Numer Algor (2018) 78:1129–1151
DOI 10.1007/s11075-017-0416-8

ORIGINAL PAPER

Adaptive high-order splitting schemes for large-scale differential Riccati equations

Tony Stillfjord[1]

Received: 2 December 2016 / Accepted: 8 September 2017 / Published online: 23 September 2017

Abstract We consider high-order splitting schemes for large-scale differential Riccati equations. Such equations arise in many different areas and are especially important within the field of optimal control. In the large-scale case, it is critical to employ structural properties of the matrix-valued solution, or the computational cost and storage requirements become infeasible. Our main contribution is therefore to formulate these high-order splitting schemes in an efficient way by utilizing a low-rank factorization. Previous results indicated that this was impossible for methods of order higher than 2, but our new approach overcomes these difficulties. In addition, we demonstrate that the proposed methods contain natural embedded error estimates. These may be used, e.g., for time step adaptivity, and our numerical experiments in this direction show promising results.

Keywords Differential Riccati equations · Large-scale · Splitting schemes · High order · Adaptivity

Mathematics Subject Classification (2010) 15A24 · 49N10 · 65L05 · 93A15

1 Introduction

We consider differential Riccati equations (DREs) of the form

$$\dot{P} = A^T P + PA + Q - PSP, \quad P(0) = P_0, \tag{1}$$

✉ Tony Stillfjord
tony.stillfjord@gu.se

[1] Mathematical Sciences, Chalmers University of Technology and the University of Gothenburg, SE-412 96 Göteborg, Sweden

where the solution $P(t)$ is matrix-valued and A, Q, and S are given matrices. Such equations arise in many different areas, e.g., in optimal/robust control, optimal filtering, spectral factorizations, $\mathbf{H}_\infty$-control, and differential games [1, 4, 22, 27].

A typical application is a linear quadratic regulator (LQR) problem, where one seeks to control the output $y = Cx$ given the state equation $\dot{x} = Ax + Bu$ by varying the input u. In the case of a finite time cost function,

$$J(u) = \int_0^T x(t)^T R_x x(t) + u(t)^T R_u u(t) \mathrm{d}t,$$

where R_x and R_u are given matrices; it is well known that the optimal input u^* is given in a state feedback form. In particular, $u^*(t) = -R_u^{-1} B^T P(T-t)x(t)$, where P is the solution to the DRE (1) with the specific matrices $Q = C^T R_x C$ and $S = B R_u^{-1} B^T$. We note that the situation $M\dot{x} = Ax + Bu$ can be handled in a straightforward way without explicitly inverting the mass matrix M, see, e.g., [34].

In this paper, we are interested in the large-scale setting. Even if $A \in \mathbb{R}^{N\times N}$ is sparse, the solution P is typically dense. Hence, a "large" dimension N is here considerably smaller than the number of components which would be considered large for a vector-valued ordinary differential equation (ODE). A naive method that works well for the small-scale case would run into storage problems already for $N = 10,000$ and be computationally expensive long before that. Recently, many non-naive methods have been proposed for DREs and similar problems, e.g., matrix-valued BDF and Rosenbrock methods [6, 7], splitting schemes [26, 34], and Krylov projection methods [14, 23]. The latter is a generalization of the Krylov approach to algebraic Riccati equations and Lyapunov equations [13, 20, 31]. Other methods for such equations, like invariant subspace techniques [2, 5, 25], typically also generalize to the DRE case by using time-stepping methods of either one- or multi-step type. Further useful references may be found in the recent surveys [9, 30]. In general, all these methods rely on the fact that the dense solution possesses certain structure. In particular, the solution is positive semi-definite, and in all practical applications, it also has low rank. This allows us to factorize $P = ZZ^T$ where Z is a matrix with many fewer columns than P. A main idea in all the algorithms listed above is then to only do computations on the factor Z and never actually form the product ZZ^T.

Further, we are interested in different types of splitting schemes, since the equation has a natural division into two parts:

$$\dot{P} = \mathcal{F}P + \mathcal{G}P, \quad \text{where} \quad \mathcal{F}P = A^T P + PA + Q \quad \text{and} \quad \mathcal{G}P = -PSP.$$

While the full problem is rather difficult, the subproblems

$$\dot{P} = \mathcal{F}P, \quad P(0) = P_0 \quad \text{and} \tag{2}$$

$$\dot{P} = \mathcal{G}P, \quad P(0) = P_0 \tag{3}$$

are separately much easier and cheaper to solve. In fact, as demonstrated in [34], there exist closed-form expressions for the solutions to both subproblems that are amenable to low-rank computations. In the following, we will denote the solution operator to the full problem by $\mathcal{T}_{\mathcal{F}+\mathcal{G}}$ and to the subproblems by $\mathcal{T}_\mathcal{F}$ and $\mathcal{T}_\mathcal{G}$; thus, for example, the solution to (2) at time t is given by $\mathcal{T}_\mathcal{F}(t)P_0$.

To introduce the simplest splitting schemes and our notation, we first discretize the time interval $[0, T]$ by n equidistant time steps of size h and set $t_j = jh$. Then the approximation to $\mathcal{T}_{\mathcal{F}+\mathcal{G}}(t_j)P_0$ by the Lie splitting scheme is given by $\mathcal{S}_{\text{Lie}}(h)^j P_0$, where

$$\mathcal{S}_{\text{Lie}}(h) = \mathcal{T}_{\mathcal{F}}(h)\mathcal{T}_{\mathcal{G}}(h).$$

That is, we switch back and forth between the affine subproblem and the nonlinear subproblem. A more accurate approximation is given by the Strang splitting scheme, defined by the time stepping operator

$$\mathcal{S}_{\text{Strang}}(h) = \mathcal{T}_{\mathcal{G}}(h/2)\mathcal{T}_{\mathcal{F}}(h)\mathcal{T}_{\mathcal{G}}(h/2).$$

In both cases, we may interchange the order of the $\mathcal{F}$ and $\mathcal{G}$ operators. For a more thorough introduction to splitting schemes in general, we refer to [21].

It can be shown as in [21] that the Lie splitting is first-order convergent and the Strang splitting is second-order convergent, i.e., the errors satisfy

$$\|\mathcal{S}_{\text{Lie}}(h)^j P_0 - \mathcal{T}_{\mathcal{F}+\mathcal{G}}(t_j)P_0\| \leq Ch \quad \text{and} \quad \|\mathcal{S}_{\text{Strang}}(h)^j P_0 - \mathcal{T}_{\mathcal{F}+\mathcal{G}}(t_j)P_0\| \leq Ch^2.$$

In general, one can also consider higher-order schemes, but so far this has not been done for DREs. This is due to the fact that multiplicative splitting schemes of the form $\mathcal{T}_{\mathcal{F}}(\alpha_1 h)\mathcal{T}_{\mathcal{G}}(\beta_1 h)\cdots\mathcal{T}_{\mathcal{F}}(\alpha_s h)\mathcal{T}_{\mathcal{G}}(\beta_s h)$ require that some coefficients α_j and β_j are either negative or complex [10, 19], which is not compatible with the low-rank implementation.

The first main contribution of this work is therefore to demonstrate that a new type of additive splitting schemes introduced in [12] allows for arbitrary high-order schemes to be implemented efficiently in a low-rank DRE setting. These schemes are of the form

$$\gamma_1\mathcal{T}_{\mathcal{F}}(h)\mathcal{T}_{\mathcal{G}}(h) + \gamma_2\big(\mathcal{T}_{\mathcal{F}}(h/2)\mathcal{T}_{\mathcal{G}}(h/2)\big)^2 + \cdots + \gamma_s\big(\mathcal{T}_{\mathcal{F}}(h/s)\mathcal{T}_{\mathcal{G}}(h/s)\big)^s$$

and thus only utilize positive step sizes. A minor drawback is that the approximations are no longer guaranteed to be positive semi-definite, since the coefficients γ_j may be negative. This prohibits the use of a ZZ^T factorization, and we therefore outline the changes necessary to instead consider a so-called LDL^T factorization (cf. [24]).

The second main contribution lies in the observation that these splitting schemes contain natural lower-order embedded methods, which allows for cheap and easy error estimation. We utilize this to construct high-order splitting schemes with adaptive time stepping, i.e., the time steps $h_j = t_{j+1} - t_j$ are no longer equidistant but chosen as large as possible while keeping the error below a given tolerance. Modifying the step size can greatly increase the efficiency, but only if the computational cost of changing the step size is small. We therefore outline which quantities can be precomputed or recomputed cheaply, and describe efficient updating strategies for the quantities that necessarily change with each step.

A brief outline of the paper is as follows. In Section 2, we state the basic assumptions on the given data and review the use of the ZZ^T and LDL^T factorizations for low-order splitting schemes. The issues that arise when considering higher-order multiplicative splitting schemes are outlined in Section 3, wherein we also present the new type of additive schemes that eliminate these issues. Error estimates and different kinds of time step adaptivity are discussed in Section 4 and an algorithm

summarizing the complete implementation is presented. In Section 5, several numerical experiments demonstrate the validity of the implementation, the efficiency of the methods, and the use of adaptive time stepping. Finally, we collect some conclusions in Section 6.

2 Low-rank factorizations

The first assumption we make on the problem data is the following:

Assumption 1 The matrices A, Q, and S and the initial condition P_0 all belong to $\mathbb{R}^{N\times N}$. In addition, Q, S, and P_0 are symmetric and positive semi-definite.

