Implementation of exponential Rosenbrock-type integrators

Marco Caliari [a,b] , Alexander Ostermann [b,*]

[a] *Department of Pure and Applied Mathematics, University of Padua, Via Trieste 63, I-35121 Padova, Italy*

[b] *Department of Mathematics, University of Innsbruck, Technikerstraße 13, A-6020 Innsbruck, Austria*

Abstract

In this paper, we present a variable step size implementation of exponential Rosenbrock-type methods of orders 2, 3 and 4. These integrators require the evaluation of exponential and related functions of the Jacobian matrix. To this aim, the Real Leja Points Method is used. It is shown that the properties of this method combine well with the particular requirements of Rosenbrock-type integrators. We verify our implementation with some numerical experiments in Matlab, where we solve semilinear parabolic PDEs in one and two space dimensions. We further present some numerical experiments in Fortran, where we compare our method with other methods from literature. We find a great potential of our method for non-normal matrices. Such matrices typically arise in parabolic problems with large advection in combination with moderate diffusion and mildly stiff reactions.

Key words: exponential integrators, Rosenbrock-type methods, real Leja points, Newton interpolation, parabolic evolution equations, implementation

1 Introduction

Exponential integrators have a long history in numerical analysis. First prototypes have been constructed already more than 50 years ago. Nevertheless, they did not play a prominent role in applications for quite a long time. The

* Corresponding author.
Email addresses: mcaliari@math.unipd.it (Marco Caliari), alexander.ostermann@uibk.ac.at (Alexander Ostermann).

main problem was that exponential integrators make explicit use of the exponential and related functions of (large) matrices. Their efficient implementation was therefore considered to be difficult or even impossible.

This view changed in the last years when very promising experiments with semilinear evolution equations where reported in literature. These experiments gave on the one hand the impetus for a rigour error and convergence analysis for stiff evolution equations, see [6, 11]. On the other hand they led to the construction of several new classes of exponential integrators, see, e.g., [13, 16].

In this paper we are concerned with exponential Rosenbrock-type methods that rely on the full Jacobian of the problem [13]. In general, this matrix has no particular structure, and it changes continuously from step to step. Therefore, standard techniques for evaluating the exponential are not very efficient. In particular, methods based on fast Fourier techniques can not be used. Recent progress in numerical linear algebra suggests Krylov subspace methods and methods based on direct polynomial interpolation as attractive alternatives. Our approach here is based on the latter.

For a given function φ, a square matrix J and a vector v, we approximate $\varphi(J)v$ by an interpolation polynomial, based on real Leja points. This interpolation procedure was first proposed in [5] in the context of exponential integrators. It is computationally attractive because it has modest storage requirements and involves matrix-vector multiplications only. We recall that Leja points are roughly distributed like Chebyshev points. This reduces the sensitivity of the interpolation polynomial with respect to perturbations. Moreover, the recursive definition of Leja points is an attractive feature in combination with Newton-type interpolation.

The main purpose of the paper is to show that exponential Rosenbrock-type methods can be implemented efficiently in that way, and that the resulting programs are competitive with existing methods for solving certain reaction-diffusion-advection equations.

An outline of the paper is as follows: In Section 2, we state our problem class and recall the definition of exponential Rosenbrock-type methods from [13]. In Section 3, we give two embedded pairs of methods of orders 3 and 4, respectively, and we provide the exponential Rosenbrock–Euler scheme with an appropriate error estimate. In Section 4 we introduce the *Real Leja Points Method* for approximating the exponential and related functions, and we explain some issues of implementation. Section 5 is devoted to numerical experiments in MATLAB and FORTRAN. Finally, we draw some conclusions in Section 6.

2 Problem class and numerical methods

In this paper we are concerned with the time discretisation of autonomous evolution equations of the form

$$u'(t) = f(u(t)) = Au(t) + g(u(t)), \qquad u(t_0) = u_0. \tag{2.1}$$

A possible extension to non-autonomous problems will be discussed at the end of this section. Equation (2.1) will be considered as an abstract differential equation in a Banach space framework of sectorial operators. In particular, we assume that the operator A generates an analytic semigroup and that the nonlinearity g is relatively bounded with respect to A, see [9] for details. This framework is sufficiently general to cover interesting examples such as reaction-diffusion-advection equations.

For the solution of (2.1), we consider exponential one-step methods which define numerical approximations u_n to the exact solution $u(t_n)$ at discrete times t_n. The methods rely on a *linearisation* of (2.1) at each step

$$u'(t) = J_n u(t) + g_n(u(t)), \qquad t_n \le t \le t_{n+1}, \tag{2.2a}$$

where J_n denotes the Fréchet derivative of f and g_n the remainder

$$J_n = \mathrm{D}_u f(u_n), \qquad g_n(u(t)) = f(u(t)) - J_n u(t). \tag{2.2b}$$

Applying the exponential Euler method with step size $h_n = t_{n+1} - t_n$ to problem (2.2) results in the numerical scheme

$$\begin{aligned} u_{n+1} &= \exp(h_n J_n) u_n + h_n \varphi_1(h_n J_n) g_n(u_n) \\ &= u_n + h_n \varphi_1(h_n J_n) f(u_n), \end{aligned} \tag{2.3}$$

which we will call *exponential Rosenbrock–Euler method* henceforth. Recall that exponential integrators are built on the exponential function and on the related entire functions

$$\varphi_k(z) = \int_0^1 \mathrm{e}^{(1-\tau)z} \frac{\tau^{j-1}}{(j-1)!}\, \mathrm{d}\tau, \qquad k \ge 1. \tag{2.4}$$

This integral representation shows that the φ-functions are well-defined for sectorial operators as arguments as well.

More generally, we consider in this paper the class of s-stage exponential

Rosenbrock-type methods

$$U_{ni} = \exp(c_i h_n J_n) u_n + h_n \sum_{j=1}^{i-1} a_{ij}(h_n J_n) g_n(U_{nj}), \tag{2.5a}$$

$$u_{n+1} = \exp(h_n J_n) u_n + h_n \sum_{i=1}^{s} b_i(h_n J_n) g_n(U_{ni}). \tag{2.5b}$$

These methods were introduced and fully analysed in [12, 13]. The exponential Rosenbrock–Euler method is a particular case with one stage, and with weight $b_1(hJ) = \varphi_1(hJ)$. We always employ the simplifying assumptions

$$\sum_{i=1}^{s} b_i(hJ) = \varphi_1(hJ), \qquad \sum_{j=1}^{i-1} a_{ij}(hJ) = c_i \varphi_1(c_i hJ), \quad i = 1, \ldots, s \tag{2.6}$$

which can be seen as natural extensions of the well-known B(1) and C(1) conditions for Runge–Kutta methods. As a consequence, all methods possess the node $c_1 = 0$.

