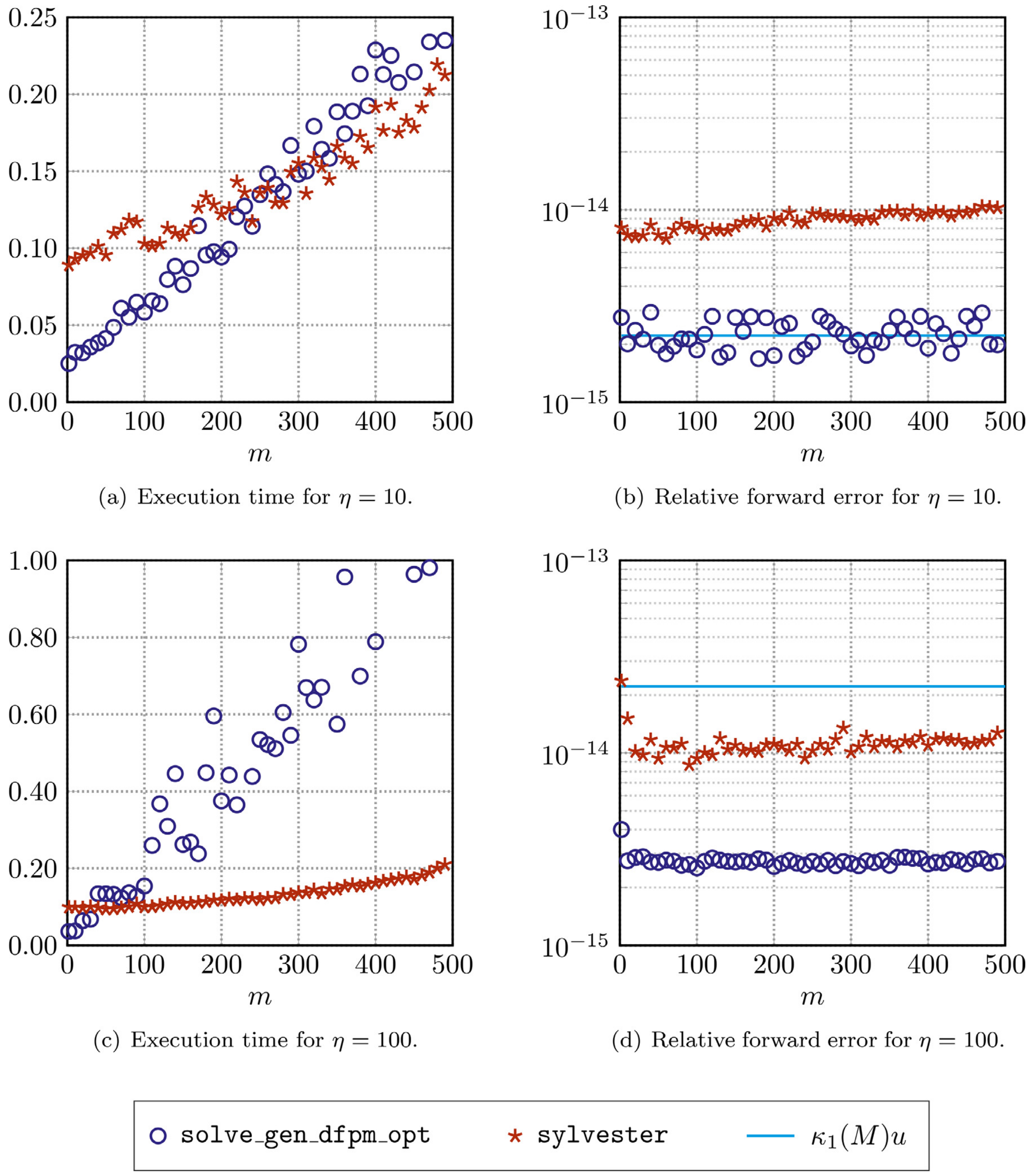

Fig. 1. Left: execution time, in seconds, required by `solve_gen_dfpm_opt` and `sylvester` to solve the matrix equation in (23) for $n = 500$. Right: relative forward error of the computed solution.