This implies the existence and uniqueness of a solution P to the DRE (1) such that $P(t)$ is also symmetric and positive semi-definite for all $t \geq 0$ [1, Theorem 4.1.6]. An important example of when Assumption 1 is satisfied is the LQR setting from the introduction, with R_x and R_u both symmetric positive definite. Secondly, we assume that the solution has the low-rank property:

Assumption 2 For each $t \in [0, T]$, the rank of the solution $P(t)$ is at most $r \ll N$ and the rank of Q is $r_Q \ll N$.

To the author's knowledge, there are currently no known useful criteria on the data in the DRE setting which guarantee that Assumption 2 is fulfilled. However, such low-rank structure is observed in all practical applications, e.g., in LQR problems where $B \in \mathbb{R}^{N\times m_B}$ and $C \in \mathbb{R}^{m_C\times N}$ with $m_B, m_C \ll N$. Recently, some results in this direction have been established for algebraic Riccati equations, i.e., the stationary version of (1), in [5]. These are generalizations of results for Lyapunov equations [3, 33] and it seems likely that further generalizations to the DRE setting could be made.

Assumptions 1 and 2 imply that we can low-rank factorize $P(t) = Z(t)Z(t)^T$ and $Q = EE^T$ with $Z(t) \in \mathbb{R}^{N\times r}$ and $E \in \mathbb{R}^{N\times r_Q}$. Similarly, as demonstrated in [34], we can low-rank factorize also the approximations $\mathcal{S}_{\text{Lie}}(h)P_0$ and $\mathcal{S}_{\text{Strang}}(h)P_0$. This is based on factorizing the exact solutions to the subproblems (2)–(3), for which we have the closed-form expressions

$$\begin{aligned}\mathcal{T}_{\mathcal{F}}(h)P_0 &= \mathrm{e}^{hA^{\mathsf{T}}}P_0\mathrm{e}^{hA} + \int_0^h \mathrm{e}^{sA^{\mathsf{T}}}Q\mathrm{e}^{sA}\mathrm{d}s \quad \text{and}\\ \mathcal{T}_{\mathcal{G}}(h)P_0 &= (I + hP_0S)^{-1}P_0.\end{aligned}$$

The latter expression quickly yields an explicit factorization while the former requires that the integral is approximated by a quadrature formula, whereafter column compression is applied.

Considering instead a so-called LDL^T factorization where $L(t) \in \mathbb{R}^{N\times r}$ and $D(t) \in \mathbb{R}^{r\times r}$ is beneficial for many schemes [24], because it can decrease the amount

of computations. This is true also for splitting schemes. Assuming that $P_0 = LDL^T$ and considering first the nonlinear subproblem, we have

$$\mathcal{T}_{\mathcal{G}}(h)P_0 = (I + hLDL^T S)^{-1}LDL^T = L(I + hDL^T SL)^{-1}DL^T,$$

by use of a simplified version of the Woodbury matrix inversion formula [17]. Thus, $\hat{L}\hat{D}\hat{L}^T$ is a low-rank factorization of the solution to the nonlinear subproblem, where $\hat{L} = L$ and $\hat{D} = (I + hDL^T SL)^{-1}D$. In contrast to the ZZ^T situation, there exist matrices D and L such that $I + hDL^T SL$ is not invertible for all h. However, it certainly is for all $h < 1/\rho(DL^T SL)$, where ρ denotes the spectral radius, and therefore, the step size can always be chosen such that $\hat{D}$ is well defined. We note that even for large time steps, this theoretical issue has not yet been observed in practice. We also note that this formulation is cheaper to compute than the corresponding ZZ^T factorization, since it is no longer necessary to compute a Cholesky factorization of the inverse.

Considering next the affine subproblem and assuming that $Q = L_Q D_Q L_Q^T$, we have

$$\begin{aligned}\mathcal{T}_{\mathcal{F}}(h)P_0 &= \mathrm{e}^{hA^{\mathsf{T}}}LDL^T\mathrm{e}^{hA} + \int_0^h \mathrm{e}^{sA^{\mathsf{T}}}L_Q D_Q L_Q^T \mathrm{e}^{sA}\mathrm{d}s \\ &= L_1 DL_1^T + \int_0^h L(s)D_Q L(s)^T \mathrm{d}s \\ &\approx L_1 DL_1^T + \sum_{k=1}^{n_Q} w_k L(s_k)D_Q L(s_k)^T,\end{aligned}$$

where $L_1 = \mathrm{e}^{hA^{\mathsf{T}}}L$, $L(s) = \mathrm{e}^{sA^{\mathsf{T}}}L_Q$, and (s_k, w_k) are the n_Q nodes and weights of a quadrature formula. We choose the parameters such that the error in this approximation is negligible with respect to the splitting error; for a splitting scheme of order p, we typically choose a quadrature formula of order $p+1$. For efficiency, the structure of A (sparsity, bandedness, etc.) should be taken into account when computing the terms L_1 and $L(s)$. In our tests, we simply use a fifth-order implicit Runge-Kutta method with a crude error estimate based on halving the internal step size. It seems likely, however, that an approach based on, e.g., Krylov subspaces or the Leja point method (see, e.g., [11]) would be even more efficient, especially if subspaces from previous steps can be (partially) reused. We note that these terms do not need to be computed to full precision, but (like for the integral term) their errors should be negligible in comparison to the splitting error. Then, similarly to the ZZ^T case, setting

$$\tilde{L} = \begin{bmatrix} L_1 & L(s_1) & \cdots & L(s_{n_Q}) \end{bmatrix} \quad \text{and} \quad \tilde{D} = \begin{bmatrix} D & & & \\ & w_1 D_Q & & \\ & & \ddots & \\ & & & w_{n_Q} D_Q \end{bmatrix}$$

means that $\tilde{L}\tilde{D}\tilde{L}^T$ is a low-rank approximation of the solution to the affine subproblem. After forming $\tilde{L}$ and $\tilde{D}$, column compression should be applied to eliminate any unnecessary columns. We refer to [24] for an efficient way to do this.

3 High-order splitting schemes

Let us now consider low-rank factorization of higher-order multiplicative splitting schemes like the Lie and Strang splitting schemes. Let

$$\mathcal{S}(h) = \mathcal{T}_{\mathcal{F}}(\alpha_1 h)\mathcal{T}_{\mathcal{G}}(\beta_1 h)\cdots\mathcal{T}_{\mathcal{F}}(\alpha_s h)\mathcal{T}_{\mathcal{G}}(\beta_s h)$$

with s and the coefficients $\{\alpha_k\}_{k=1}^s$ and $\{\beta_k\}_{k=1}^s$ be chosen such that $\mathcal{S}(h)$ is a splitting scheme of order $p \geq 3$. Then the coefficients must include either negative or complex values [10, 19]. In the first case, computing $\mathrm{e}^{\gamma h A^T} P_0$ for such a negative coefficient γ corresponds to taking a negative time step for the system $\dot{x} = A^T x$. If A, e.g., corresponds to a discretization of the Laplacian (a common application), we are thus solving the heat equation backwards in time, which is ill-posed. It is therefore only possible to consider the class of problems where A corresponds to the discretization of an analytic operator, but even in this case, the evaluation of $(I + hZ^T SZ)^{-1}$ or $(I + hDL^T SL)^{-1}$ tends to yield step size restrictions. We therefore do not think that this is a worthwhile direction of research to pursue.

In the case of a ZZ^T factorization, a complex coefficient γ destroys the structure of $I + \gamma h Z^T SZ$ and we can only factorize it in very special cases. Considering instead an LDL^T factorization leads to problems with complex arithmetic: If L and D are real, the approximation $\hat{L}\hat{D}\hat{L}^T$ to $\mathcal{T}_{\mathcal{G}}(\gamma h)LDL^T$ will have $\hat{L}$ real but $\hat{D}$ complex-valued. Such input to the affine subproblem will then lead to both $\hat{L}$ and $\hat{D}$ being complex-valued. Once this is the case, we not only have to do computations fully in complex arithmetic but we also have issues with column compression since the complex values do not match the "transpose" formulation. Switching instead to a complex LDL^H factorization results in similar issues. Like negative coefficients, using complex coefficients thus does not seem worthwhile.

However, the necessity of negative or complex coefficients only holds for the type of multiplicative splitting schemes mentioned above. Recently, a new type of additive splitting schemes was introduced in [12]. These are either of the asymmetric type

$$\mathcal{S}^s_{\mathrm{asym}}(h) = \sum_{k=1}^{s} \gamma_k \big(\mathcal{T}_{\mathcal{F}}(h/k)\mathcal{T}_{\mathcal{G}}(h/k)\big)^k, \tag{4}$$

which are of order s if the coefficients $\gamma_1, \ldots, \gamma_s$ are chosen appropriately, and the symmetric type

$$\mathcal{S}^{2s}_{\mathrm{sym}}(h) = \sum_{k=1}^{s} \gamma_k \Big(\big(\mathcal{T}_{\mathcal{F}}(h/k)\mathcal{T}_{\mathcal{G}}(h/k)\big)^k + \big(\mathcal{T}_{\mathcal{G}}(h/k)\mathcal{T}_{\mathcal{F}}(h/k)\big)^k \Big), \tag{5}$$

which are of order $2s$. (We only consider the case of minimal number of stages here. One might of course add extra stages in order to improve the local error structure, but given the form of the schemes, it would then make more sense to instead increase the order.) In both cases, the roles of $\mathcal{F}$ and $\mathcal{G}$ may be interchanged.