Remark. We briefly sketch how exponential Rosenbrock-type methods can be applied to non-autonomous problems

$$u'(t) = f(t, u(t)), \qquad u(t_0) = u_0. \tag{2.7}$$

The key step is to rewrite (2.7) in autonomous form by adding the equation $t' = 1$, and to linearise at (t_n, u_n). This yields

$$\begin{bmatrix} t' \\ u'(t) \end{bmatrix} = \begin{bmatrix} 0 & 0 \\ \mathrm{D}_t f(t_n, u_n) & \mathrm{D}_u f(t_n, u_n) \end{bmatrix} \begin{bmatrix} t \\ u(t) \end{bmatrix} + \begin{bmatrix} 1 \\ g_n(t, u(t)) \end{bmatrix} \tag{2.8}$$

with

$$g_n(t, u(t)) = f(t, u(t)) - \mathrm{D}_t f(t_n, u_n)\, t - \mathrm{D}_u f(t_n, u_n)\, u(t).$$

The exponential Rosenbrock method (2.5) is then applied to this augmented system. For an efficient implementation, one uses the following result.

Lemma 1 *Let J be a real $N \times N$ matrix and w a vector in $\mathbb{R}^N$. Then*

$$\varphi_k\Bigl(\begin{bmatrix} 0 & 0 \\ w & J \end{bmatrix}\Bigr) = \begin{bmatrix} \varphi_k(0) & 0 \\ \varphi_{k+1}(J) w & \varphi_k(J) \end{bmatrix}. \tag{2.9}$$

Proof. From (2.4) we see that

$$\varphi_k\Bigl(\begin{bmatrix} 0 & 0 \\ w & J \end{bmatrix}\Bigr) \begin{bmatrix} \alpha \\ \beta \end{bmatrix}$$

is the solution of the problem

$$\begin{aligned} x'(t) &= \frac{t^{k-1}}{(k-1)!}\,\alpha, & x(0) &= 0, \\ y'(t) &= w\,x(t) + Jy(t) + \frac{t^{k-1}}{(k-1)!}\,\beta, & y(0) &= 0, \end{aligned} \tag{2.10}$$

at $t = 1$. From

$$x(t) = \frac{t^k}{k!}\,\alpha, \qquad x(1) = \varphi_k(0)\,\alpha,$$

we obtain

$$y(1) = \varphi_{k+1}(J)w\alpha + \varphi_k(J)\beta$$

by using once more (2.4). This finally proves the result. □

3 Embedded methods of orders two, three and four

The construction of high-order integrators is usually based on order conditions. We are mainly interested here in the stiff case, requiring our methods to converge uniformly with respect to the stiffness. For exponential Rosenbrock-type methods, the corresponding order conditions were derived in [12, 13], where also embedded pairs of orders 3 and 4 were constructed. We summarise the order conditions in Table 1, and we recall some simple methods below.

Since we require exponential Rosenbrock-type methods to satisfy the simplifying assumptions (2.6), all considered methods are of order two at least.

Order 2. The exponential Rosenbrock–Euler method (2.3) is computationally attractive since it achieves second order with one stage only. For the purpose of local error estimation, we consider also

$$\widetilde{u}_{n+1} = \exp(h_n J_n)u_n + h_n\varphi_1(h_n J_n)g_n(u_{n+1}). \tag{3.1}$$

Table 1
Order conditions for exponential Rosenbrock-type methods up to order 4.

order	order condition
1	$\sum_{i=1}^{s} b_i(hJ) = \varphi_1(hJ)$
2	$\sum_{j=1}^{i-1} a_{ij}(hJ) = c_i\varphi_1(c_i hJ), \quad 2 \le i \le s$
3	$\sum_{i=2}^{s} b_i(hJ)c_i^2 = 2\varphi_3(hJ)$
4	$\sum_{i=2}^{s} b_i(hJ)c_i^3 = 6\varphi_4(hJ)$

The dominating term in the error control

$$\begin{aligned} \widetilde{u}_{n+1} - u_{n+1} &= h_n \varphi_1(h_n J_n)\Big(g_n(u_{n+1}) - g_n(u_n)\Big) \\ &= h_n \varphi_1(h_n J_n)\Big(\frac{1}{2}\frac{\partial^2 f}{\partial u^2}(u_n)(u_{n+1}-u_n)^2 + \ldots\Big) \end{aligned} \tag{3.2}$$

is of third order (under reasonable smoothness assumptions). The exponential Rosenbrock–Euler method with the above error estimate will be called `erow2` henceforth.

Order 3. With two stages, it is possible to construct a third-order method with coefficients

$$\begin{array}{c|cc} c_1 & & \\ c_2 & a_{21} & \\ \hline & b_1 & b_2 \end{array} \quad = \quad \begin{array}{c|cc} 0 & & \\ 1 & \varphi_1 & \\ \hline & \varphi_1 - 2\varphi_3 & 2\varphi_3 \end{array}$$

For error control, we employ the exponential Rosenbrock–Euler method. The resulting embedded pair will be called `erow32` henceforth.

Order 4. The method `erow43` is a fourth-order method with a third-order error estimator. Its coefficients are

$$\begin{array}{c|ccc} c_1 & & & \\ c_2 & a_{21} & & \\ c_3 & a_{31} & a_{32} & \\ \hline & b_1 & b_2 & b_3 \\ & \widetilde{b}_1 & \widetilde{b}_2 & \widetilde{b}_3 \end{array} \quad = \quad \begin{array}{c|ccc} 0 & & & \\ \frac{1}{2} & \frac{1}{2}\varphi_1\big(\frac{1}{2}\cdot\big) & & \\ 1 & 0 & \varphi_1 & \\ \hline & \varphi_1 - 14\varphi_3 + 36\varphi_4 & 16\varphi_3 - 48\varphi_4 & -2\varphi_3 + 12\varphi_4 \\ & \varphi_1 - 14\varphi_3 & 16\varphi_3 & -2\varphi_3 \end{array}$$

where $\widetilde{b}_i$ are the corresponding weights of the embedded method

$$\widetilde{u}_{n+1} = \exp(h_n J_n)u_n + h_n \sum_{i=1}^{s} \widetilde{b}_i(h_n J_n) g_n(U_{ni}). \tag{3.3}$$

Note that the internal stages of all considered methods are exponential Rosenbrock–Euler steps.

A competitive implementation of exponential integrators strongly relies on an efficient computation of the involved φ-functions. Our approach, to be described in the next section, is based on direct interpolation.

