Master Thesis

Matrix-free Leja based exponential integrators in Python

Maximilian Samsinger

February 28, 2020

Supervised by Lukas Einkemmer and
Alexander Ostermann

Leopold-Franzens-Universität Innsbruck

Eidesstattliche Erklärung

Ich erkläre hiermit an Eides statt durch meine eigenhändige Unterschrift, dass ich die vorliegende Arbeit selbständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel verwendet habe. Alle Stellen, die wörtlich oder inhaltlich den angegebenen Quellen entnommen wurden, sind als solche kenntlich gemacht.

Ich erkläre mich mit der Archivierung der vorliegenden Masterarbeit einverstanden.

Datum	Unterschrift

Matrix-free Leja based exponential integrators in Python

Abstract

In this master thesis we develop an algorithm to approximate the action of a matrix exponential function for matrix-free linear operators. This is achieved by using a modified version of the real Leja method where we choose optimal interpolation parameters based on a spectral radius estimate of the power method. With this procedure we construct exponential Rosenbrock-type integrators to solve stiff advection-diffusion-reaction equations. We compare the performance of these integrators with other matrix-free differential equation solvers. As part of this thesis we provide the code for the matrix-free Leja method for the action of the matrix exponential function.

1 Introduction

Consider the action of the matrix exponential function

$$e^A v \quad \text{where} \quad A \in \mathbb{C}^{N\times N} \quad \text{and} \quad v \in \mathbb{C}^N.$$

Due to computational constraints it can be difficult or impossible to compute e^A in a first step and then the action $e^A v$ in a separate step. This is especially true in applications where $N > 10000$ is common. Furthermore the matrix exponential of a sparse matrix is in general no longer sparse. Therefore it is more feasible to compute the action of the matrix exponential in a single step. This can be done by approximating the matrix exponential with a matrix polynomial p_m of degree m in A

$$e^A v \approx p_m(A)v.$$

This approach has many benefits. The cost of the computation of $p_m(A)v$ mainly depends on the calculation of m matrix-vector multiplications with A, which is inexpensive for sparse matrices. Furthermore the explicit knowledge of A itself is no longer required. A can be replaced by a linear operator, which can be convenient and save memory.

We introduce an polynomial interpolation procedure, the Leja method, in Section 2. The Leja method has many computational advantages. All interpolation nodes can be precomputed and stored for later use. The interpolation itself is done iteratively and can be interrupted if the interpolation error is small enough. For entire functions the Leja interpolation has beneficial convergence properties which translate well to the corresponding matrix functions. After this brief introduction to the Leja method we discuss an application to the action of the matrix exponential function. We consider an approach which bounds the backward error of the interpolation. This will lead to an algorithm which provides cost-minimizing parameters for the Leja interpolation based on the operator norm $\|A\|$ of A.

In Section 3 we adapt the algorithm in order to obtain a matrix-free version. The implementation of the Newton interpolation is straight-forward for linear operators. However, without the matrix representation it can be computationally infeasible to calculate the operator norm. Instead we replace all instances of $\|A\|$ with the spectral radius, which can be cheaply estimated using the power method. This modification introduces new challenges. On the one hand it is unclear how many power iterations are necessary for a sufficient estimate of the spectral radius. On the other hand it is no longer possible to specify the norm for the backward error bound.
In Section 4 we study the matrix-free Leja method for the discretized linear, one-dimensional advection-diffusion equation. For this specific problem we experimentally and numerically investigate the behavior of the power method. We identify a suitable choice for the number of power iterations. Furthermore we presume that this choice is reasonable in the nonlinear case as well. In order to verify that claim we introduce exponential Rosenbrock-type integrators in Section 5. Finally, we compare the performance of three matrix-free Leja based exponential integrators of different order against other matrix-free differential equation solvers in Section 6.

2 The Leja method

This section serves as an introduction to the Leja method for approximating the action of the exponential function. We briefly cover the key concepts and definitions in Section 2.1 and 2.2. For more details and all proofs we refer to [4] and [5].

2.1 Leja interpolation

Let $K \in \mathbb{C}$ be a compact set in the complex plane and $\xi_0 \in K$ be arbitrary. The sequence $(\xi_k)_{k=0}^{\infty}$ recursively defined as

$$\xi_k = \arg\max_{\xi \in K} \prod_{j=0}^{k-1} |\xi - \xi_j|$$

is called a Leja sequence. Due to the maximum principle all elements in the sequence realize their maximum on the border ∂K. Typically ξ_0 is also chosen on ∂K.
For analytical functions $f\colon K \to \mathbb{C}$ the Newton interpolation polynomial p_m with nodes $(\xi_k)_{k=0}^{m}$ has the following beneficial properties.