At first sight, these methods may look computationally expensive. However, (as noted in [12]) if we have the possibility to work in parallel, then taking one step with either method is only as expensive as taking s Lie splitting steps. More important

is that they only require real, positive step sizes. This eliminates all the issues listed above and allows us to consider splitting schemes for DREs of arbitrarily high order.

Because the coefficients $\{\gamma_k\}_{k=1}^s$ may include negative values, using a ZZ^T factorization to formulate these methods is impossible. However, instead, using an LDL^T factorization is not only possible but rather straightforward after the preliminary work in the previous section. The only additional computational work is a column compression step after forming the linear combinations. In an optimized code, most of this work could additionally be done while waiting for the slowest processor that takes s steps to finish. Using a higher-order method also requires us to compute terms of the form $\mathrm{e}^{\gamma h A^T} L$ more accurately (unless we also increase the step size h and thereby the error) and to use a higher-order quadrature formula to approximate the integral term in $\mathcal{T}_{\mathcal{F}}(h)P_0$.

We also note here that while the LDL^T factorization does not guarantee that the approximations are positive semi-definite, in practice, this still seems to hold. This is likely due to the fact that the approximations are very close to the solution of the full problem, which is guaranteed to be positive semi-definite.

4 Time adaptivity

An additional major feature of the schemes (4) and (5) is the existence of natural embedded lower-order methods. This seems to have been overlooked by [12]. For example, the scheme

$$\mathcal{S}^2_{\mathrm{asym}}(h) = -1\big(\mathcal{T}_{\mathcal{F}}(h)\mathcal{T}_{\mathcal{G}}(h)\big) + 2\big(\mathcal{T}_{\mathcal{F}}(h/2)\mathcal{T}_{\mathcal{G}}(h/2)\big)^2$$

is of order 2, and it obviously contains the first-order method $\mathcal{T}_{\mathcal{F}}(h)\mathcal{T}_{\mathcal{G}}(h)$. This holds true for all the schemes, symmetric and asymmetric. In general, neglecting the last terms of the sum and using other coefficients $\{\gamma_k\}$ yields embedded methods of order $p-1$ in the asymmetric case and of order $2p-2$ in the symmetric case with $p = 2, 3, \ldots, s$. Since these lower-order approximations are simply linear combinations of previously computed terms, they are cheap to compute. In our case, the only extra computational effort is a column compression step.

The embedded methods yield natural error estimates. For example, we have

$$\begin{aligned}\big(\mathcal{S}^s_{\mathrm{asym}}(h)P_0 - \mathcal{T}_{\mathcal{F}+\mathcal{G}}(h)P_0\big) - \big(\mathcal{S}^{s-1}_{\mathrm{asym}}(h)P_0 - \mathcal{T}_{\mathcal{F}+\mathcal{G}}(h)P_0\big)\\ = \Phi^s(P_0)h^{s+1} + \mathcal{O}(h^{s+2}) - \Phi^{s-1}(P_0)h^s + \mathcal{O}(h^{s+1})\\ = -\Phi^{s-1}(P_0)h^s + \mathcal{O}(h^{s+1}),\end{aligned}$$

where Φ^s and Φ^{s-1} are the principal error functions of the two methods. Thus, the difference $\mathcal{S}^s_{\mathrm{asym}}(h)P_0 - \mathcal{S}^{s-1}_{\mathrm{asym}}(h)P_0$ is a local error estimate of order $s-1$. In the symmetric case, we instead get an error estimate of order $2s-2$.

These error estimators may be used to control the size of h, with the aim of keeping the local error below a certain tolerance while minimizing the computational effort. There are many different kinds of such controllers, see, e.g., [16, 32]. As an

example, we choose a simple PI controller which typically provides a smoother step size sequence than the commonly used deadbeat I controller. It is given by [16, 32]

$$h_{n+1} = \left(\frac{\epsilon\,\mathrm{TOL}}{e_{n+1}}\right)^{k_I}\left(\frac{e_n}{e_{n+1}}\right)^{k_P} h_n,$$

where h_n is the nth time step, e_n is the error estimate at t_n, TOL is the desired accuracy (tolerance), and ϵ is a safety factor . The parameters (k_I, k_P) determine the characteristics of the controller such as responsiveness and robustness. In our numerical experiments, we set $\epsilon = 0.9$ and $(k_I, k_P) = (0.2/p, 0.2/p)$, where p is the order of the error estimate. These are similar to the values recommended for explicit Runge-Kutta methods when using the error per unit step strategy [15]. Clearly, these are not optimal values for splitting schemes, but an in-depth investigation for a variety of typical problems is out of the scope of this paper.

The evaluation of $\mathcal{T}_{\mathcal{G}}(h)P_0$ requires the same effort whether the step size is varying or not. Evaluating $\mathcal{T}_{\mathcal{F}}(h)P_0$, on the other hand, requires that the approximation of the integral term

$$I_Q(h_n) = \int_0^{h_n} \mathrm{e}^{sA^\mathsf{T}} Q \mathrm{e}^{sA}\mathrm{d}s$$

is recomputed in every step, while it previously could be precomputed. We note that typically the rank of Q is sufficiently small in relation to the rank of the solution approximation that this extra computational cost is small and easily outweighed by the benefits of adaptivity. Nevertheless, we suggest here a strategy to decrease this cost further.

We need to compute $\sum_{k=1}^{n_Q} w_k L(s_k) D_Q L(s_k)^T$ where $L(s) = \mathrm{e}^{sA^\mathsf{T}} L_Q$ for given nodes s_k and weights w_k. The main idea now is to change only a few nodes (and thereby also the weights) in each step, such that the interval $[0, h_n]$ is covered as evenly as possibly. For a quadrature rule of order p, we need $n_Q = p + 1$ nodes if we do not place the nodes optimally, in contrast to, e.g., Gaussian quadrature which would need roughly half as many. However, by storing the computed matrices $L(s_k)$ and keeping most of the nodes unchanged, we will still decrease the overall computation cost. Thus, we define the initial nodes by $s_k = \frac{kh_1}{p}$ for $k = 0, \ldots, p$ and compute the initial weights from

$$\begin{bmatrix} s_0^0 & s_1^0 & \cdots & s_p^0 \\ s_0^1 & s_1^1 & & \vdots \\ \vdots & & \ddots & \\ s_0^p & \cdots & & s_p^p \end{bmatrix} \begin{bmatrix} w_0 \\ w_1 \\ \vdots \\ w_p \end{bmatrix} = \begin{bmatrix} h_n \\ h_n^2/2 \\ \vdots \\ h_n^{p+1}/(p+1) \end{bmatrix} \tag{6}$$

with $n = 1$. Then, to update these nodes and weights given a new h_n, we follow the procedure outlined in algorithmic form in Algorithm 1. (In Algorithm 1 and in the following, blkdiag denotes the block diagonal operator, i.e., it places its block arguments on the diagonal of an otherwise zero matrix.)

Essentially, we add a node at h_n if the interval increases and then remove the node which makes the remaining sequence as close to equidistributed as possible.

Similarly, if the interval decreases, we iteratively relocate the nodes that are outside the new interval to the midpoints of the largest gaps between the nodes in the new interval. In order to ensure that the nodes s_k cover the interval $[0, h_n]$ well, we recompute the whole sequence if the step size changes by more than 25%. We note that we could of course, in theory, store all the previously computed $L(s_k)$ and use increasingly high-order quadrature formulae. However, this would yield a major increase in the storage requirements while having little effect on the overall accuracy.

Remark 1 We note that in, e.g., a real-world optimal control problem, it is frequently the case that the state of the system is sampled at regular, predetermined intervals. The feedback control thus needs the solution of the corresponding DRE at these specific times. This suggests that a constant, matching step size should be employed, or that the adaptive step size is restricted. Neither approach is desirable; the former is inefficient compared to the adaptive approach, and the latter destroys the smooth time step sequence the PI controller is intended to provide. However, assuming that the exact solution is sufficiently regular, we may still use the more efficient adaptive time stepping and simply interpolate the computed approximations to find the values at the desired times. For example, assume that we use piecewise linear interpolation. Then on the interval $[t_{n-1}, t_n]$, the error between the interpolant P_I and the exact solution P is bounded by

$$\|P_I(t) - P(t)\| \leq \mathrm{TOL} + h_n^2 \sup_{s\in[t_{n-1},t_n]} \|\ddot{P}(s)\|/8.$$

To estimate the second term, we can first use the available approximation $P_n \approx P(t_n)$ to estimate $\dot{P}(t_n) \approx A^T P_n + P_n A + Q - P_n S P_n$. Then by differentiating (1), we get $\ddot{P} = A^T \dot{P} + \dot{P} A - \dot{P} S P - P S \dot{P}$, from which we can estimate $\ddot{P}(t_n)$. Both of these operations may be low-rank factorized; if $P_n = L D L^T$ then $\dot{P}(t_n) \approx \tilde{L}\tilde{D}\tilde{L}^T$ where $\tilde{L} = \begin{bmatrix} A^T L & L & L_Q \end{bmatrix}$, and $\ddot{P}(t_n) \approx \hat{L}\hat{D}\hat{L}^T$ where $\hat{L} = \begin{bmatrix} A^T\tilde{L} & \tilde{L} & L \end{bmatrix}$ (with appropriate 3×3 block matrices $\tilde{D}$ and $\hat{D}$). Thus, the norm of $\ddot{P}$ may be estimated efficiently by two applications of A^T and two column compression operations. This estimation may be incorporated into the step size controller to automatically ensure that the interpolation error is bounded by a fixed tolerance. Whether this is cost-effective or not is of course heavily dependent on the rank of the approximation and, thus, of the problem data.