4 Newton interpolation at Leja points

Let J be a real $N \times N$ matrix, $h > 0$ and v a vector in $\mathbb{R}^N$. In order to compute $\varphi_k(hJ)v$, we use the *Real Leja Points Method*, first proposed in [5] for φ_1. This

method has shown very attractive computational features. It is based on the interpolation in the Newton form of the scalar function $\varphi_k(hz)$ at a sequence of Leja points $\{z_i\}$ on a real focal interval, say $[a, b]$, of a family of confocal ellipses in the complex plane.

A sequence of Leja points $\{z_i\}$ is defined recursively, usually starting from $|z_0| = \max\{|a|, |b|\}$, in such a way that the $(m+1)$-th point satisfies

$$\prod_{i=0}^{m-1} |z_m - z_i| = \max_{z\in[a,b]} \prod_{i=0}^{m-1} |z - z_i|.$$

In practice, Leja points can be extracted from a sufficiently dense set of uniformly distributed points on $[a, b]$. From the definition, it is clear that it is possible to increase the interpolation degree just by adding new nodes of the same sequence.

The use of Leja points is suggested, besides the optimal properties in the Newton interpolation form (cf. [17]), by the fact that they guarantee maximal and superlinear convergence of the interpolant on every ellipse of the confocal family. Therefore, they also guarantee superlinear convergence of the corresponding matrix polynomials, provided that the spectrum (or the field of values) of the matrix in contained in one of the above ellipses.

A key step in the approximation procedure consists in estimating cheaply the real focal interval $[a, b]$ such that the "minimal" ellipse of the confocal family containing the spectrum has a moderate *capacity*. We recall that the capacity of an ellipse is the half sum of its semi-axes. The numerical experience with matrices arising from stable spatial discretisations of parabolic equations, which are the main target of the method, has shown that good results can be obtained at a very low cost simply by intersecting the Gerschgorin's disks of the matrix with the real axis.

In order to avoid underflow or overflow problems with the computation of divided differences (cf. [20]), it is convenient to interpolate the scaled and shifted function $\varphi_k(h(c+\gamma\xi))$, $[a, b] = [c-2\gamma, c+2\gamma]$ at the Leja points $\{\xi_i\}$, $\xi_0 = 2$, of the reference interval $[-2, 2]$. The latter can be computed once and for all. The matrix Newton polynomial of degree m is then

$$\begin{aligned} p_m(hJ)v &= p_{m-1}(hJ)v + d_m q_m \,, \\ q_m &= \Big((J - cI)/\gamma - \xi_{m-1} I\Big) q_{m-1}, \end{aligned} \tag{4.1a}$$

where

$$p_0(hJ)v = d_0 q_0, \qquad q_0 = v, \tag{4.1b}$$

and $\{d_i\}_{i=0}^m$ are the divided differences of the function $\varphi_k(h(c+\gamma\xi))$ at the points $\{\xi_i\}$. Notice that in the practical implementation of (4.1) it is sufficient to use a vector $q = q_{m-1}$ and update it at each iteration. The degree of

approximation is not known a priori and depends on the accuracy required for the approximation of $\varphi_k(hJ)v$. A practical estimate of the interpolation error is given by

$$\begin{aligned} \|e_m\| = \|p_{m+1}(hJ)v - p_m(hJ)v\| = |d_{m+1}| \cdot \|q_{m+1}\| \approx \\ \approx \|p_m(hJ)v - \varphi_k(hJ)v\|. \end{aligned} \tag{4.2}$$

In order to filter possible oscillations in the error estimate, the average on the last five values $\|e_m\|, \ldots, \|e_{m-4}\|$ is used instead as the error estimate at degree m.

The attractive computational features of the method are clear: there is no Krylov subspace to store and the complexity of the two-term recurrence (4.1a) is linear and not quadratic in m, as the long-term recurrence in the standard Krylov method for nonsymmetric matrices hJ, see, e.g., [7, 10, 3]. Moreover, it is not required to solve real or complex linear systems, as in rational Krylov approximations [15, 21] or Carathéodory–Fejér and contour integrals approximations [18]. Finally, the Real Leja Points Method is very well structured for a parallel implementation, as shown in [2, 14] for the φ_1-function.

When the expected degree m for convergence is too large, the original time step h has to be split into a certain number of substeps, say L and the approximation of $\varphi_k(hJ)v$ is recovered from $\varphi_k(\tau hJ)v$ with $\tau = 1/L$. It is possible to use a variant of the scaling and squaring approach as suggested in [4]. This approach, however, is restricted to not too large matrices, since it requires the explicit computation of some $\varphi_i(hJ)$, $i \leq k$. Here we prefer to consider $\varphi_k(hJ)v$ as the solution of a ODE system at time $t = 1$ and to use an exact integrator for the system. Let $K = hJ$ and consider

$$y'(t) = Ky(t) + \frac{t^{k-1}}{(k-1)!}\, v, \qquad y(0) = 0 \tag{4.3}$$

for $k > 0$. We set $y_0 = 0$ and $\tau_\ell = \ell\tau$. Applying the variation-of-constants formula to this problem, we get

$$\begin{aligned} y(\tau_{\ell+1}) = y_{\ell+1} &= \exp(\tau K)y_\ell + \int_{\tau_\ell}^{\tau_{\ell+1}} \exp((\tau_{\ell+1} - s)K)\frac{s^{k-1}}{(k-1)!} v \,\mathrm{d}s \\ &= \exp(\tau K)y_\ell + \int_0^{\tau} \exp((\tau - \zeta)K)\frac{(\zeta + \tau_\ell)^{k-1}}{(k-1)!} v \,\mathrm{d}\zeta. \end{aligned} \tag{4.4}$$

The last integral can be rewritten as

$$\begin{aligned}
&\int_0^\tau \exp((\tau-\zeta)K)\frac{(\zeta+\tau_\ell)^{k-1}}{(k-1)!}v\,\mathrm{d}\zeta \\
&\qquad = \sum_{i=0}^{k-1}\binom{k-1}{i}\int_0^\tau \exp((\tau-\zeta)K)\frac{\zeta^i\tau_\ell^{k-1-i}}{(k-1)!}v\,\mathrm{d}\zeta \\
&\qquad = \sum_{i=0}^{k-1}\frac{(k-1)!}{(k-1-i)!(k-1)!}\int_0^\tau \exp((\tau-\zeta)K)\frac{\zeta^i\tau_\ell^{k-1-i}}{i!}v\,\mathrm{d}\zeta \\
&\qquad = \sum_{i=0}^{k-1}\frac{\tau_\ell^{k-1-i}\tau^{i+1}}{(k-1-i)!}\varphi_{i+1}(\tau K)v,
\end{aligned} \tag{4.5}$$

where the last equality holds since

$$\tau^k\varphi_k(\tau z) = \int_0^\tau \mathrm{e}^{(\tau-\zeta)z}\frac{\zeta^{k-1}}{(k-1)!}\mathrm{d}\zeta.$$

Inserting (4.5) into (4.4) finally gives

$$y_{\ell+1} = \exp(\tau K)y_\ell + \tau^k\sum_{i=0}^{k-1}\frac{\ell^{k-1-i}}{(k-1-i)!}\varphi_{i+1}(\tau K)v, \quad \ell = 0,\dots,L-1. \tag{4.6}$$

The computation of (4.6) requires k evaluations of a matrix function for $\ell = 0$. When increasing ℓ by one, only one additional evaluation, namely that of $\exp(\tau K)y_\ell$ is required. The other terms are scalar multiples of the previous evaluations, which have to be stored.