Convergence properties: The sequence $(p_m)_{m=0}^{\infty}$ converges maximally to f. That is, let $(p_m^*)_{m=0}^{\infty}$ be the best uniform approximation polynomials for f in K. Then

$$\limsup_{m\to\infty} \|f - p_m\|_K^{1/m} = \limsup_{m\to\infty} \|f - p_m^*\|_K^{1/m},$$

where $\|\cdot\|_K$ is the maximum norm on K. Furthermore if f is an entire function, then $(p_m)_{m=0}^{\infty}$ converges superlinearly to f

$$\limsup_{m\to\infty} \|f - p_m\|_{\mathbb{C}}^{1/m} = \limsup_{m\to\infty} \|f - p_m^*\|_{\mathbb{C}}^{1/m} = 0.$$

For entire functions f the corresponding matrix polynomials achieves similar superlinear convergence

$$\limsup_{m\to\infty} \|f(A)v - p_m(A)v\|_2^{1/m} = 0,$$

for $A \in \mathbb{C}^{n\times n}, v \in \mathbb{C}^n$.

Early termination: The Newton interpolation polynomial p_m can be constructed iteratively since the corresponding Leja interpolation points $(\xi_k)_{k=0}^m$ are defined recursively. Therefore if the approximation $p_n \approx f$ is accurate enough after $n < m$ steps the interpolation can be stopped early to reduce the cost of the interpolation. Note that this is not possible with Chebyshev nodes.

Leja sequence can be stored: For a given K the Leja interpolation nodes only need to be computed once and for all. These values can be stored a priori and loaded once they are needed for the interpolation. If f is fixed the same is also true for the corresponding divided differences.

In summary the Leja points offer convergence properties similar to Chebyshev nodes for interpolation, while having computational advantages. All results hold true for the corresponding matrix interpolation polynomials.

2.2 Approximating the matrix exponential function:

Inspired by the previous subsection we try to find a low-cost approximation of the action of the matrix exponential $e^A v$ using Leja interpolation polynomials. From now on, we will fix

$$K = [-c, c], \quad f = e^{\cdot} \quad \text{and} \quad \xi_0 = c$$

for $c > 0$. With $L_{m,c}$ we denote the Leja interpolation polynomial on the interval $[-c, c]$ with Leja points $(\xi_j)_{j=0}^m$. We use the well-known property of the exponential function

$$e^A v = (e^{s^{-1}A})^s v, \quad \text{with} \quad s \in \mathbb{N}.$$

Now we can approximate the action of the matrix exponential in s substeps

$$v_0 := v, \quad v_{j+1} := L_{m,c}(s^{-1}A)v_j, \quad \text{and} \quad v_s \approx e^A v.$$

So far we placed no restrictions on m, s and c. We choose optimal parameters based on the backward-error analysis done in [5].

Bounding the backward error For a given matrix A we interpret the Leja interpolation polynomial as the exact solution of a perturbed matrix exponential function

$$L_{m,c}(s^{-1}A)^s v =: e^{A+\Delta A} v$$

m	5	10	15	20	25	30	35
half	6.43e-01	2.12e+00	3.55e+00	5.00e+00	6.37e+00	7.51e+00	8.91e+00
single	9.62e-02	8.33e-01	1.96e+00	3.26e+00	4.69e+00	5.96e+00	7.44e+00
double	1.74e-03	1.14e-01	5.31e-01	1.23e+00	2.16e+00	3.18e+00	4.34e+00

m	40	45	50	55	60	65	70
half	1.00e+01	1.10e+01	1.23e+01	1.35e+01	1.48e+01	1.59e+01	1.71e+01
single	8.71e+00	1.00e+01	1.15e+01	1.27e+01	1.40e+01	1.52e+01	1.64e+01
double	5.48e+00	6.67e+00	7.99e+00	9.24e+00	1.06e+01	1.18e+01	1.32e+01

m	75	80	85	90	95	100
half	1.84e+01	1.94e+01	2.07e+01	2.20e+01	2.30e+01	2.42e+01
single	1.76e+01	1.87e+01	1.99e+01	2.12e+01	2.23e+01	2.35e+01
double	1.46e+01	1.58e+01	1.71e+01	1.86e+01	1.99e+01	2.13e+01

Table 1: Samples of the precomputed values θ_m. The backward error of the Leja interpolation is bounded if $c \leq \theta_m$, where $[-c, c]$ is the interpolation interval and m the interpolation degree. Half, single and double correspond to the tolerances 2^{-10}, 2^{-24} and 2^{-53} respectively [5, Table 1].