To actually compute the interpolant in a low-rank setting, we note that if $P_{n-1} = L_{n-1} D_{n-1} L_{n-1}^T$ and $P_n = L_n D_n L_n^T$, then $L = \begin{bmatrix} L_{n-1} & L_n \end{bmatrix}$ and $D = \begin{bmatrix} \alpha D_{n-1} & 0 \\ 0 & (1-\alpha) D_n \end{bmatrix}$ constitute a low-rank factorization of $\alpha P_{n-1} + (1-\alpha) P_n$, so that this computation comes at the cost of one column compression step. This interpolation procedure is less straightforward if the setting is generalized to that of time-varying matrices. However, in that case, the strategy of sampling the system at constant time intervals is also rather dubious.

Remark 2 It is enough if the terms involved in one step of the splitting methods are of the same accuracy as the local error. Therefore, the error estimates may additionally

Algorithm 1 Updating the low-rank factorization of $I_Q(h_n)$

Input: Old and new step sizes (h_{n-1}, h_n), previous nodes $\{s_k\}_{k=0}^p$, matrices $\{L(s_k)\}_{k=0}^p$

if $h_n \le 0.8\, h_{n-1}$ **or** $h_n \ge 1.25\, h_{n-1}$ **then**
  Set $\hat{s}_k = \frac{k h_n}{p}$, $k = 0, \ldots, p$
  Recompute all $L(\hat{s}_k)$
else if $h_n > h_{n-1}$ **then**
  Set $\hat{s}_k = s_k$, $k = 0, \ldots, p$, and $\hat{s}_{p+1} = h_n$
  Remove the node $\hat{s}_j$ such that $d_j = \min_{0 \le k \le p+1} d_k$, where $d_0 = \hat{s}_1$, $d_{p+1} = h_n - \hat{s}_p$ and $d_k = \hat{s}_{k+1} - \hat{s}_{k-1}$ for $k = 1, \ldots, p$
  if the new node $\hat{s}_{p+1}$ was removed **then**
    Set $L(\hat{s}_k) = L(s_k)$, $k = 0, \ldots, p$
  else
    Compute $L(h_n) = \mathrm{e}^{h_n A^T} L_Q$
    Set $L(\hat{s}_k)$ to the matrices $L(s_k)$ and $L(h_n)$ that match the nodes $\hat{s}_k$
  end if
else if $h_n < h_{n-1}$ **then**
  Find the number of nodes to recompute: $\tilde{n} = p+1-j$, with $j = \max\{k : s_k \le h_n\}$
  if $\tilde{n} = 0$ **then**
    Set $\hat{s}_k = s_k$ and $L(\hat{s}_k) = L(s_k)$ for $k = 0, \ldots, p$, i.e. do nothing
  else
    Set $\hat{s}_k = s_k$ and $L(\hat{s}_k) = L(s_k)$ for $k = 0, \ldots, j$
    for $l = 1, \ldots, \tilde{n}$ **do**
      Find i such that $d_i = \max_{0 \le k \le j+l} d_k$, where $d_0 = \hat{s}_0$, $d_{j+l} = h_n - \hat{s}_{j+l-1}$ and $d_k = \hat{s}_k - \hat{s}_{k-1}$ for $k = 1, \ldots, j+l-1$
      Add a new node $\hat{s}_{j+l}$ at $\hat{s}_0/2$ if $i = 0$, at $(h_n + \hat{s}_{j+l-1})/2$ if $i = j+l$ or at $(s_i + s_{i-1})/2$ if $1 \le i \le j+l-1$
      Compute $L(\hat{s}_{j+l}) = \mathrm{e}^{\hat{s}_{j+l} A^T} L_Q$
      Reorder $\hat{s}_k$ and $L(\hat{s}_k)$ so that the nodes are increasing
    end for
  end if
end if
Compute new weights $\{\hat{w}_k\}_{k=0}^p$ from (6)
Form $\hat{L} = \begin{bmatrix} L(\hat{s}_0) & \cdots & L(\hat{s}_p) \end{bmatrix}$ and $\hat{D} = \mathrm{blkdiag}(\hat{w}_0 D_Q, \ldots, \hat{w}_p D_Q)$
Column-compress $\hat{L}$ and $\hat{D}$

Output: New nodes $\{\hat{s}_k\}_{k=0}^p$, matrices $\{L(\hat{s}_k)\}_{k=0}^p$, weights $\{\hat{w}_k\}_{k=0}^p$, matrices $\hat{L}$ and $\hat{D}$ such that $\hat{L}\hat{D}\hat{L}^T \approx I_Q(h_n)$

be used to determine the optimal tolerances for column compression and the actions of the matrix exponentials. As these quantities are obviously not independent, however, a proper implementation requires some care. We have not used this feature in our numerical experiments and instead rely on experience to choose reasonable tolerances.

Finally, we present the full procedure for approximating the solution to (1) in algorithmic form in Algorithms 1–4. We consider only the symmetric case of the additive splitting schemes, since the asymmetric version is analogous; change the order of the error estimator from $2s-2$ to $s-1$ and only use the L_+ or L_- terms instead of both. When a step is rejected, it is likely that it is because the approximation of $I_Q(h_n)$ is poor. We thus first recompute the whole sequence of quadrature nodes and then retry the step with the same step size. Only if this also fails do we decrease the step size and proceed as normal.

Remark 3 The s computations on Line 5 of Algorithm 4 could be performed by first approximating $I_Q(h_n/s)$, then approximating the integral over the interval $[h_n/s, h_n/(s-1)]$ and so on. This results in less total work than s independent evaluations, though the locations of the quadrature points then require extra care. Additionally, parallelization could still be faster. The code used for the experiments in the next section employs neither of these two options and simply scales the current quadrature points by $1/j$, $j=1,\ldots,s$.

Algorithm 2 Computing the low-rank factorization of $\mathcal{T}_{\mathcal{G}}(h)P_0$

Input: Matrices $S \in \mathbb{R}^{N\times N}$, $L_0 \in \mathbb{R}^{N\times r}$ and $D_0 \in \mathbb{R}^{r\times r}$ with $P_0 = L_0 D_0 L_0^T$, step size h

1: Compute $D = (I + hD_0 L_0^T S L_0)^{-1} D_0$

2: Set $L = L_0$

Output: Matrices L and D such that $LDL^T \approx \mathcal{T}_{\mathcal{G}}(h)P_0$

Algorithm 3 Computing the low-rank factorization of $\mathcal{T}_{\mathcal{F}}(h)P_0$

Input: Matrices $L_0 \in \mathbb{R}^{N\times r}$, $D_0 \in \mathbb{R}^{r\times r}$ such that $P_0 = L_0 D_0 L_0^T$, step size h, approximate low-rank factorization $L_I D_I L_I^T$ of $I_Q(h)$

1: Compute $\hat{L} = \mathrm{e}^{hA^T} L_0$

2: Form $L = [\hat{L}, L_I]$ and $D = \begin{bmatrix} D_0 & 0 \\ 0 & D_I \end{bmatrix}$ and column-compress

Output: Matrices L and D such that $LDL^T \approx \mathcal{T}_{\mathcal{F}}(h)P_0$

5 Numerical experiments

In order to verify the validity of the proposed splitting schemes, a number of numerical experiments were performed using MATLAB implementations of the presented algorithms.

Different norms may be used to measure the errors. In all our experiments, we consider relative errors at the final time, measured in the Frobenius norm. That is, if

Algorithm 4 Approximating the solution to (1)

Input: Matrices $A, S \in \mathbb{R}^{N\times N}$, $L_Q \in \mathbb{R}^{N\times r_Q}$ and $D_Q \in \mathbb{R}^{r_Q\times r_Q}$ such that $Q = L_Q D_Q L_Q^T$, $L_0 \in \mathbb{R}^{N\times r}$ and $D_0 \in \mathbb{R}^{r\times r}$ such that $P_0 = L_0 D_0 L_0^T$

Input: Desired method order $2s$, coefficients $\{\gamma_k\}_{k=1}^s$ for order $2s$, coefficients $\{\beta_k\}_{k=1}^s$ for order $2s-2$, initial time step h_1, desired error tolerance TOL

Input: Equidistant nodes s_k and weights w_k for a quadrature rule of order s+1 on $[0, h_1]$

Set $\alpha_k = \gamma_k - \beta_k$ for $k = 1, \ldots, s-1$ and $\alpha_s = \gamma_s$

Set $k_I = 0.2/(2s-2)$, $k_P = 0.2/(2s-2)$

Set $n = 1$, $t_n = 0$ and $e_n = 0$

while $t_n + h_n \le T$ **do**

  Low-rank approximate $I_Q(h_n/j) \approx L_I^j D_I^j (L_I^j)^T$, $j = 1, \ldots, s$, according to Algorithm 1, store the computed $L(s_k)$