Remark. It is worth noting that, using recursively the relation

$$\varphi_{k-1}(z) = \frac{1}{(k-1)!} + \varphi_k(z)z, \quad k > 0, \tag{4.7}$$

it is possible to express all appearing φ-functions in (4.6) by φ_k. For example, for $k = 2$, we have

$$y_{\ell+1} = y_\ell + \tau K y_\ell + \ell\tau^2 v + \tau^2\varphi_2(\tau K)(K^2 y_\ell + v + \ell\tau K v)$$

This approach is computationally more convenient for $k > 1$ and small L, since it requires only one matrix function evaluation for $\ell = 0$, too. However, since powers of K are involved in the formula, severe cancellation errors might appear, destroying the stability of the integrator (4.6).

5 Numerical comparisons and implementation issues

Exponential Rosenbrock-type methods are *explicit* time stepping schemes. Their implementation is therefore quite standard, apart from the calculation of the exponential and related functions, which has been described above.

Our variable step size implementations of `erow2`, `erow32` and `erow43` are based on a reliable control of the local error

$$v = u_{n+1} - \widetilde{u}_{n+1},$$

where $\widetilde{u}_{n+1}$ is the result of the embedded method, see Section 3. When stepping from u_n to u_{n+1}, errors are measured in the weighted and scaled norm

$$\|v\|_{\mathrm{E}} = \sqrt{\frac{1}{N}\sum_{i=1}^{N}\left(\frac{v_i}{\mathrm{scal}_i}\right)^2}, \qquad \mathrm{scal}_i = \mathrm{tol}_{\mathrm{a}} + \mathrm{tol}_{\mathrm{r}} \cdot \max\{|u_{n,i}|, |u_{n+1,i}|\}, \quad (5.1)$$

with the absolute tolerance $\mathrm{tol}_{\mathrm{a}}$ and the relative tolerance $\mathrm{tol}_{\mathrm{r}}$, see [8, Chap. IV.8]. The result of the step is accepted whenever

$$\|v\|_{\mathrm{E}} \;\leq\; 1.$$

In the evaluation of the φ-functions, the degree of approximation m has to be determined by accuracy requirements. The estimated approximation error (4.2) is again measured in the norm (5.1), but using a slightly modified scaling factor

$$\mathrm{scal}_i = \mathrm{tol}_{\mathrm{a}} + \mathrm{tol}_{\mathrm{r}} \cdot \|u_n\|_\infty. \quad (5.2)$$

As stopping criterium, we take

$$\mathtt{sf} \cdot \|e_m\|_{\mathrm{E}} \;\leq\; 1, \quad (5.3)$$

where `sf` denotes a security factor. Our numerical tests gave best results for $\mathtt{sf} = 10^p$ with p denoting the order of the method, see Figure 3.

5.1 Numerical experiments in MATLAB

As a first numerical example, we consider the semilinear reaction-diffusion-advection equation

$$\partial_t u = \varepsilon(\partial_{xx}u + \partial_{yy}u) - \alpha(\partial_x u + \partial_y u) + \gamma\, u(u - \tfrac{1}{2})(1 - u) \quad (5.4a)$$

on the unit square $\Omega = [0,1]^2$, subject to homogeneous Neumann boundary conditions. We choose $\varepsilon = 1/20$, $\alpha = -1$, $\gamma = 1$, and take as initial condition

$$u(t = 0, x, y) = 0.3 + 256\left(x(1-x)y(1-y)\right)^2. \quad (5.4b)$$

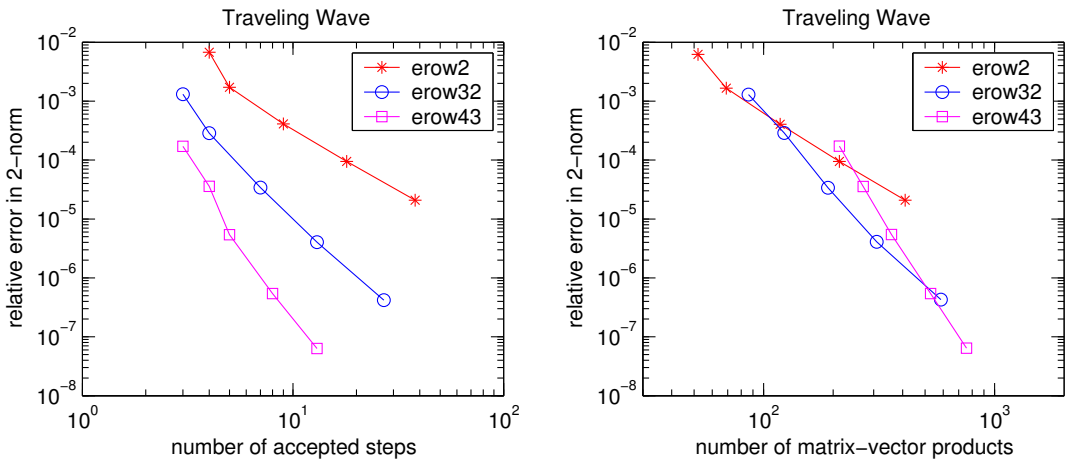

Fig. 1. Achieved accuracy vs. number of steps (left) and number of matrix-vector multiplications (right) for the exponential Rosenbrock methods `erow2`, `erow32`, and `erow43`, when applied to (5.4). The symbols mark the results, obtained for the prescribed tolerances $\mathrm{tol_a} = \mathrm{tol_r} = 10^{-2}$, 10^{-3}, 10^{-4}, 10^{-5}, and 10^{-6}, respectively.

We discretise this problem in space by standard finite differences with mesh-width $\Delta x = \Delta y = 0.05$. This gives a mildly stiff system of ODEs of size $N = 441$, which is integrated in time up to $T = 0.3$. (A much finer discretisation, leading to a stiffer system, will be discussed in Section 5.2 below). The numerical results are displayed in Figure 1. The left figure clearly shows the expected orders of convergence of our variable step size implementations. The right figure indicates that the methods `erow2`, `erow32` and `erow43` are most efficient for low, middle and high accuracies, respectively.