Our goal is to bound the backward error

$$\frac{\|\Delta A\|}{\|A\|} \leq \text{tol},$$

for a given tolerance tol. Furthermore we want to minimize the cost of the interpolation. A priori it is unclear for which values m, s and c the inequality is satisfied. The authors of [5] conducted a backward error analysis and chose an approach which puts an upper bound on c depending only on s and m. For various tolerances tol they precomputed values θ_m, see 1, which satisfy

$$\text{If } \|s^{-1}A\| \leq \theta_m \text{ and } 0 \leq c \leq \theta_m \text{ then } \frac{\|\Delta A\|}{\|A\|} \leq \text{tol}\,.$$

For our purposes it is important to note that the optimal choice for c is given by $c = \rho(s^{-1}A)$, where $\rho(A)$ is the spectral radius of A. However, computing $\rho(A)$ introduces additional costs for the algorithms proposed in [5]. Our matrix-free implementation relies on the computations of the spectral radius, but it does not need to compute the operator norm $\|A\|$, see Section 3.

Choosing cost-minimizing parameters The cost of the Leja interpolation mainly depends on the the number of matrix-vector products

$$C_m = sm.$$

In order to minimize the costs of the interpolation C_m we select the smallest m for any given s such that

$$\|s^{-1}A\| \leq \theta_m$$

is satisfied. This leads to the optimal choice for m and s

$$m_* = \underset{2\leq m\leq m_{\max}}{\arg\min} \left\{ \left\lceil \frac{\|A\|}{\theta_m} \right\rceil m \right\} \quad \text{and} \quad s_* = \left\lceil \frac{\|A\|}{\theta_m} \right\rceil . \tag{1}$$

In our algorithm we set $m_{\max} = 100$ in order to avoid over- and underflow errors.

Shifting the matrix The cost of the interpolation can be decreased by employing a shift $\mu \in \mathbb{C}$. Let I be the identity matrix. We replace the matrix A with $A - \mu I$ for all computations. If the shifted matrix $A - \mu I$ satisfies $\|A - \mu I\| < \|A\|$ then the cost C_{m_*} of the interpolation decreases. We compensate for the shift by multiplying with e^{μ} since

$$e^A = e^{\mu} e^{A-\mu I}.$$

A well-chosen shift centers the eigenvalues of $A - \mu I$ around 0. Such a shift can be found by using Gerschgorin's circle theorem. This is, however, not possible in the matrix-free case.

3 Matrix-free implementation

In this section we introduce a matrix-free version of the Leja method for the action of a matrix exponential. After a brief motivation we discuss the main advantages and disadvantages of such an approach.
Performance on most modern systems is limited by memory bandwidth. A matrix-free implementation gives us the opportunity to significantly save memory and thus increase performance. For example, a matrix-free implementation reduces the amount of memory transactions compared to a sparse matrix implementation based on the CSR format by at least a factor of 5. Consider finite-difference schemes

$$v_k = \sum_{j=-n}^{n} a_{k+j} v_{k+j}$$

for $v \in \mathbb{C}^N$, $n \in \mathbb{N}$ and periodic boundaries i.e. $a_{l+N} = a_l$ and $v_{l+N} = v_l$ for all $l \in \mathbb{Z}$. This finite difference scheme can be written as an matrix-vector multiplication with an $(2n+1)$-diagonal matrix. Even if the matrix is stored in a sparse format it is still necessary to save $N(2n+1)$ non-zero elements. In the case of two-dimensional five-point stencil method on a rectangular grid we have $n = 2$ and $N = N_x N_y$, where N_x and N_y is the number of grid points on the x- and y-axis respectively. This leads to a storage demand of $20 N_x N_y$ bytes when saving matrix entries as float data. This means that we surpass 1 megabyte when we choose $N_x, N_y > 250$, which might cause cache misses. This is especially true for L1 and L2 caches. This problem is further magnified for stencils on

three-dimensional domains.
We avoid these difficulties with matrix-free methods, meaning that matrix representations are not explicitly given. Instead we consider linear operators of the form

$$\begin{aligned} A\colon \mathbb{C}^N &\to \mathbb{C}^N \\ v &\mapsto A(v) \end{aligned}.$$

Linear operators can be more convenient to work with in applications where the matrix representation is not easily available. They can also be more memory efficient. For the finite difference schemes we discussed at the beginning of this section we only need to store $(2n+1)$ values independent of N. This serves as a motivation to propose an alternative for the classical Leja method we discussed in Section 2.