  Compute in parallel $L_\pm^j$ and $D_\pm^j$ such that

$$L_+^j D_+^j (L_+^j)^T = \left(\mathcal{T}_\mathcal{F}(h_n/j)\mathcal{T}_\mathcal{G}(h_n/j)\right)^j L_{n-1} D_{n-1} L_{n-1}^T \quad \text{and}$$

$$L_-^j D_-^j (L_-^j)^T = \left(\mathcal{T}_\mathcal{G}(h_n/j)\mathcal{T}_\mathcal{F}(h_n/j)\right)^j L_{n-1} D_{n-1} L_{n-1}^T,$$

  for $j = 1, \ldots, s$, according to Algorithms 2 and 3

  Form $L_n = [L_+^1\ L_-^1 \cdots L_+^s\ L_-^s]$, $D_n = \text{blkdiag}(\gamma_1 D_+^1, \gamma_1 D_-^1, \ldots, \gamma_s D_+^s, \gamma_s D_-^s)$ and column compress

  Form $\hat{L}_n = L_n$, $\hat{D}_n = \text{blkdiag}(\alpha_1 D_+^1, \alpha_1 D_-^1, \ldots, \alpha_s D_+^s, \alpha_s D_-^s)$ and column compress

  Compute the error estimate $e_{n+1} = \|\hat{L}_n \hat{D}_n \hat{L}_n^T\|_F = \left(\text{trace}\left(\left(\hat{L}_n^T \hat{L}_n \hat{D}_n\right)^2\right)\right)^{1/2}$

  if $e_{n+1} >$ TOL **then**

    Reject the step

    If first rejection, do a full recomputation of $I_Q(h_n/j)$ and redo step with same h_n

    If still rejected, redo step with $h_n = \left(\frac{0.9\,\text{TOL}}{e_{n+1}}\right)^{1/(2s-2)} h_n$

  else

    Set $t_n = t_{n-1} + h_n$

    if $t_n = T$ **then**

      break

    end if

    Update the time step by $h_{n+1} = \left(\frac{0.9\,\text{TOL}}{e_{n+1}}\right)^{k_I} \left(\frac{e_n}{e_{n+1}}\right)^{k_P} h_n$

    Set $n = n + 1$

  end if

  if $t_n + h_n > T$ **then**

    Set $h_n = T - t_n$

  end if

end while

Output: Time steps $t_k \in [0, T]$, approximations L_k, D_k such that $L_k D_k L_k^T \approx P(t_k)$

the approximation P_n and a given reference approximation P_{ref} both approximate the solution $P(T)$, the error is given by

$$\frac{\|P_n - P_{\text{ref}}\|_F}{\|P_{\text{ref}}\|_F},$$

where $\|\cdot\|_F$ denotes the Frobenius norm.

5.1 Order investigation, small-scale

As a first test, we demonstrate that the methods exhibit the expected orders of convergence when constant step sizes are used. For this, we consider a small-scale problem with $N = 10$ and take A, Q, S, and P_0 to be random matrices with the latter three having rank 4. The small dimension of the problem means that we may compute a highly accurate reference approximation by unrolling the matrix-valued problem into a vector-valued problem of dimension N^2 and applying a standard method for ODEs. Here we utilize the MATLAB built-in function `ode15s`, which implements an adaptive variable-order multistep method, with an absolute tolerance of 10^{-20} and a relative tolerance of $2.22 \cdot 10^{-14}$ (the minimum).

For this test, we consider the asymmetric splitting schemes (4) of orders 2 and 3, the symmetric schemes (5) of orders 2, 4, 6, and 8, as well as the second-order Strang splitting. To compute terms of the form $\mathrm{e}^{hA^T} L$, we use the fifth-order implicit Runge-Kutta scheme RadauIA [18, Chapter IV.5] and halve the step size until two subsequent approximations differ (relatively) by at most 10^{-6}. This can clearly be done better, ideally with adaptive time stepping also on this level, but it is sufficient for our purposes. We set the column compression tolerance to 10^{-16} so that it has no effect on the results.

The results are shown in Fig. 1, where it can be seen that all the methods do, indeed, achieve the expected converge orders. However, a few comments are in order. First, the third-order asymmetric scheme actually exhibits an order of convergence which is slightly larger than 3. This is not true in general and we interpret this as the structure of the error being favorable for this particular problem. Secondly, the errors for the sixth- and eighth-order methods level out around 10^{-12}. This is due to round-off error accumulation in each step. Using a dense instead of low-rank factored version of the code, computing e^{hA} explicitly and approximating $I_Q(h)$ to high accuracy gives similar results. The leveling out of all the error curves for large step sizes is due to leaving the asymptotic regime; for these step sizes, also lower-order error terms influence the result. Thirdly, we note that the second-order asymmetric method performs slightly better than both the Strang splitting and the second-order symmetric method. However, since it is 50% more expensive if parallelization is not used, and even more so if it is, we clearly still prefer the symmetric method.

5.2 Order investigation, larger-scale

We consider also a larger, real-world problem, arising from the optimal control of steel cooling [8, 28]. This is essentially a finite element discretization of a semi-linear PDE given on a non-convex two-dimensional domain. It results in matrices

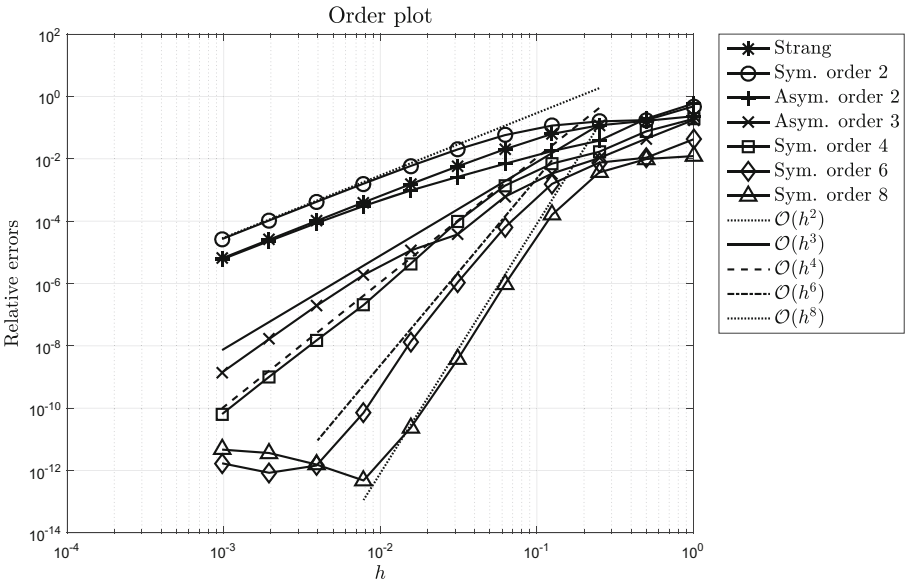

Fig. 1 Errors plotted against step sizes for the problem defined in Section 5.1. We observe that all the methods exhibit the expected convergence orders until the round-off level is reached, except for very large step sizes

$A \in \mathbb{R}^{N\times N}$, $B \in \mathbb{R}^{N\times 7}$, and $C \in \mathbb{R}^{6\times N}$ from which we construct $Q = C^T C$ and $S = BR^{-1}B^T$, with $R^{-1} = I$. The problem also involves a mass matrix, i.e., the state equation is $M\dot{x} = Ax + Bu$. We handle this without inverting M by straightforward modifications to the code as in [34]. Additionally, due to a scaling of the problem, a simulation time step of 1 s corresponds to a real time step of 10^{-2} s. To avoid confusion, we work with the simulation time throughout and, therefore, use a final time $T = 4500$.

The exact solution to the problem is unavailable, and since the other currently existing methods are limited to low orders, it is infeasible to use these to compute a sufficiently accurate reference approximation. Instead, we use the eighth-order symmetric splitting scheme itself for this, but with a step size half as large as the smallest step size for the actual approximations. In this experiment, we do employ parallelization through use of MATLAB's `parfor` command, using eight cores on a cluster built out of Intel 2650v3 CPUs. We restrict ourselves to the Strang splitting and the symmetric methods, since our tests indicate that these are typically more efficient than the asymmetric methods. We perform two tests, one with $N = 371$ and one with $N = 1357$. Except the time step size, the only varying parameter is the relative tolerance for computing the matrix exponential actions. In the smaller example, this is set to 10^{-3} for the Strang splitting and 10^{-3}, 10^{-6}, 10^{-8}, and 10^{-8} for the additive schemes of orders 2, 4, 6, and 8, respectively. In the larger example, we take instead 10^{-3}, 10^{-3}, 10^{-5}, 10^{-6}, and 10^{-6}, respectively. The column compression tolerance is in all cases set to $N\epsilon$, where ϵ is the machine epsilon.

Figure 2 shows the results, with $N = 371$ on the left and $N = 1357$ on the right. In the smaller example, we observe that the second-order methods behave as expected, while the higher-order methods only achieve their respective orders for small step sizes. The fact that the errors level out at around 10^{-11} can be avoided by computing the matrix exponentials more accurately, but at additional cost. In the larger example, the situation is slightly worse in that neither of the higher-order methods reaches their asymptotic regimes with the used step sizes. This issue may be due to a lack of regularity in the solution to the exact problem. As in the smaller example, we could eliminate the leveling out of the error by decreasing the tolerance for the matrix exponential actions. However, as the computation times required for these small errors are already rather long, we do not do this. In spite of these issues, we note that the higher-order methods still produce much smaller errors for all step sizes except the largest.