Our results suggest that `solve_gen_kron` is the most accurate of the four algorithms, and is the only one that achieves a forward error of the magnitude of $\kappa_1(M)u$. The two implementations based on the dynamical functional particle method achieve a similar level of accuracy. Finally, `solve_gen_gbia` is always the least accurate of the algorithms we test and for four of our test matrices it fails to satisfy the stopping criterion within 50,000 iterations.

We now compare the four algorithms in terms of performance. For matrix equations of small size, `solve_gen_kron` is typically the most efficient algorithm, followed by `solve_gen_dfpm_opt`, `solve_gen_dfpm_app`, and finally `solve_gen_gbia`. For these small matrices, the approximation of the extreme eigenvalues of M in (2) is inexpensive, and the cost of the three iterative algorithms depends mostly on the number of iterations that are necessary to achieve convergence. As `solve_gen_dfpm_opt` requires considerably fewer iterations than the other two methods, it is the fastest for m below 40.

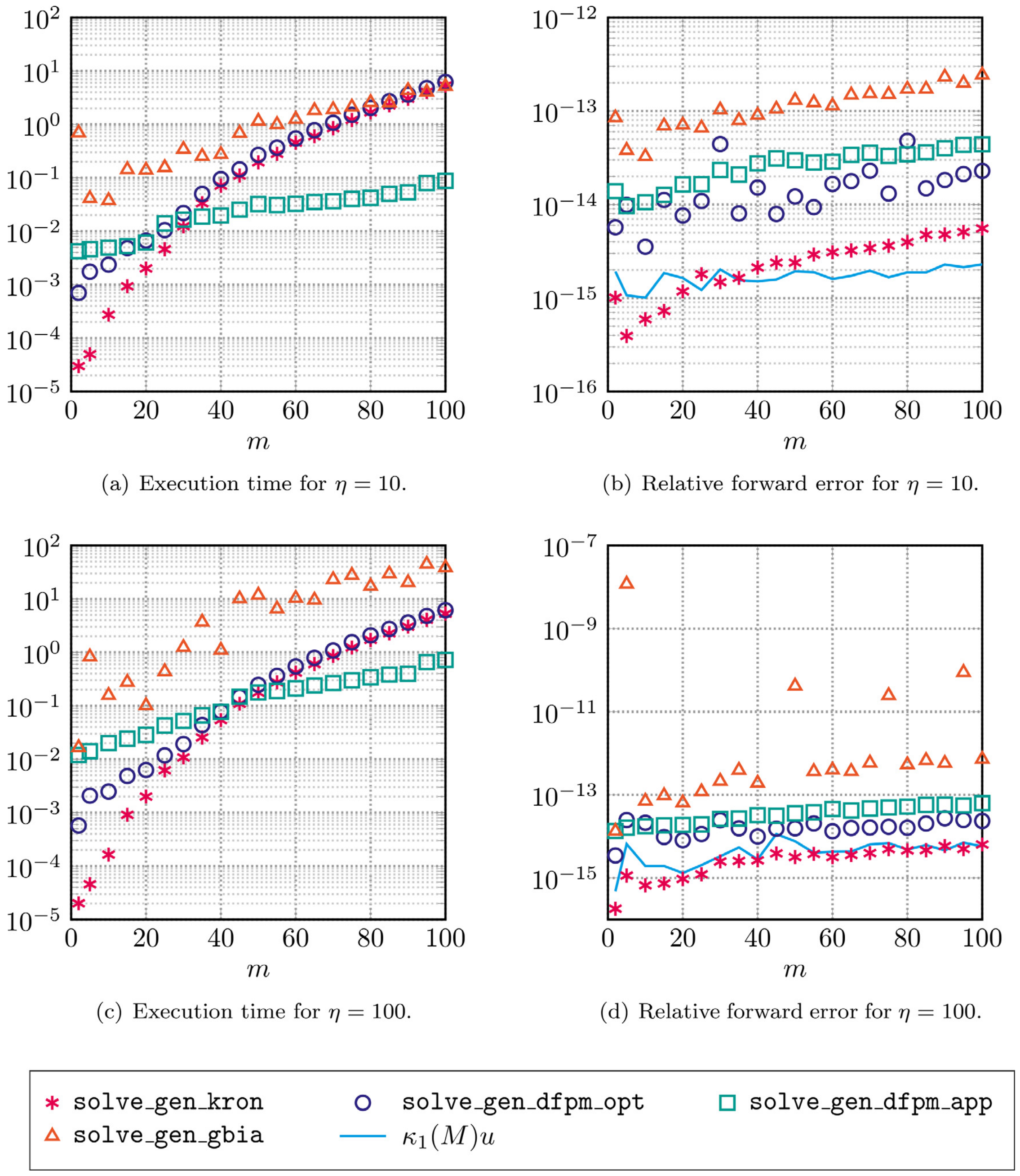

Fig. 2. Left: execution time, in seconds, required by `solve_gen_kron`, `solve_gen_dfpm_opt`, `solve_gen_dfpm_app` and `solve_gen_gbia` to solve the matrix equation (24) with $m = n$ between 1 and 100. Right: relative forward error of the computed solution.

In our implementation, `solve_gen_dfpm_opt` cannot be faster than `solve_gen_kron`: estimating $\lambda_{\min}(M)$ requires the solution of at least one—but typically several—linear systems with coefficient matrix M, thus the computation that `solve_gen_kron` performs is, in a sense, just a pre-processing step for `solve_gen_dfpm_opt`. In order for this method to be competitive, an alternative technique for estimating the smallest eigenvalue of M is necessary, but we do not investigate this further here, since the most appropriate technique is likely to depend on the problem being solved and on a priori knowledge of the properties of the coefficient matrices.