Matrix-free Leja method For the most part it is unproblematic to use linear operators for the Leja method instead of matrices, since the Newton interpolation only relies on computation of matrix-vector products. However, difficulties arise when the interpolation parameters need to be determined. Without the matrix representation it can be expensive to compute the operator norm $\|A\|$ in (1). We will circumvent this problem by replacing $\|A\|$ with the spectral radius $\rho(A)$.

The backward error analysis in [5] holds true for every operator norm. We use a well-known result from the matrix analysis literature [9, Lemma 5.6.10.]. For every A and for every $\varepsilon > 0$ exists an induced operator norm $\|\cdot\|_{A,\varepsilon}$ such that

$$\rho(A) \leq \|A\|_{A,\varepsilon} \leq \rho(A) + \varepsilon.$$

The first inequality holds true for every operator norm. We choose ε small enough, such that

$$\|s^{-1}A\|_{A,\varepsilon} \leq \min_{\rho(s^{-1}A)<\theta_m} \theta_m.$$

For this choice of $\|\cdot\|_{A,\varepsilon}$ the cost-minimizing parameters are given by

$$m_* = \operatorname*{arg\,min}_{2\leq m\leq m_{\max}} \left\{ \left\lceil \frac{\rho(A)}{\theta_m} \right\rceil m \right\} \quad \text{and} \quad s_* = \left\lceil \frac{\rho(A)}{\theta_m} \right\rceil. \tag{2}$$

The explicit knowledge of $\|\cdot\|_{A,\varepsilon}$ is no longer required. Additionally we can choose $c = \rho(A)$ without introducing additional costs, since we have to compute $\rho(A)$ to determine m_* and s_*. For positive and negative semi-definite operators A we can choose the shift $\mu = -\rho(A)/2$ and $\mu = \rho(A)/2$ respectively. This shift works particularly well if the absolutely smallest eigenvalue of A is close to 0.
We should point out that this approach has some drawbacks. While we are able to bound the backward error

$$\frac{\|\Delta A\|_{A,\varepsilon}}{\|A\|_{A,\varepsilon}} \leq \text{tol}$$

we can no longer specify in which norm this error has to be bound. Furthermore, it is hard to find a good shift μ for non-semi-definite operators.

Power method The spectral radius $\rho(A)$ can be cheaply approximated using the power method. Given an initial vector $b_0 \in \mathbb{C}^N$ the n-th iteration of the power method is given by

$$b_{n+1} = \frac{Ab_n}{\|Ab_n\|}.$$

Let λ_1 be a maximal eigenvalue with respect to the modulus. Furthermore let λ_2 be a maximal eigenvalue satisfying $|\lambda_1| \neq |\lambda_2|$. Both eigenvalues are not necessarily unique. For each iteration we get an approximation to the spectrum of A

$$\|Ab_n\| = \rho(A) + \mathcal{O}\left(\left|\frac{\lambda_2}{\lambda_1}\right|^{n+1}\right).$$

This can be used to compute (2). However, the power method underestimates the spectrum of A, which might cause the Leja method to not converge. Therefore we have to multiply the estimate with a safety factor. The convergence speed depends on the eigenvalues of A. This will be further analysed in Section 4. From now on we denote matrix-free Leja method for the matrix exponential function as `expleja`. Depending on the chosen tolerance, see Table 1, we will refer to the algorithm as half, single or double precision `expleja` respectively.

4 Linear advection-diffusion equation

In this section we consider a simple initial value problem which serves as a test-bed for future experiments. We also want to examine the power method, an algorithm to (under)-estimate the absolutely largest eigenvalue of an operator.
Consider the one-dimensional advection-diffusion equation

$$\begin{aligned} \partial_t u &= a\partial_{xx} u + b\partial_x u \quad \text{with} \quad a, b \geq 0 \quad \text{and} \\ u_0(x) &= e^{-80\cdot(x-0{,}45)^2} \quad \text{with} \quad x \in [0,1] \end{aligned} \tag{3}$$

on the time interval $[0, 0.1]$. For a fixed $N \in \mathbb{N}$ we approximate the diffusive part of the differential equation with second-order central differences on an equidistant grid with grid size $h = \frac{1}{N-1}$ and grid points $x_k = kh$, $k = 0 \ldots, N-1$

$$\partial_{xx} u(x_k) = \frac{u(x_{k+1}) - 2u(x_k) + u(x_{k-1})}{h^2} + \mathcal{O}(h^2).$$