Also included in Fig. 2 are the corresponding errors for the second-order Rosenbrock method proposed in [6]. These computations were done using the MATLAB software M-MESS 1.0.1 [29], which implements the improved LDL^T formulation given in [24]. We choose the parameters suggested in the example code for the steel cooling problem. For the smaller problem, we observe clear second-order convergence with errors of comparable size to the splitting schemes. The situation is similar in the larger problem, except that the error evens out for small step sizes, likely due to inner iterations not being computed accurately enough.

Finally, we note that the column compression tolerance has been chosen rather small. This is required for the small step sizes, due to the small errors produced by the high-order methods. In Fig. 3, we demonstrate the effect of increasing this tolerance. We note that the convergence behavior is unaffected until the truncation level is reached. Obviously, the ranks of the approximations are heavily affected by changing this tolerance. Table 1 illustrates this, by tabulating the rank of the Strang splitting approximation at the final time when $N = 371$. The ranks of the other methods differ (at most) by $\pm$ 10 from these values for the two largest step sizes and by $\pm$ 3 for the other step sizes. In all cases, the rank increases monotonically until the

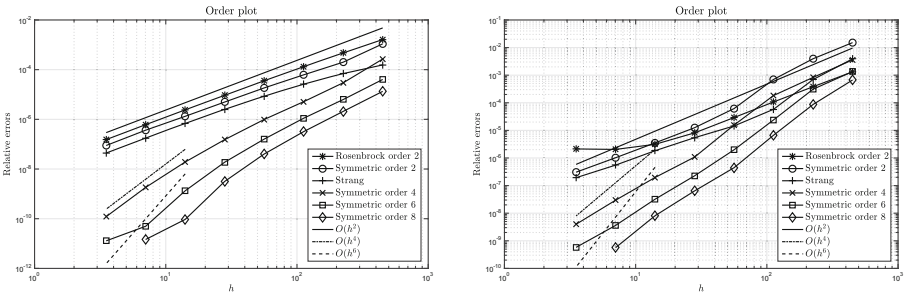

Fig. 2 Errors plotted against step sizes for the problem defined in Section 5.2. Left: $N = 371$. Right: $N = 1357$. We observe that the second-order methods show second-order behavior for all step sizes used, while the higher-order methods suffer from order reduction. For the smaller problem size, we recapture the higher-order behavior for the smallest step sizes, while the larger problem size requires even smaller step sizes before this happens. Regardless of this, the errors of the higher-order methods are significantly smaller than those of the second-order methods

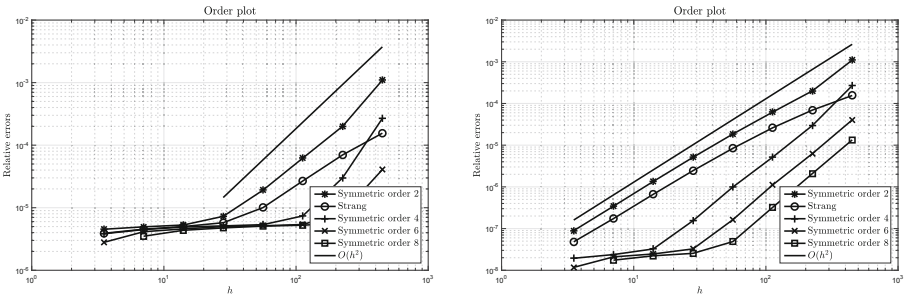

Fig. 3 Errors plotted against step sizes for the problem defined in Section 5.2 with $N = 371$ and the column compression tolerances 10^{-8} (left) and 10^{-10} (right). We note that the convergence is unaffected until the truncation level is reached. Because these errors are introduced in each time step, the error levels out at a value larger than the specified tolerance

final time, i.e., the presented ranks are the maximum attained during the simulation. For comparison, the rank of the corresponding ARE (which the DRE solution tends to as $t \to \infty$) is 138. When $N = 1357$, the ranks of the approximations are of similar size, which fits well with the expectation that low rank is a property inherent to the DRE, independent of the discretizations.

5.3 Efficiency

While the higher-order methods produce smaller errors, this is only relevant if their computational costs are similar to that of the lower-order methods. We therefore also provide a rough comparison of the efficiency of the different methods. Figure 4 shows the errors plotted against the required computation time (wall clock time) for the small-scale problem given in Section 5.1, with the same method parameters. These are the same errors as in Fig. 1, i.e., the step size h is the only varying parameter. We observe that all the methods are roughly equivalent for high tolerances, while for error levels below 10^{-4}, the symmetric methods outperform the others. For very small errors, the sixth- and eighth-order methods are clearly superior. This is in spite of the fact that parallelization was not used in this case (since the extra time spent on transferring of data was much larger than the actual computation time).

Table 1 Approximation ranks

Tolerance \ No. of steps	10	20	40	80	160	320	640	1280
10^{-8}	68	70	71	70	68	67	66	65
10^{-10}	83	86	86	86	85	84	84	82
$8.2 \cdot 10^{-14}$	102	107	109	110	110	109	107	107

The ranks of the Strang splitting approximation at the final time when $N = 371$, for different column compression tolerances and different numbers of time steps. The last value $8.2 \cdot 10^{-14}$ is equal to $N\epsilon$, where ϵ is the machine epsilon

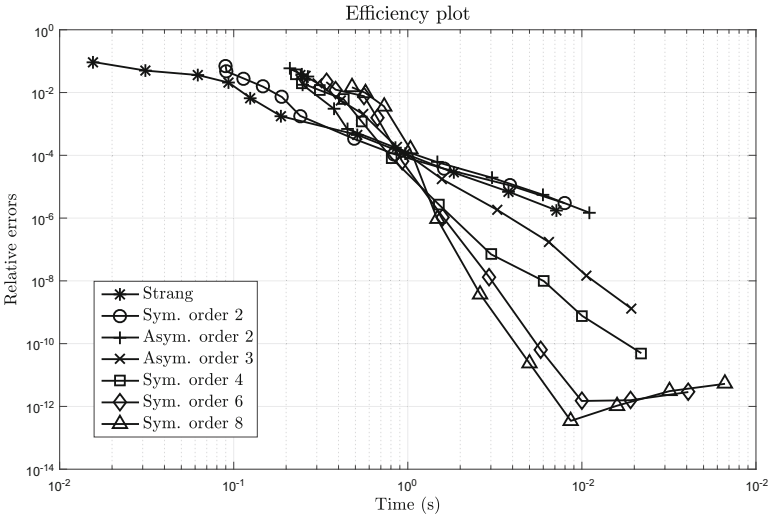

Fig. 4 Errors plotted against computation times for the problem defined in Section 5.1. We see that the lower-order methods are most efficient for high error levels, while the higher-order methods are most efficient for low error levels. For errors around 10^{-4}, the efficiency of all the methods is comparable

The results for the steel cooling problem are shown in Fig. 5. These are similar to the small-scale case in that the higher-order methods are more efficient for small errors while the Strang splitting is most efficient for large errors. The plot is slightly misleading, because the low matrix exponential tolerances required for the high-order methods to reach the smallest errors are not strictly required for the less accurate approximations. Similarly, the lower-order methods would need to compute the matrix exponentials more accurately when the step size is further decreased. Thus, the real cutoff point where the higher-order methods become more efficient lies somewhere between the error levels 10^{-5} and 10^{-7}. We also observe that the

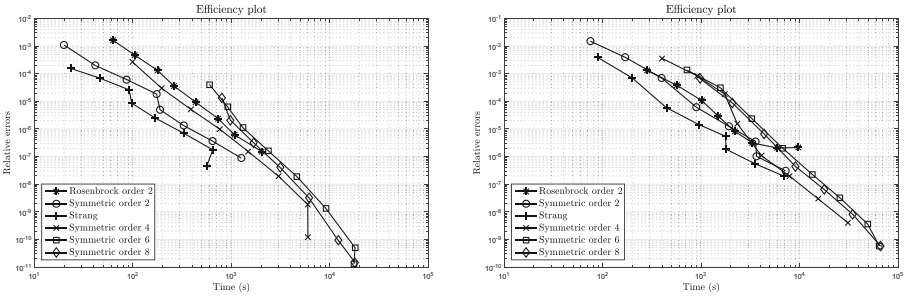

Fig. 5 Errors plotted against computation times for the problem defined in Section 5.2. Left: $N = 371$. Right: $N = 1357$. The lower-order methods are again most efficient for high error levels and vice versa, though the difference between the methods is much less than in Fig. 4

eighth-order method is superior to the sixth-order method for small errors. This is due to the parallelization: the cost of increasing the order by 2 is equivalent to only one extra Lie splitting step and one extra processor. Using even higher orders may thus be beneficial, but eventually the overhead costs incurred by the parallelization will dominate.

The strange kinks in the error curves require an explanation. For the Strang splitting, this happens twice when $N = 371$, and on the latter occurrence, the computation time even decreases slightly when the step size is decreased. This happens due to the way we compute the actions of the matrix exponentials: if the requested accuracy is not reached, the computation is repeated with twice as many sub-steps. Reducing the time step by a factor of 2 makes this computation easier, and it may thus be that most of these computations require only half as many sub-steps as for the larger time step. With twice as many time steps, the total computation time is therefore roughly unchanged.