As the size of the matrix coefficients of the equations grows, estimating the extreme eigenvalues of M becomes more expensive, and the lower number of iterations obtained with the optimal choice of parameters is not sufficient to offset the time spent estimating $\lambda_{\min}(M)$. Therefore, for larger matrices the crude choice of parameters of `solve_gen_dfpm_app` pays off, leading to an execution time two order of magnitudes smaller for m as small as 100.

Table 2
Forward error (Error) of solutions computed using `solve_gen_dfpm_app`; and execution time in seconds (Time) and number of iterations (Steps) required to compute these solutions. The experiment considers matrix equations with square coefficients of order m between 250 and 2,000. Two values of the parameter η are considered.

	$\eta = 10$			$\eta = 100$		
m	Error	Time	Steps	Error	Time	Steps
250	8.98×10^{-14}	5.99×10^{-1}	150	1.28×10^{-13}	5.68×10^{0}	1,490
500	1.55×10^{-13}	2.98×10^{0}	147	2.22×10^{-13}	2.86×10^{1}	1,460
750	1.93×10^{-13}	8.53×10^{0}	146	3.10×10^{-13}	8.25×10^{1}	1,440
1,000	2.82×10^{-13}	1.87×10^{1}	144	4.03×10^{-13}	1.80×10^{2}	1,430
1,250	3.45×10^{-13}	3.58×10^{1}	143	4.79×10^{-13}	3.48×10^{2}	1,420
1,500	3.97×10^{-13}	6.04×10^{1}	142	5.72×10^{-13}	5.97×10^{2}	1,410
1,750	4.89×10^{-13}	9.68×10^{1}	141	6.61×10^{-13}	9.53×10^{2}	1,400
2,000	4.84×10^{-13}	1.38×10^{2}	141	7.67×10^{-13}	1.41×10^{3}	1,400

Table 2 reports the performance and accuracy of `solve_gen_dfpm_app` on matrix equations with square coefficients of size between 250 and 2,000. None of the other algorithms considered in Figure 2 can be used for comparison here, as their large execution time makes them an impractical option for problems of this size. Overall, `solve_gen_dfpm_app` scales well with the size of the problem. The forward error grows with the order of the matrix coefficients, but the loss of accuracy grows slower than m, which can be regarded as satisfactory. The number of iterations required to compute the solution remains roughly constant, and in fact slightly decreases as the size of the problem grows.