In order to avoid numerical instabilities we discretize the advective part with forward differences, similar to the upwind scheme

$$\partial_x u(x_k) = \frac{u(x_{k+1}) - u(x_k)}{h} + \mathcal{O}(h).$$

The resulting system of ordinary differential equation is given by

$$\partial_t u = Au.$$

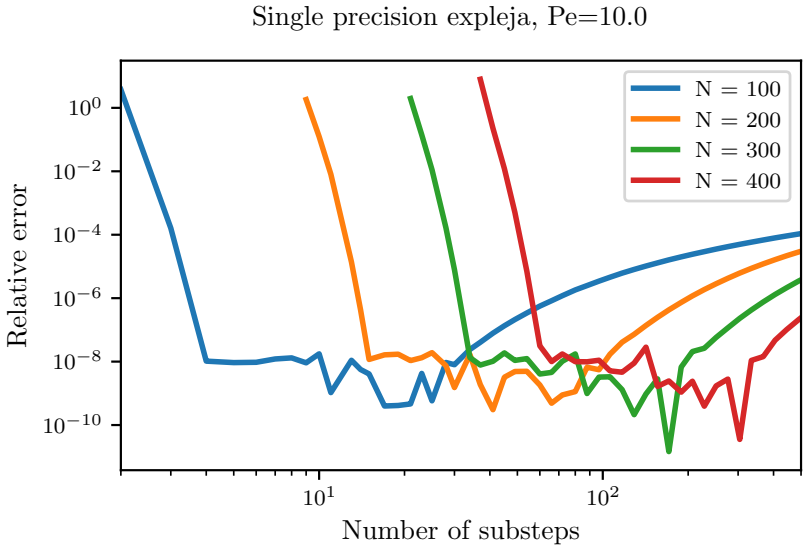


Figure 1: Approximation of $e^{0.1A}u_0$ using single precision `expleja` for a fixed interpolation degree $m = 100$ and varying number of substeps s. The relative error is measured in the Eucledian norm. The reference solution was computed using the double precision `expleja` algorithm.

In order to measure the relative strength of advection compared to diffusion we employ the Péclet number $\mathrm{Pe} = \frac{b}{a}$. The solution of the differential equation is given by $e^{0.1A}u_0$, which can be approximated using the Leja method, as shown in Figure 1. For the matrix-free case we need to compute the spectral radius of A, which can be done using the power method.

4.1 Analysis of the power method

We investigate the rate of convergence of the power method to the largest eigenvalue (with respect to the modulus) λ_{max} of A. For our analysis we assume periodic boundary conditions

$$A = \frac{a}{h^2}\begin{bmatrix} -2 & 1 & 0 & \cdots & 0 & 1 \\ 1 & -2 & 1 & & & 0 \\ 0 & 1 & \ddots & \ddots & & \vdots \\ \vdots & & \ddots & \ddots & 1 & 0 \\ 0 & & & 1 & -2 & 1 \\ 1 & 0 & \cdots & 0 & 1 & -2 \end{bmatrix} + \frac{b}{h}\begin{bmatrix} -1 & 1 & 0 & \cdots & \cdots & 0 \\ 0 & -1 & 1 & & & \vdots \\ \vdots & & \ddots & \ddots & & \vdots \\ \vdots & & & \ddots & 1 & 0 \\ 0 & & & & -1 & 1 \\ 1 & 0 & \cdots & \cdots & 0 & -1 \end{bmatrix}.$$

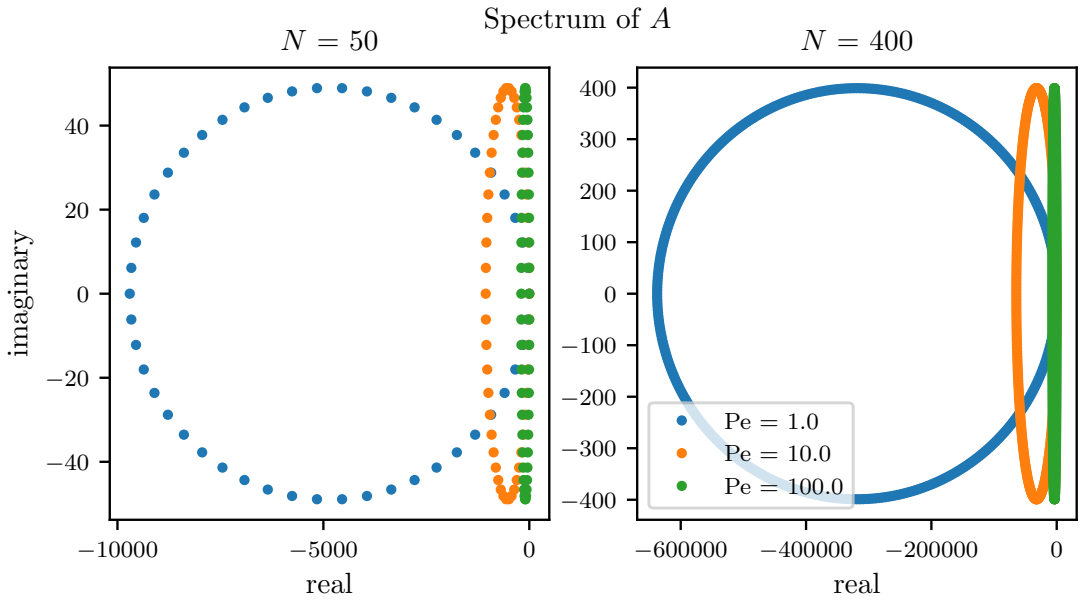


Figure 2: The spectrum of A. We assume periodic boundary conditions.