To provide an indication of what parts of the splitting schemes are expensive, all the methods were run through MATLAB's `profile` command while solving the steel cooling problem with $N = 1357$ and with either 40 or 640 time steps, corresponding to $h = 112.5$ or $h = 7.0313$. The tolerance for the $\mathrm{e}^{\gamma h A^T} L$ computations was set to 10^{-6} in all cases. The results are shown in Tables 2 and 3. We observe that, as expected, the evaluations of $\mathcal{T}_\mathcal{G}$ are essentially free in comparison to $\mathcal{T}_\mathcal{F}$. The cost of the latter completely dominates the overall procedure. Along with the observation in the previous paragraph, this provides additional incentive for studying better implementation strategies for this basic operation.

Further, we observe that the relative cost of column compression increases with the order of the method, since $L_\pm^j$ and $D_\pm^j$ increase in size. In total, however, this cost is negligible, despite the fact that the ranks of the approximations are not very small. It should be noted here that due to the difficulties of accurately timing parallel code in this level of detail, a serial implementation was used. While this skews the ratios, the effect is very small because almost all column compressions originate from $\mathcal{T}_\mathcal{F}$

Table 2 Computational time breakdown – large time steps

Operation \ Method	Strang	Additive 2	Additive 4	Additive 6	Additive 8
$\mathcal{T}_\mathcal{G}$	0.01	0.03	0.03	0.03	0.03
$\mathcal{T}_\mathcal{F}$	96.60	95.90	97.50	98.06	98.32
Column compression	0.16	0.31	0.33	0.36	0.38
$\mathrm{e}^{\gamma h A^T} L$	99.10	98.88	99.16	99.16	99.18
$I_Q(h)$	3.38	3.92	2.30	1.71	1.42

Computational time breakdown for the splitting schemes when applied to the steel cooling problem with $N = 1357$ and 40 time steps. Shown is the time spent on the given operation, divided by the total time for the integration (in percent). The numbers are not independent, e.g., computing $I_Q(h)$ requires several $\mathrm{e}^{\gamma h A^T} L$ evaluations and column compressions

Table 3 Computational time breakdown – small time steps

Operation \ Method	Strang	Additive 2	Additive 4	Additive 6	Additive 8
$\mathcal{T}_{\mathcal{G}}$	0.02	0.03	0.03	0.03	0.03
$\mathcal{T}_{\mathcal{F}}$	99.86	99.66	99.68	99.68	99.67
Column compression	0.13	0.31	0.33	0.34	0.37
$\mathrm{e}^{\gamma h A^T} L$	99.51	99.33	99.30	99.30	99.27
$I_Q(h)$	0.12	0.13	0.08	0.05	0.04

Computational time breakdown for the splitting schemes when applied to the steel cooling problem with $N = 1357$ and 640 time steps. Shown is the time spent on the given operation, divided by the total time for the integration (in percent). The numbers are not independent, e.g., computing $I_Q(h)$ requires several $\mathrm{e}^{\gamma h A^T} L$ evaluations and column compressions

evaluations (95% for the order 8 method and the smallest step size). As expected, the relative cost of the one-time computation of $I_Q(h)$ is higher for a small number of time steps, but even in the worst case, it is measured in single-digit percentages. With many time steps, the relative cost is negligible.

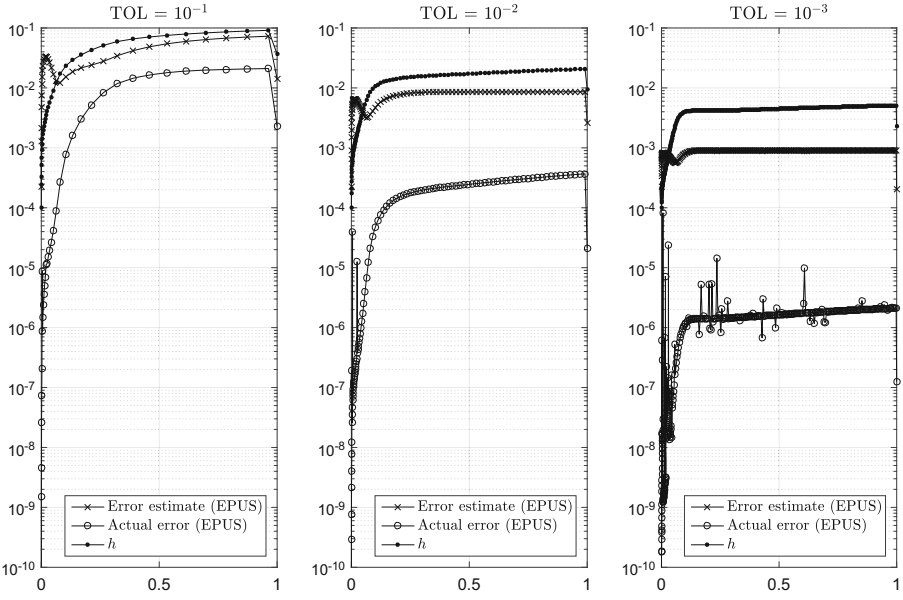

Fig. 6 Computed error estimate, actual error, and step size for each time step, when applying the adaptive fourth-order symmetric splitting scheme to the problem defined in Section 5.1. We consider error per unit step (EPUS), i.e., all errors are divided by the time step. The tolerances used are, from left to right, 10^{-1}, 10^{-2}, and 10^{-3}. We observe that the adaptivity finds the maximum step size such that the error estimate is equal to the tolerance. Due to the lower-order estimate, the actual error is in all cases less than the estimated error, and the difference increases as the step size decreases. The sudden drop in step size (and error) in the final step is necessary in order to exactly reach the final time T

Finally, we note that Fig. 5, like Fig. 2, also includes the results for the second-order Rosenbrock method. While a fair comparison is difficult, and many parameters could be further fine-tuned for all the methods, these results clearly indicate that the splitting schemes constitute a competitive alternative to this class of methods.

5.4 Time adaptivity

Finally, we test the full time step adaptive code with the fourth-order symmetric splitting scheme. In Fig. 6, we have plotted the results of using four different tolerances on the small-scale problem defined in Section 5.1. We plot both the error estimated by the method using the embedded method and the actual error. The latter is computed by using the same method, but by taking 10 equidistant steps in each of the steps given by the adaptive code. We observe that the actual error is in all cases less than the estimated error, and the difference increases as the tolerance decreases. This is due to the fact that the error estimate is of a lower order than the actual method used. The effect is more pronounced here than usual, since in the symmetric case, the accuracy of the estimate is two orders less than the method. In each figure, we have also plotted the step sizes, and we see that the controller works well in finding the maximum possible step size. For the largest tolerance, the controller is too cautious and does not quite reach the tolerance until the simulation is over. Effects like

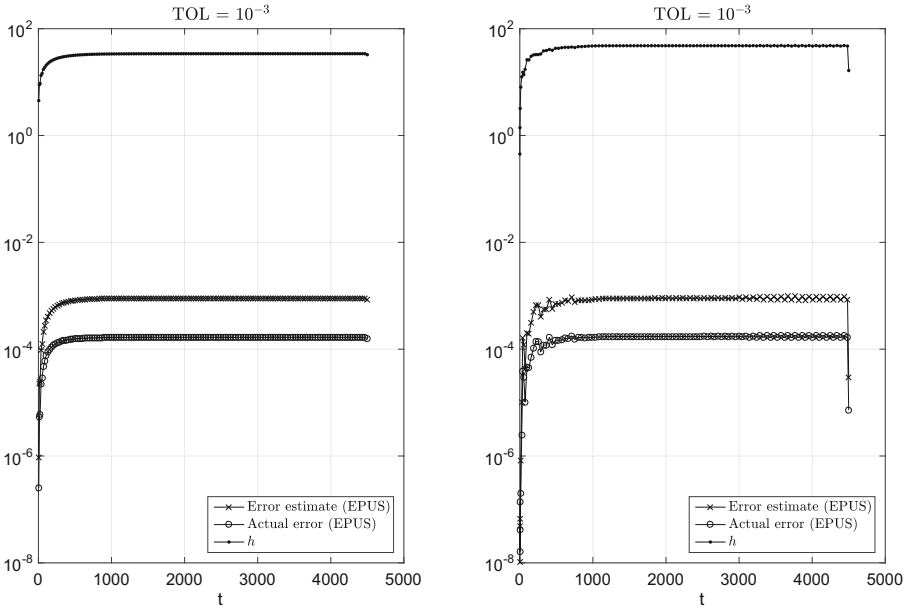

Fig. 7 Computed error estimate, actual error, and step size for each time step, when applying the adaptive fourth-order symmetric splitting scheme with tolerance 10^{-3} to the problem defined in Section 5.2. The method was applied either without (left) or with (right) Algorithm 1. We consider error per unit step (EPUS), i.e., all errors are divided by the time step. We observe that the adaptivity works rather well

Table 4 Computational time breakdown – benefit of Algorithm 1

Method \ Operation	Total time	$I_Q(h_n)$	$\mathcal{T}_{\mathcal{F}}$	$\mathrm{e}^{\gamma h A^T} L$	CC
Without Algorithm 1	100.00	46.57	52.42	87.32	1.78
With Algorithm 1	65.90	7.97	89.90	86.90	6.14

Computational time breakdown for the adaptive splitting scheme when applied to the steel cooling experiment in Section 5.4, either with or without the use of Algorithm 1. The first column shows the relative computation times depending on this choice. The other columns show the time spent on the given operation, divided by the total time for the respective method. All numbers are in percent, and "CC" is an abbreviation for column compression. Only the numbers in the last two columns are independent. The remaining computation time was spent on (unoptimized) caching of matrices and general bookkeeping

this can (and should, this is one area we aim to pursue in the near future) be tuned by adjusting the parameters k_I and k_P.