5. Conclusion

We have developed a family of algorithms for the solution of a class of matrix equations of the form (1), and we have explained how these methods can be tailored to tackle some special cases of particular importance in applications. The new techniques build upon the dynamical functional particle method for the solution of linear systems and exploit the equivalence between the two formulations (1) and (2).

Numerical results show that our implementations are typically capable of outperforming existing methods for the solution of (1) in terms of both accuracy and execution speed. In order to show the potential of our new techniques, we compared the algorithm for the Sylvester equation (7) with the built-in MATLAB function `sylvester`. We found that if the matrix coefficients on the left-hand side are sufficiently well conditioned and have very different size, then our implementation of the discrete functional particle method can be faster and more accurate than the built-in alternative.

The new algorithms require that the eigenvalues of the coefficient matrix M in (2) be real and all have same sign. This follows from an analogous restriction in the dynamical functional particle method for the solution of the linear system $Gy = b$: the method may not converge if the coefficient matrix G has at least a pair of conjugate eigenvalue with arbitrarily small imaginary part. In the case of linear systems, if G is nonsingular then one can consider the equivalent linear system $G^T Gz = G^T b$, since z satisfies $Gz = b$ and the matrix $G^T G$ has only positive real eigenvalues. This comes at a price: on the one hand, the cost of each step of the iteration nearly doubles, on the other, the conditioning of the linear system to be solved grows considerably, since $\kappa_2(G^T G) = \kappa_2(G)^2$ [41, sect. 8.1]. However, this is not necessarily a problem when G is well conditioned, and a similar technique can be used in the algorithm discussed in Section 3.

As the dynamical functional particle method has been successfully applied to nonlinear optimization problems, it is natural to ask whether similar techniques for the solution of nonlinear matrix equations can be derived. This nontrivial problem will be the subject of future work.

Acknowledgements

The work of Andrii Dmytryshyn was supported by the Swedish Research Council (VR) grant 2021-05393. The work of Massimiliano Fasi was supported by the Wenner-Gren Foundations postdoctoral grant UPD2019-0067. The authors thank the anonymous referees for their feedback, which helped them improve the quality of the paper.

References

[1] J.J. Sylvester, Sur l'equations en matrices $px = xq$, C. R. Acad. Sci. Paris 99 (2) (1884) 67–71.
[2] A. Bouhamidi, K. Jbilou, A note on the numerical approximate solutions for generalized Sylvester matrix equations with applications, Appl. Math. Comput. 206 (2) (2008) 687–694, doi:10.1016/j.amc.2008.09.022.
[3] B. Zhou, G.-R. Duan, On the generalized Sylvester mapping and matrix equations, Syst. Control Lett. 57 (3) (2008) 200–208, doi:10.1016/j.sysconle.2007.08.010.
[4] M. Konstantinov, D.-W. Gu, V. Mehrmann, P. Petkov, Perturbation theory for matrix equations, Studies in Computational Mathematics, volume 9, Elsevier, Amsterdam, The Netherlands, 2003, doi:10.1016/s1570-579x(13)60003-8.
[5] P. Lancaster, Explicit solutions of linear matrix equations, SIAM Rev. 12 (4) (1970) 544–566, doi:10.1137/1012104.
[6] A. Ben-Israel, T.N. Greville, Generalized inverses: theory and applications, CMS Books in Mathematics, 2nd, Springer-Verlag, New York, 2003, doi:10.1007/b97366.