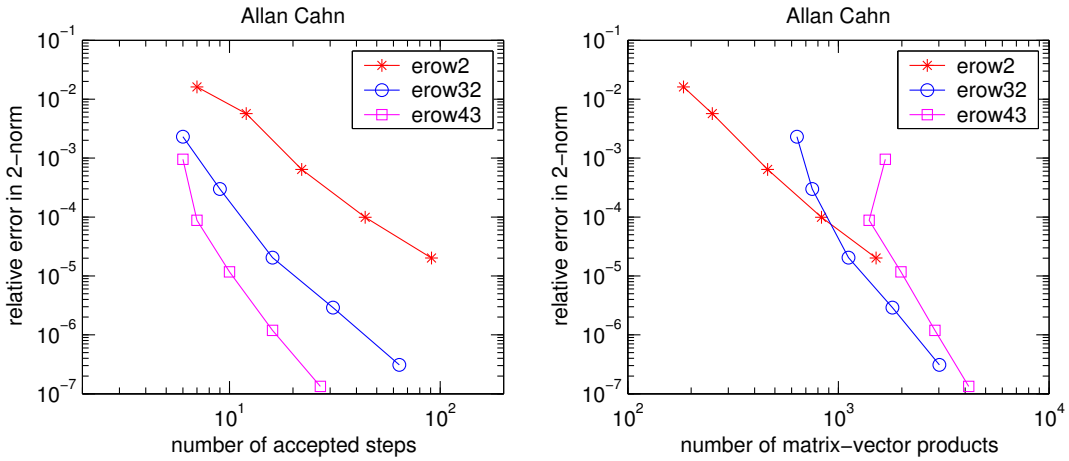

Fig. 2. Achieved accuracy vs. number of steps (left) and number of matrix-vector multiplications (right) for the exponential Rosenbrock methods `erow2`, `erow32`, and `erow43`, when applied to (5.5). The symbols mark the results, obtained for the prescribed tolerances $\mathrm{tol_a} = \mathrm{tol_r} = 10^{-2}$, 10^{-3}, 10^{-4}, 10^{-5}, and 10^{-6}, respectively.

As a second example, we consider the Allen–Cahn equation

$$\partial_t u = \varepsilon \partial_{xx} u + (u + x) - (u + x)^3 \tag{5.5a}$$

on the interval $\Omega = [-1, 1]$, subject to homogeneous Dirichlet boundary conditions. We choose $\varepsilon = 0.01$ and take as initial condition

$$u(t = 0, x) = 0.53\, x + 0.47 \sin(-\tfrac{3}{2}\pi x) - x, \tag{5.5b}$$

which is compatible with the boundary conditions. We discretise this problem in space with a standard Chebyshev spectral method. This gives a stiff ODE system of size $N = 30$, which is integrated in time up to $T = 3$. The left figure again clearly shows the expected orders of convergence of our variable step size implementations. The right figure indicates that `erow2` is most efficient for low and middle accuracies, whereas `erow32` is superior for higher ones.

The choice of the security factor `sf` in (5.3) strongly influences the achieved accuracy of the overall integration. We illustrate this with a numerical experiment. We integrated once more the Allan–Cahn equation with `erow32`, but this time for various choices of the security factor. The obtained results are displayed in Figure 3. The outcome of this and many further experiments motivated us to choose `sf` $= 10^p$ with p denoting the order of the method.

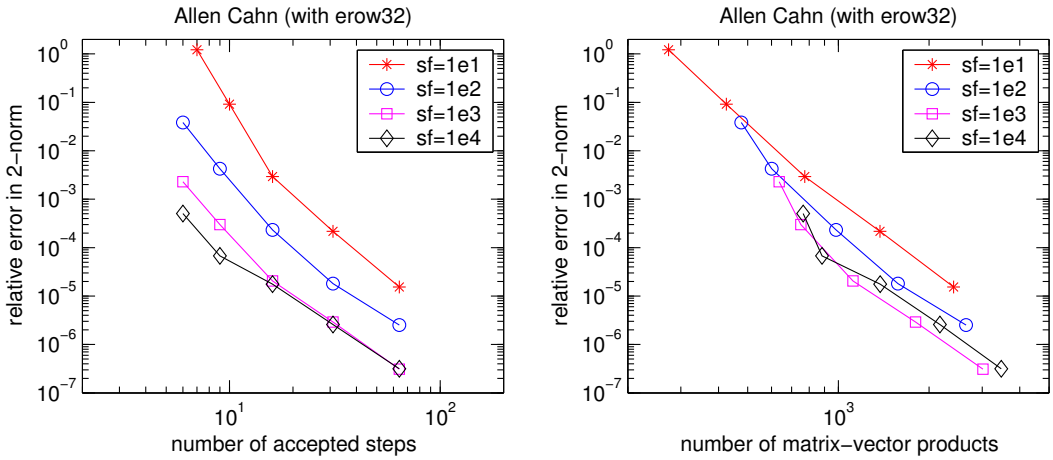

Fig. 3. Achieved accuracy vs. number of steps (left) and number of matrix-vector multiplications (right) for the exponential Rosenbrock method `erow32` when applied to (5.5). We chose as security factors `sf` = 10, 100, 1000, and 10000, respectively. The symbols mark the results, obtained for the prescribed tolerances $\mathrm{tol_a} = \mathrm{tol_r} = 10^{-2}$, 10^{-3}, 10^{-4}, 10^{-5}, and 10^{-6}, respectively.

5.2 Numerical experiments in FORTRAN

The exponential Rosenbrock method of order two has been implemented also in Fortran, using the Real Leja Points Method as described in Section 4. The

resulting code is called `erow2l`. We further implemented the same Rosenbrock integrator with a standard Krylov subspace method for approximating the matrix functions. To this aim, we used the `phipro` Fortran code by Saad (cf. [7]). The resulting code is called `erow2k`.

As numerical example, we consider once more the semilinear reaction-diffusion-advection equation (5.4). This time, we discretise the problem in space by standard finite differences with meshwidth $\Delta x = \Delta y = 0.005$. This results in a stiff system of ODEs of size $N = 40401$, which is integrated up to $T = 0.3$.

Regarding the `phipro` code, the dimension of the Krylov subspace m is an input parameter: the time step h is automatically subdivided into smaller substeps τ in order to compute $\varphi_1(\tau h J)v$ within the maximum number of iterations m. In our experiments, m was chosen equal to 10 for $\gamma = 100$ and equal to 15 for $\gamma = 1$. The reason for this choice is that smaller time steps are expected for the stiffer reaction.