Consider the discrete Fourier basis

$$v_j = \frac{1}{\sqrt{N}} \begin{bmatrix} e^{i\frac{2\pi}{N}j0} \\ e^{i\frac{2\pi}{N}j1} \\ \vdots \\ e^{i\frac{2\pi}{N}j(N-1)} \end{bmatrix}, \quad j \in 0 \dots N-1.$$

Each v_j is an eigenvector of A

$$Av_j = \lambda_j v_j,$$

where the eigenvalues λ_j are given by

$$\begin{aligned} \lambda_j &= \frac{a}{h^2}\left(e^{i\frac{2\pi}{N}j} - 2 + e^{-i\frac{2\pi}{N}j}\right) - \frac{b}{h}\left(e^{i\frac{2\pi}{N}j} - 1\right) \\ &= -\left(\frac{4a}{h^2} - \frac{2b}{h}\right)\sin^2\left(\frac{\pi j}{N}\right) + i\frac{b}{h}\sin\left(\frac{2\pi j}{N}\right), \end{aligned}$$

see Figure 2. We restrict our analysis to the case $a = 0$ and $b = 1$, i.e.

$$\lambda_j = -\frac{4}{h^2}\sin^2\left(\frac{\pi j}{N}\right).$$

The modulus is maximized for $j = \lfloor N/2 \rfloor$, i.e. $\lambda_{max} = \lambda_{\lfloor N/2 \rfloor}$. From now on we study the behaviour of $v = \frac{1}{N}\sum_{j=0}^{N-1} v_j$, the normalized sum of all eigenvectors. Let $n \in \mathbb{N}$

be the number of power iterations. We begin our analysis with the following auxiliary calculation

$$\|A^n v\|_2 = \sqrt{\frac{1}{N}\sum_{j=0}^{N-1} |\lambda_j|^{2n}} = \frac{2^{2n}}{h^{2n}}\sqrt{I_{N,n}},$$

where

$$I_{N,n} = \frac{1}{N}\sum_{j=0}^{N-1} \sin^{4n}\left(\frac{\pi j}{N}\right).$$

The first equality holds since all eigenvectors are orthogonal. We interpret the sum of sine functions as an integral approximated by the composite trapezoidal rule. Using the nodes $\frac{k}{N}$ for $k = 0 \dots N$ we have

$$\int_0^1 \sin^{4n}(\pi x) \approx \frac{1}{N}\sum_{j=1}^{N-1} \sin^{4n}\left(\frac{\pi j}{N}\right) = I_{N,n}.$$

Since sin^{4n} is a trigonometric polynomial of degree $4n$ the trapezoidal rule is exact for $N > 4n$. For readers unfamiliar with this property we refer to [10, Corollary 3.3]. The approximation is even better than the theory predicts. We numerically verify that the trapezoidal rule is exact for $N + 1 > n$. We underestimate the largest eigenvalue λ_{max} by a factor of

$$\frac{\|A^{n+1}v\|_2}{\|A^n v\|_2}\frac{1}{|\lambda_{max}|} = \frac{\frac{2^{2n+2}}{h^{2n+2}}\sqrt{I_{N,n+1}}}{\frac{2^{2n}}{h^{2n}}\sqrt{I_{N,n}}}\frac{1}{|\lambda_{max}|} = \sqrt{\frac{I_{N,n+1}}{I_{N,n}}}\sin^{-2}\left(\frac{\pi}{N}\left\lfloor\frac{N}{2}\right\rfloor\right).$$

using the power method. For large N we have $\frac{4a}{|h^2\lambda_{max}|} \approx 1$. In order to continue our analysis and get some asymptotic bounds we interpret the n-th root the summands of $I_{N,n}$ as a function of x

$$S_N(x) = \left(1 - \frac{hb}{2a}\right)^2 \sin^4(\pi x) + \frac{hb}{4a}\sin^2(2\pi x).$$