[7] P. Benner, T. Damm, Lyapunov equations, energy functionals, and model order reduction of bilinear and stochastic systems, SIAM J. Control Optim. 49 (2) (2011) 686–711, doi:10.1137/09075041x.
[8] Z. Gajić, M.T.J. Qureshi, Lyapunov matrix equation in system stability and control, Mathemtics in Scinences and Engineering, volume 195, Academic Press, San Diego, CA, USA, 1995.
[9] V. Simoncini, Computational methods for linear matrix equations, SIAM Rev. 58 (3) (2016) 377–441, doi:10.1137/130912839.
[10] R.H. Bartels, G.W. Stewart, Algorithm 432: solution of the matrix equation $AX + XB = C$, Comm. ACM 15 (9) (1972) 820–826, doi:10.1145/361573.361582.
[11] A. Dmytryshyn, B. Kågström, Coupled Sylvester-type matrix equations and block diagonalization, SIAM J. Matrix Anal. Appl. 36 (2) (2015) 580–593, doi:10.1137/151005907.
[12] Y. Feng, M. Yagoubi, Robust control of linear descriptor systems, Studies in Systems, Decision and Control, Springer-Verlag, Singapore, 2017, doi:10.1007/978-981-10-3677-4.
[13] N.J. Higham, Accuracy and stability of numerical algorithms, 2nd, Society for Industrial and Applied Mathematics, Philadelphia, PA, USA, 2002, doi:10.1137/1.9780898718027.
[14] W.S. Gray, J. Mesko, Energy functions and algebraic Gramians for bilinear systems, IFAC P. Vol. 31 (17) (1998) 101–106, doi:10.1016/s1474-6670(17)40318-1.
[15] C. Hartmann, B. Schäfer-Bung, A. Thöns-Zueva, Balanced averaging of bilinear systems with applications to stochastic control, SIAM J. Control Optim. 51 (3) (2013) 2356–2378, doi:10.1137/100796844.
[16] H.C. Elman, D.G. Furnival, C.E. Powell, $H(\mathrm{div})$Preconditioning for a mixed finite element formulation of the diffusion problem with random data, Math. Comp. 79 (270) (2009) 733–760, doi:10.1090/s0025-5718-09-02274-1.
[17] O.G. Ernst, C.E. Powell, D.J. Silvester, E. Ullmann, Efficient solvers for a linear stochastic Galerkin mixed formulation of diffusion problems with random data, SIAM J. Sci. Comput. 31 (2) (2009) 1424–1447, doi:10.1137/070705817.
[18] C.E. Powell, D. Silvester, V. Simoncini, An efficient reduced basis solver for stochastic Galerkin matrix equations, SIAM J. Sci. Comput. 39 (1) (2017) A141–A163, doi:10.1137/15m1032399.
[19] S. Edvardsson, M. Gulliksson, J. Persson, The dynamical functional particle method: an approach for boundary value problems, J. Appl. Mech. 79 (2) (2012), doi:10.1115/1.4005563.
[20] D.S. Watkins, Francis's algorithm, Amer. Math. Month. 118 (5) (2011) 387–403, doi:10.4169/amer.math.monthly.118.05.387.
[21] D.S. Watkins, Understanding the QR algorithm, SIAM Rev. 24 (4) (1982) 427–440, doi:10.1137/1024100.
[22] D.S. Watkins, The QR algorithm revisited, SIAM Rev. 50 (1) (2008) 133–145, doi:10.1137/060659454.
[23] G.H. Golub, C.F. Van Loan, Matrix computations, 4th, Johns Hopkins University Press, Baltimore, MD, USA, 2013.
[24] M. Fasi, N.J. Higham, Multiprecision algorithms for computing the matrix logarithm, SIAM J. Matrix Anal. Appl. 39 (1) (2018) 472–491, doi:10.1137/17m1129866.
[25] F. Ding, T. Chen, Gradient based iterative algorithms for solving a class of matrix equations, IEEE Trans. Automat. Control 50 (8) (2005) 1216–1221, doi:10.1109/tac.2005.852558.