The above methods have been compared with three other second-order methods, namely the explicit Runge–Kutta–Chebyshev method `rkc` [19], the explicit Orthogonal Runge–Kutta–Chebyshev method `rock2` [1], and the classical implicit Crank–Nicolson scheme `cn`. For the latter, an inexact Newton scheme has been used (the Jacobian was computed only at the first iteration), with the absolute error estimate at iteration r (cf. [8, Chap. IV.8])

$$\|e_r\|_{\mathrm{E}} = \eta_r \|u_{n+1}^r - u_{n+1}^{r-1}\|_{\mathrm{E}}, \quad \eta_r = \frac{\theta_r}{1-\theta_r}, \quad \theta_r = \frac{\|u_{n+1}^r - u_{n+1}^{r-1}\|_{\mathrm{E}}}{\|u_{n+1}^{r-1} - u_{n+1}^{r-2}\|_{\mathrm{E}}}. \tag{5.6}$$

The linear systems inside the Newton iterations have been solved by the BiCGStab method, preconditioned by the ILU factorisation with no fill-in. For estimating the local truncation error and step size control, we have used a stabilised finite-difference approximation of the third derivatives. The stabilisation was achieved by solving an additional linear system involving the same Jacobian as in the Newton process. A similar idea has been used in [8, Chap. IV.8].

From our numerical experiments, we draw the following conclusions. First of all, it can be seen from Tables 2–4 that `erow2l` performs better (in almost all cases) than `erow2k` in terms of total CPU time. The maximum speed up is about 2.1, see Table 3, $\mathrm{tol_a} = \mathrm{tol_r} = $ 1E-4, $\varepsilon = 1$, where both methods are much more precise than the prescribed accuracy, with `erow2k` slightly better in terms of achieved accuracy. The dimension of the Krylov subspace m could be tuned in order to improve the performance and to reduce the extra accuracy that `erow2k` sometimes reaches (see, e.g., Table 3, $\mathrm{tol_a} = \mathrm{tol_r} = $ 1E-5, $\varepsilon = 1$ or Table 4, $\mathrm{tol_a} = \mathrm{tol_r} = $ 1E-4, $\alpha = -5$). Nevertheless, the optimal choice of the Krylov subspace is not known a priori and it highly depends on the parameters of the problem, the spatial discretisation and the prescribed accuracy.

Table 2
CPU time in seconds (CPU), number of time steps (nts) and relative error in 2-norm (err.) for various methods when applied to (5.4) with $\alpha = -1$ and $\gamma = 1$, and the value of ε and the prescribed tolerances as reported.

$\mathrm{tol_a} = \mathrm{tol_r} =$ 1E-4									
	$\varepsilon = 0.05$			$\varepsilon = 0.1$			$\varepsilon = 1$		
method	CPU	nts	err.	CPU	nts	err.	CPU	nts	err.
`rkc`	1.92	22	5.2E-4	2.63	22	5.1E-4	8.52	32	9.4E-5
`rock2`	2.10	39	1.5E-4	2.89	39	1.2E-4	9.07	52	4.6E-5
`erow2k`	7.21	12	4.2E-4	11.40	11	3.1E-4	81.34	9	6.8E-5
`erow2l`	4.40	12	4.3E-4	7.71	12	3.4E-4	55.10	10	7.7E-4
`cn`	15.20	17	1.1E-3	21.57	18	9.4E-4	66.77	29	2.2E-3
$\mathrm{tol_a} = \mathrm{tol_r} =$ 1E-5									
	$\varepsilon = 0.05$			$\varepsilon = 0.1$			$\varepsilon = 1$		
method	CPU	nts	err.	CPU	nts	err.	CPU	nts	err.
`rkc`	2.66	40	1.5E-4	3.61	40	1.2E-4	12.16	63	2.5E-5
`rock2`	3.83	118	1.5E-5	5.17	115	1.3E-5	15.55	153	6.2E-6
`erow2k`	7.43	21	9.2E-5	11.98	20	7.0E-5	91.93	14	1.9E-5
`erow2l`	5.05	21	9.2E-5	9.14	20	6.8E-5	68.41	15	3.2E-4
`cn`	22.21	33	4.0E-4	31.40	33	3.6E-4	114.13	53	7.8E-5
$\mathrm{tol_a} = \mathrm{tol_r} =$ 1E-6									
	$\varepsilon = 0.05$			$\varepsilon = 0.1$			$\varepsilon = 1$		
method	CPU	nts	err.	CPU	nts	err.	CPU	nts	err.
`rkc`	4.07	83	3.3E-5	5.49	84	2.5E-5	18.24	130	5.3E-6
`rock2`	7.55	381	1.6E-6	9.79	366	1.3E-6	27.93	472	6.9E-7
`erow2k`	8.93	41	2.1E-5	13.35	38	1.6E-5	95.22	25	5.1E-6
`erow2l`	7.60	41	2.2E-5	10.90	38	1.6E-5	76.50	26	6.0E-6
`cn`	33.54	66	4.2E-4	50.08	67	3.2E-4	165.75	107	1.8E-5

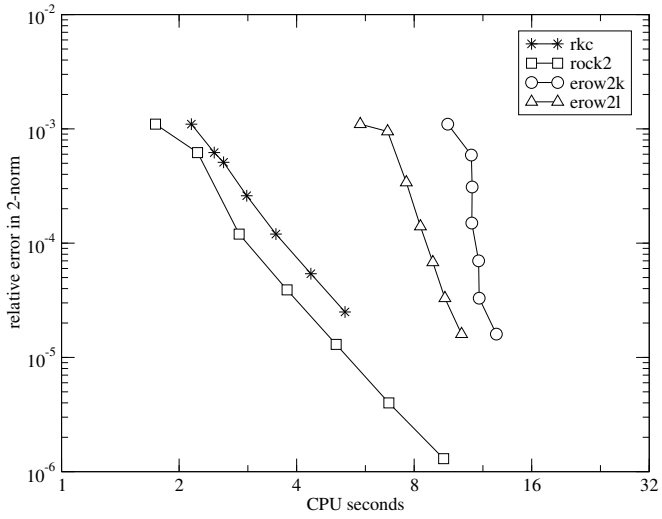

Fig. 4. Achieved accuracy vs. CPU time for various methods when applied to (5.4) with $\varepsilon = 0.1$, $\alpha = -1$ and $\gamma = 1$. The symbols mark the results, obtained for the prescribed tolerances $\mathrm{tol_a} = \mathrm{tol_r} = 10^{-3}, 10^{-3.5}, 10^{-4}, 10^{-4.5}, 10^{-5}, 10^{-5.5}$, and 10^{-6}.

Table 3
CPU time in seconds (CPU), number of time steps (nts) and relative error in 2-norm (err.) for various methods when applied to (5.4) with $\alpha = -1$ and $\gamma = 100$, and the value of ε and the prescribed tolerances as reported.