In Fig. 7, we have repeated the same experiment but on the steel problem with $N = 371$ and only with the tolerance 10^{-3}. Also in this case, the adaptiveness seems to work well—the maximum possible step size (given the tolerance) is quickly reached and after this it varies very little. In this case, the difference between the error estimate and the actual error is not as large as in the previous example. This is likely due to the order reductions observed in Section 5.2.

The left plot shows the results when Algorithm 1 is not used and the right plot when it is. In both cases, we used a column compression tolerance of 10^{-8}, a relative tolerance of 10^{-4} for the matrix exponential actions and quadrature of order 9 to compute $I_Q(h_n)$. When not using Algorithm 1, we use Gaussian quadrature rather than Newton-Cotes, and thus, these computations only need four quadrature nodes compared to the updating formula which needs 10. Still, as demonstrated by the computational time breakdown in Table 4, the latter is more efficient because typically only one or even none of these nodes need to be updated in each step. In the current experiment, 111 steps were taken. Of these, 6 were rejected which required all 10 nodes to be updated. During the remaining 105 steps, a total of 10 nodes required an update.

6 Conclusions

We have introduced a family of splitting schemes for differential Riccati equations which may be of arbitrarily high order, and shown that they may be implemented efficiently in a large-scale setting by utilizing the low-rank LDL^T factorization. Our numerical experiments indicate that the higher-order methods are more efficient when high accuracy is desired, though this of course depends on the actual problem. In addition, we have demonstrated that these methods contain natural embedded error estimates, which, e.g., may be used for time step adaptivity. While further research on appropriate controller parameters in this setting is required, experiments show that even a basic implementation gives promising results.

References

1. Abou-Kandil, H., Freiling, G., Ionescu, V., Jank, G.: Matrix Riccati Equations. Systems & Control: Foundations & Applications. Basel, Birkhäuser (2003)
2. Amodei, L., Buchot, J.M.: An invariant subspace method for large-scale algebraic Riccati equation. Appl. Numer. Math **60**(11), 1067–1082 (2010). https://doi.org/10.1016/j.apnum.2009.09.006
3. Antoulas, A.C., Sorensen, D.C., Zhou, Y.: On the decay rate of Hankel singular values and related issues. Syst. Control Lett **46**(5), 323–342 (2002). https://doi.org/10.1016/S0167-6911(02)00147-0
4. Baśar, T., Bernhard, P. H^{∞}-Optimal Control and Related Minimax Design Problems, 2nd edn. Sys. Con. Fdn. Birkhäuser Boston, Inc, Boston (1995). A Dynamic Game Approach
5. Benner, P., Bujanović, Z.: On the solution of large-scale algebraic Riccati equations by using low-dimensional invariant subspaces. Linear Algebra Appl. **488**, 430–459 (2016). https://doi.org/10.1016/j.laa.2015.09.027
6. Benner, P., Mena, H.: Rosenbrock methods for solving Riccati differential equations. IEEE Trans. Automat. Control **58**(11), 2950–2956 (2013). https://doi.org/10.1109/TAC.2013.2258495
7. Benner, P., Mena, H.: Numerical solution of the infinite-dimensional LQR problem and the associated Riccati differential equations. J. Numer. Math. https://doi.org/10.1515/jnma-2016-1039 (in press)
8. Benner, P., Saak, J.: A Semi-Discretized Heat Transfer Model for Optimal Cooling of Steel Profiles. In: Dimension Reduction of Large-Scale Systems, Lecture Notes in Computational Science and Engineering, vol. 45, pp. 353–356. Springer, Berlin (2005)
9. Benner, P., Saak, J.: Numerical solution of large and sparse continuous time algebraic matrix Riccati and Lyapunov equations: a state of the art survey. GAMM-Mitt **36**(1), 32–52 (2013). https://doi.org/10.1002/gamm.201310003
10. Blanes, S., Casas, F.: On the necessity of negative coefficients for operator splitting schemes of order higher than two. Appl. Numer. Math **54**(1), 23–37 (2005). https://doi.org/10.1016/j.apnum.2004.10.005
11. Caliari, M., Kandolf, P., Ostermann, A., Rainer, S.: Comparison of software for computing the action of the matrix exponential. BIT **54**(1), 113–128 (2014). https://doi.org/10.1007/s10543-013-0446-0
12. De Leo, M., Rial, D., de la Vega, C.S.: High-order time-splitting methods for irreversible equations. IMA J. Numer. Anal. **36**(4), 1842–1866 (2016). https://doi.org/10.1093/imanum/drv058
13. Druskin, V., Knizhnerman, L., Simoncini, V.: Analysis of the rational Krylov subspace and ADI methods for solving the Lyapunov equation. SIAM J. Numer. Anal. **49**(5), 1875–1898 (2011). https://doi.org/10.1137/100813257
14. Güldoğan, Y., Hached, M., Jbilou, K., Kurulay, M.: Low rank approximate solutions to large-scale differential matrix Riccati equations. arXiv:1612.00499v2[math.NA] (2017)
15. Gustafsson, K.: Control-theoretic techniques for stepsize selection in explicit Runge-Kutta methods. ACM Trans. Math. Softw. **17**(4), 533–554 (1991). https://doi.org/10.1145/210232.210242
16. Gustafsson, K., Lundh, M., Söderlind, G.: A PI stepsize control for the numerical solution of ordinary differential equations. BIT **28**(2), 270–287 (1988). https://doi.org/10.1007/BF01934091
17. Hager, W.W.: Updating the inverse of a matrix. SIAM Rev. **31**(2), 221–239 (1989)
18. Hairer, E., Wanner, G. Solving Ordinary Differential Equations. II, Springer Series in Computational Mathematics, 2nd edn., vol. 14. Springer, Berlin (1996)
19. Hansen, E., Ostermann, A.: High order splitting methods for analytic semigroups exist. BIT **49**(3), 527–542 (2009). https://doi.org/10.1007/s10543-009-0236-x
20. Heyouni, M., Jbilou, K.: An extended block Arnoldi algorithm for large-scale solutions of the continuous-time algebraic Riccati equation. Electron. Trans. Numer. Anal. **33**, 53–62 (2008/09)
21. Hundsdorfer, W., Verwer, J.: Numerical Solution of Time-Dependent Advection-Diffusion-Reaction Equations. Springer Series in Computational Mathematics, vol. 33. Springer, Berlin (2003)
22. Ichikawa, A., Katayama, H.: Remarks on the time-varying H_{∞} Riccati equations. Syst. Control Lett. **37**(5), 335–345 (1999)

23. Koskela, A., Mena, H.: A Structure Preserving Krylov Subspace Method for Large Scale Differential Riccati Equations. arXiv:1705.07507v1[math.NA] (2017)
24. Lang, N., Mena, H., Saak, J.: On the benefits of the LDL^T factorization for large-scale differential matrix equation solvers. Linear Algebra Appl. **480**, 44–71 (2015). https://doi.org/10.1016/j.laa.2015.04.006
25. Lin, Y., Simoncini, V.: A new subspace iteration method for the algebraic Riccati equation. Numer. Linear Algebra Appl. **22**(1), 26–47 (2015). https://doi.org/10.1002/nla.1936
26. Mena, H., Ostermann, A., Pfurtscheller, L., Piazzola, C.: Numerical low-rank approximation of matrix differential equations. arXiv:1705.10175 (2017)
27. Petersen, I.R., Ugrinovskii, V.A., Savkin, A.V.: Robust Control Design Using H^∞ Methods. Springer, London (2000)
28. Saak, J.: Effiziente numerische Lösung eines Optimalsteuerungsproblems für die Abkühlung von Stahlprofilen. Master's Thesis, Fachbereich 3/Mathematik und Informatik, Universität Bremen (2003)
29. Saak, J., Köhler, M., Benner, P.: M-M.E.S.S.-1.0.1—the matrix equations sparse solvers library. https://doi.org/10.5281/zenodo.50575. See also: www.mpi-magdeburg.mpg.de/projects/mess (2016)
30. Simoncini, V.: Computational methods for linear matrix equations. SIAM Rev. **58**(3), 377–441 (2016). https://doi.org/10.1137/130912839
31. Simoncini, V., Szyld, D.B., Monsalve, M.: On two numerical methods for the solution of large-scale algebraic Riccati equations. IMA J. Numer. Anal **34**(3), 904–920 (2014). https://doi.org/10.1093/imanum/drt015
32. Söderlind, G.: Automatic control and adaptive time-stepping. Numer. Algorithms **31**(1-4), 281–310 (2002). https://doi.org/10.1023/A:1021160023092. Numerical methods for ordinary differential equations (Auckland, 2001)
33. Sorensen, D.C., Zhou, Y.: Bounds on eigenvalue decay rates and sensitivity of solutions to Lyapunov equations. Tech. Rep 02-07, Dept. of Comp. Appl. Math., Rice Univ., Houston. http://www.caam.rice.edu/caam/trs/tr02.html#TR02-07 (2002)
34. Stillfjord, T.: Low-rank second-order splitting of large-scale differential Riccati equations. IEEE Trans. Automat. Control **60**(10), 2791–2796 (2015). https://doi.org/10.1109/TAC.2015.2398889