In the limit $S_N(\cdot)$ converges to $\sin^4(\pi\cdot)$. Furthermore we have uniform convergence since for all for $x \in [0, 1]$

$$|S_N(x) - \sin^4(\pi x)| = \left|\frac{hb}{2a}\left(-2 + \frac{hb}{2a}\right)\sin^4(\pi x) + \frac{hb}{4a}\sin^2(2\pi x)\right| \le \frac{hb}{a}\sin^4(\pi x) \le \frac{hb}{a},$$

if N is sufficiently large. Therefore, if we keep n fixed, we can take the limit

$$\lim_{N\to\infty} I_{N,n} = \lim_{N\to\infty}\frac{1}{N}\sum_{j=0}^{N-1}(S_N(x_j))^n = \int_0^1 \sin^{4n}(\pi x)dx = \frac{1}{\pi}\,\mathrm{B}(2n + 0.5, 0.5),$$

where B is the beta function. Finally we can establish an asymptotic bound

$$\begin{aligned}
\lim_{N\to\infty} \frac{I_{N,n+1}}{I_{N,n}} &= \frac{\mathrm{B}(2n+2.5,0.5)}{\mathrm{B}(2n+0.5,0.5)} \\
&= \frac{\Gamma(2n+1)\Gamma(2n+2.5)}{\Gamma(2n+3)\Gamma(2n+0.5)} \\
&= \frac{(2n)!}{(2n+2)!}\frac{\Gamma(2n+2.5)}{\Gamma(2n+0.5)} \\
&= \frac{(2n)!}{(2n+2)!}\frac{2^{4n}\sqrt{\pi}}{2^{4n+4}\sqrt{\pi}}\frac{(4n+4)!}{(4n)!}\frac{(2n)!}{(2n+2)!} \\
&= \frac{(4n+4)(4n+3)(4n+2)(4n+1)}{16(2n+2)^2(2n+1)^2} \\
&= \frac{(4n+3)(4n+1)}{(4n+4)(4n+2)} \\
&= \left(1-\frac{1}{4n+4}\right)\left(1-\frac{1}{4n+2}\right)
\end{aligned}$$

For the fourth equality we applied the duplication formula for the gamma function. All in all we underestimate the largest eigenvalue λ_{max} by a factor of

$$\lim_{N\to\infty} \frac{\|A^{n+1}v\|_2}{\|A^n v\|_2}\frac{1}{|\lambda_{max}|} = \sqrt{\left(1+\frac{1}{4n+4}\right)\left(1-\frac{3}{4n+2}\right)} \approx 1-\frac{1}{4n+3}$$

at the limit $N \to \infty$.

4.2 Experiments for the power method

As discussed in Section 3 we need to estimate the largest eigenvalue λ_{max} of A for the matrix-free `expleja` algorihm. However, we can only guarantee convergence if we overestimate λ_{max}. Therefore we have to include a safety factor sf by which we multiply the output of the power method.

We solve the initial value problem 3 for a fixed number of substeps s with fixed interpolation degree $m = 100$ per substep, while still allowing for an early termination of the interpolation if the error is small enough. See Figure 3. While the algorithm did not converge for all combinations of N, sf and Péclet numbers we tried, we observe that $n = 4$ and $\mathrm{sf} = 1.1$ is a robust choice independent of N for our initial vector u_0.
For all future experiments we choose $\mathrm{sf} = 1.1$ and $n = 4$ for the matrix-free `expleja` algorithm. As an additional optimization we save the approximated eigenvector corresponding to λ_{max} when integrating over multiple time steps. We use it as the initial vector for the power method for the next time step. Furthermore we terminate the power method early if the relative change of the approximated eigenvalue is smaller than 0.01.