$\mathrm{tol_a} = \mathrm{tol_r} =$ 1E-4									
	$\varepsilon = 0.05$			$\varepsilon = 0.1$			$\varepsilon = 1$		
method	CPU	nts	err.	CPU	nts	err.	CPU	nts	err.
`rkc`	3.16	54	1.4E-3	4.05	49	1.2E-3	10.66	43	2.4E-5
`rock2`	3.57	105	3.4E-4	4.56	92	2.7E-4	11.59	72	1.7E-5
`erow2k`	11.79	72	4.5E-4	17.07	66	4.5E-4	78.21	44	1.2E-6
`erow2l`	9.35	73	5.0E-4	10.96	68	4.5E-4	37.08	45	6.0E-6
`cn`	27.79	43	2.6E-3	39.09	38	2.1E-3	74.99	35	1.1E-5
$\mathrm{tol_a} = \mathrm{tol_r} =$ 1E-5									
	$\varepsilon = 0.05$			$\varepsilon = 0.1$			$\varepsilon = 1$		
method	CPU	nts	err.	CPU	nts	err.	CPU	nts	err.
`rkc`	4.93	115	3.2E-4	6.17	102	2.7E-4	15.53	87	5.5E-6
`rock2`	7.00	337	3.5E-5	8.64	289	2.7E-5	20.01	215	2.9E-6
`erow2k`	16.42	155	1.1E-4	19.97	141	1.1E-4	82.56	89	3.0E-7
`erow2l`	17.59	155	1.1E-4	18.53	143	1.3E-4	49.18	90	2.5E-6
`cn`	43.77	89	6.8E-4	60.21	79	7.3E-4	136.59	69	4.5E-6
$\mathrm{tol_a} = \mathrm{tol_r} =$ 1E-6									
	$\varepsilon = 0.05$			$\varepsilon = 0.1$			$\varepsilon = 1$		
method	CPU	nts	err.	CPU	nts	err.	CPU	nts	err.
`rkc`	7.78	247	7.4E-5	9.63	217	6.0E-5	22.43	180	1.2E-6
`rock2`	15.14	1151	3.8E-6	17.26	953	2.8E-6	34.81	668	4.7E-7
`erow2k`	35.01	334	2.6E-5	32.67	303	2.4E-5	96.14	185	7.6E-8
`erow2l`	33.68	334	2.6E-5	37.83	304	2.4E-5	62.21	186	1.5E-6
`cn`	70.50	190	1.2E-3	92.90	167	1.2E-3	243.39	147	1.7E-6

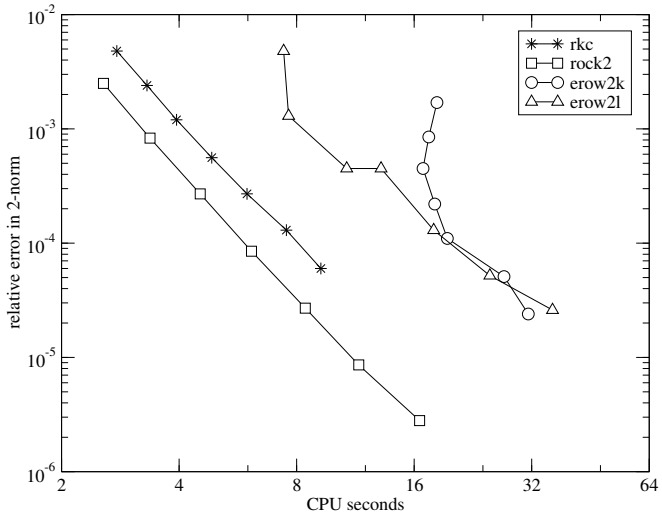

Fig. 5. Achieved accuracy vs. CPU time for various methods when applied to (5.4) with $\varepsilon = 0.1$, $\alpha = -1$ and $\gamma = 100$. The symbols mark the results, obtained for the prescribed tolerances $\mathrm{tol_a} = \mathrm{tol_r} = 10^{-3}, 10^{-3.5}, 10^{-4}, 10^{-4.5}, 10^{-5}, 10^{-5.5}$, and 10^{-6}.

Table 4
CPU time in seconds (CPU), number of time steps (nts) and relative error in 2-norm (err.) for various methods when applied to (5.4) with $\varepsilon = 0.1$ and $\gamma = 1$, and the value of α and the prescribed tolerances as reported.

$\mathrm{tol_a} = \mathrm{tol_r} =$ 1E-4						
	$\alpha = -5$			$\alpha = -10$		
method	CPU	nts	err.	CPU	nts	err.
`rkc`	6.93	123	2.3E-5	10.59	218	2.9E-5
`rock2`	4.56	102	7.7E-4	11.30	471	3.9E-4
`erow2k`	14.22	16	2.5E-7	10.97	14	5.9E-6
`erow2l`	8.52	17	1.4E-5	6.22	15	1.1E-5
`cn`	33.20	40	5.1E-5	32.45	43	1.8E-2
$\mathrm{tol_a} = \mathrm{tol_r} =$ 1E-5						
	$\alpha = -5$			$\alpha = -10$		
method	CPU	nts	err.	CPU	nts	err.
`rkc`	6.00	104	7.9E-6	11.53	285	2.8E-6
`rock2`	7.98	295	7.1E-6	13.91	656	1.9E-5
`erow2k`	15.05	29	3.1E-7	14.10	26	2.4E-6
`erow2l`	9.25	30	1.7E-5	7.82	26	5.8E-6
`cn`	48.40	81	4.2E-5	56.90	90	4.0E-5
$\mathrm{tol_a} = \mathrm{tol_r} =$ 1E-6						
	$\alpha = -5$			$\alpha = -10$		
method	CPU	nts	err.	CPU	nts	err.
`rkc`	8.95	215	2.3E-6	24.33	1020	3.6E-7
`rock2`	16.01	984	8.7E-7	21.59	1356	2.6E-6
`erow2k`	18.16	57	3.7E-7	16.39	51	1.5E-6
`erow2l`	13.53	58	3.5E-7	11.44	51	1.6E-6
`cn`	75.32	168	3.9E-5	86.42	185	6.0E-5