[26] F. Ding, T. Chen, On iterative solutions of general coupled matrix equations, SIAM J. Control Optim. 44 (6) (2006) 2269–2284, doi:10.1137/s0363012904441350.
[27] P. Benner, R.-C. Li, N. Truhar, On the ADI method for Sylvester equations, J. Comput. Appl. Math. 233 (4) (2009) 1035–1045, doi:10.1016/j.cam.2009.08.108.
[28] M. Gulliksson, The discrete dynamical functional particle method for solving constrained optimization problems, Dolomites Res. Notes Approx. 10 (Special Issue) (2017) 6–12, doi:10.14658/pupj-drna-2017-Special_Issue-2.
[29] M. Gulliksson, M. Ögren, A. Oleynik, Y. Zhang, Damped dynamical systems for solving equations and optimization problems, Handbook Math. Art. Sci. (2018) 1–44, doi:10.1007/978-3-319-70658-0_32-1.
[30] S. Edvardsson, M. Neuman, P. Edström, H. Olin, Solving equations through particle dynamics, Comput. Phys. Comm. 197 (2015) 169–181, doi:10.1016/j.cpc.2015.08.028.
[31] E. Hairer, G. Wanner, C. Lubich, Geometric numerical integration: structure-preserving algorithms for ordinary differential equations, Springer Series in Computational Mathematics, Springer-Verlag, Berlin, Heidelberg, 2006, doi:10.1007/3-540-30666-8.
[32] Y. Saad, Numerical methods for large eigenvalue problems, revised, Society for Industrial and Applied Mathematics, Philadelphia, PA, USA, 2011, doi:10.1137/1.9781611970739.
[33] V. Druskin, L. Knizhnerman, Extended Krylov subspaces: approximation of the matrix square root and related functions, SIAM J. Matrix Anal. Appl. 19 (3) (1998) 755–771, doi:10.1137/S0895479895292400.
[34] R.A. Horn, C.R. Johnson, Topics in matrix analysis, Cambridge University Press, Cambridge, UK, 1991, doi:10.1017/CBO9780511840371.
[35] J. Fehr, J. Heiland, C. Himpe, J. Saak, Best practices for replicability, reproducibility and reusability of computer-based experiments exemplified by model reduction software, AIMS Math. 1 (3) (2016) 261–281, doi:10.3934/math.2016.3.261.
[36] IEEE Standard for floating-point arithmetic, IEEE std 754–2019 (revision of IEEE std 754–2008), Institute of Electrical and Electronics Engineers, Piscataway, NJ, USA, 2019, doi:10.1109/IEEESTD.2019.8766229.
[37] IEEE Standard for binary floating-point arithmetic, ANSI/IEEE standard 754–1985, Institute of Electrical and Electronics Engineers, Piscataway, NJ, USA, 1985, doi:10.1109/IEEESTD.1985.82928. Reprinted in SIGPAN Notices 22(2) (1987) 9–25.
[38] IEEE Standard for floating-point arithmetic, IEEE std 754–2008 (revision of IEEE std 754–1985), Institute of Electrical and Electronics Engineers, Piscataway, NJ, USA, 2008, doi:10.1109/IEEESTD.2008.4610935.
[39] M. Fasi, N.J. Higham, Generating extreme-scale matrices with specified singular values or condition numbers, SIAM J. Sci. Comput. 43 (1) (2021) A663–A684, doi:10.1137/20M1327938.
[40] G.H. Golub, S. Nash, C.F. Van Loan, A Hessenberg–Schur method for the problem $AX + XB = C$, IEEE Trans. Automat. Control AC-24 (6) (1979) 909–913, doi:10.1109/TAC.1979.1102170.
[41] Y. Saad, Iterative methods for sparse linear systems, 2nd, Society for Industrial and Applied Mathematics, Philadelphia, PA, USA, 2003.