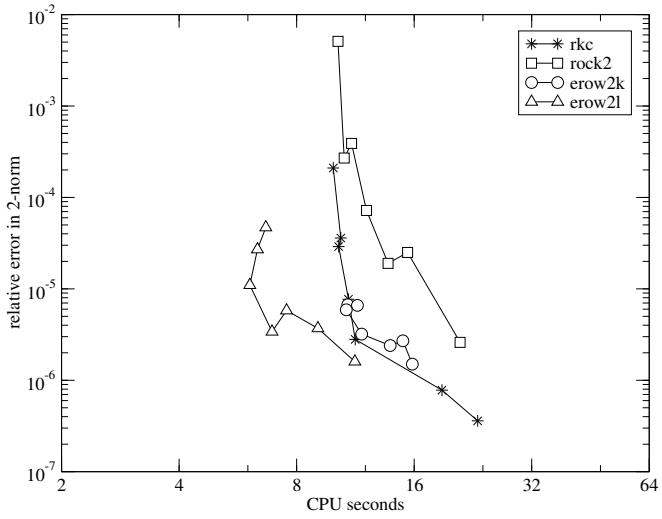

Fig. 6. Achieved accuracy vs. CPU time for various methods when applied to (5.4) with $\varepsilon = 0.1$, $\alpha = -10$ and $\gamma = 1$. The symbols mark the results, obtained for the prescribed tolerances $\mathrm{tol_a} = \mathrm{tol_r} = 10^{-3}, 10^{-3.5}, 10^{-4}, 10^{-4.5}, 10^{-5}, 10^{-5.5}$, and 10^{-6}.

From Tables 2–3, collecting the results for small advection $\alpha = -1$, it is clear that the `rkc` and `rock2` perform significantly better than the other methods, with `rock2` being more reliable in reaching the prescribed accuracy (as it is also clear from Figures 4–5). For large advection (that is, non-normal matrices), the results of the exponential Rosenbrock methods are much closer to the results of the Runge–Kutta–Chebyshev methods, and the former perform sometimes even better (in particular `erow2l`, see Table 4, $\alpha = -10$).

In any case, the implicit `cn` method performs very badly in comparison with the others, both in terms of CPU time and achieved accuracy.

6 Conclusions

In this paper we have presented a variable step size implementation of exponential Rosenbrock-type methods of orders 2, 3 and 4. These integrators require the exponential and related functions, evaluated at the Jacobian matrix of the problem. In general, this matrix has no special structure and varies continuously from step to step. For evaluating the matrix functions, the Real Leja Points Method has been used. The properties of this method meet well the requirements of Rosenbrock-type integrators.

For large problems, the computational work of the Real Leja Points Method mainly consists in performing matrix-vector multiplications. In our substepping implementation, the computational work grows roughly linearly with the stiffness of the problem. For Runge–Kutta–Chebyshev methods, however, the stability domain grows *quadratically* with the number of stages [8, Chap. IV.2]. The computational work thus grows with the *square root* of the stiffness. This fact explains well, why Runge–Kutta–Chebyshev methods are more efficient for pure reaction-diffusion problems, where the reaction part can be evaluated cheaply, see Tables 2 and 3.

However, as soon as advection plays a decisive role, the picture changes, see Table 4. We find a great potential of our methods for non-normal matrices. A typical situation where this occurs in practice are problems with large advection in combination with diffusion and mildly stiff (and maybe expensive) reactions.

References

[1] Abdulle, A., Medovikov, A. A., 2001. Second order Chebyshev methods based on orthogonal polynomials. Numer. Math. 90, 1–18.

[2] Bergamaschi, L., Caliari, M., Martínez, A., Vianello, M., 2005. A parallel exponential integrator for large-scale discretizations of advection-diffusion models. In: Di Martino, B. e. a. (Ed.), Recent Advances in Parallel Virtual Machine and Message Passing Interface. Vol. 3666 of LNCS. Springer, Berlin, Heidelberg, pp. 483–492.

[3] Bergamaschi, L., Caliari, M., Martínez, A., Vianello, M., 2006. Comparing Leja and Krylov approximations of large scale matrix exponentials. In: Alexandrov, V. N. e. a. (Ed.), Computational Science — ICCS 2006. Vol. 3994 of LNCS. Springer, Berlin, Heidelberg, pp. 685–692.

[4] Berland, H., Skaflestad, B., Wright, W., 2006. Expint — A Matlab package for exponential integrators. ACM Trans. Math. Software, to appear.

[5] Caliari, M., Vianello, M., Bergamaschi, L., 2004. Interpolating discrete advection-diffusion propagators at Leja sequences. J. Comput. Appl. Math. 172, 79–99.

[6] Calvo, M. P., Palencia, C., 2006. A class of explicit multistep exponential integrators for semilinear problems. Numer. Math. 102, 367–381.

[7] Gallopoulos, E., Saad, Y., 1992. Efficient solution of parabolic equations by Krylov subspace methods. SIAM J. Sci. Statist. Comput. 13, 1236–1264.

[8] Hairer, E., Wanner, G., 1996. Solving Ordinary Differential Equations II. Stiff and Differential-Algebraic Problems, 2nd Edition. Springer, Berlin, Heidelberg.

[9] Henry, D., 1981. Geometric Theory of Semilinear Parabolic Equations. Vol. 840 of LMN. Springer, Berlin, Heidelberg.

[10] Hochbruck, M., Lubich, C., 1997. On Krylov subspace approximations to the matrix exponential operator. SIAM J. Numer. Anal. 34, 1911–1925.

[11] Hochbruck, M., Ostermann, A., 2005. Explicit exponential Runge-Kutta methods for semilinear parabolic problems. SIAM J. Numer. Anal. 43, 1069–1090.

[12] Hochbruck, M., Ostermann, A., 2006. Exponential integrators of Rosenbrock-type. Oberwolfach Reports 3, 1107–1110.

[13] Hochbruck, M., Ostermann, A., Schweitzer, J., 2006. Exponential Rosenbrock-type methods. In preparation.

[14] Martínez, A., Bergamaschi, L., Caliari, M., Vianello, M., 2006. Efficient massively parallel implementation of the ReLPM exponential integrator for advection-diffusion models. Submitted.

[15] Novati, P., Moret, I., 2004. RD-rational approximation of the matrix exponential operator. BIT 44, 595–615.

[16] Ostermann, A., Thalhammer, M., Wright, W., 2006. A class of explicit exponential general linear methods. BIT 46, 409–431.

[17] Reichel, L., 1990. Newton interpolation at Leja points. BIT 30, 332–346.

[18] Schmelzer, T., Trefethen, L. N., 2006. Evaluating matrix functions for exponential integrators via Carathéodory-Fejér approximation and contour integrals. Submitted to ETNA.

[19] Sommeijer, B. P., Shampine, L. F., Verwer, J. G., 1997. RKC: An explicit

solver for parabolic PDEs. J. Comp. Appl. Math. 88, 315–326.
[20] Tal-Ezer, H., 1991. High degree polynomial interpolation in Newton form. SIAM J. Sci. Statist. Comput. 12, 648–667.
[21] van den Eshof, J., Hochbruck, M., 2006. Preconditioning Lanczos approximations to the matrix exponential. SIAM J. Sci. Comp. 27, 1438–1457.