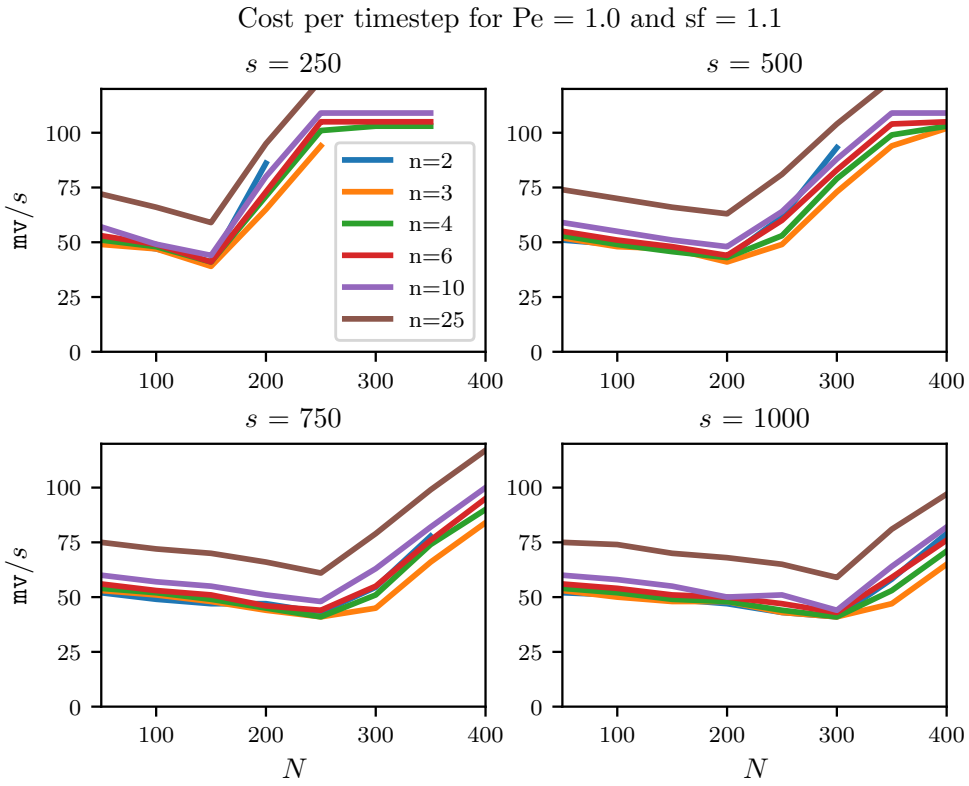


Figure 3: Space dimension N vs costs `mv` per timestep s for matrix-free single precision `expleja`. The interpolation degree m is fixed to 100. The Newton interpolation can still terminate early resulting in smaller numbers of `mv`$/s$. The number of power iterations are denoted by n. For this experiment we chose Pe $= 1.0$ and sf $= 1.1$. Results are only shown if they achieve single precision.

5 Matrix-free Leja based exponential integrators

Exponential integrators are a class of numerical integrators which excel at solving stiff differential equations. Unlike most numerical ordinary differential equation (ODE) solvers their construction is based on the variation-of-constants formula. Consider the semilinear initial value problem

$$\begin{aligned} \partial_t u &= F(u) = Au + g(u) \\ u(0) &= u_0 \end{aligned} \tag{4}$$

where $A = \partial_u F$ and $g(u) = F(u) - Au$ is the linear and nonlinear part of F respectively. The solution of the ODE is given by the variation-of-constants formula

$$u(t) = e^{At}u_0 + \int_0^t e^{(t-\tau)A} g(u(\tau))d\tau.$$

Similar to Runge-Kutta methods we replace the integrand with a polynomial approximation. Unlike Runge-Kutta methods we leave the matrix exponential untouched and only replace g. The most well-known Rosenbrock-type exponential integrator, the exponential Rosenbrock-Euler method, can be obtained by using the left hand rule. By replacing g with $g(u_0)$ we get

$$u(t) \approx e^{At}u_0 + \int_0^t e^{(t-\tau)A}g(u_0)d\tau = e^{At}u_0 + \varphi_1(tA)g(u_0),$$

where $\varphi_1(z) = \frac{e^z-1}{z}$. The exponential Rosenbrock-Euler method is of order 2 and is exact for linear problems, i.e. if g(u)=0. We will refer to it as `exprb2` in Section 6.

5.1 Higher order Rosenbrock methods

Exponential Rosenbrock methods are a special class of exponential integrators which efficiently solve semi-linear problems (4). For a given time step size τ the numerical solution u_1 is given by

$$\begin{aligned} U_i &= e^{c_i\tau A}u_0 + \tau\sum_{j=1}^{i-1} a_{ij}(\tau A)g(U_j), \\ u_1 &= e^{\tau A}u_0 + \tau\sum_{i=1}^{s} b_i(\tau A)g(U_i), \end{aligned} \tag{5}$$

where $s \in \mathbb{N}$ and a_{ij}, b_i are matrix functions. The numerical scheme can be represented as a Butcher tableau

$$\begin{array}{c|cccc} c_1 & & & & \\ c_2 & a_{21}(\tau A) & & & \\ \vdots & \vdots & \ddots & & \\ c_s & a_{s1}(\tau A) & \dots & a_{s,s-1}(\tau A) & \\ \hline & b_1(\tau A) & \dots & b_{s-1}(\tau A) & b_s(\tau A) \end{array}.$$

The functions a_{ij} and b_i are typically given as linear combinations of the φ_k-functions, which in turn are recursively defined as

$$\varphi_{k+1}(z) = \frac{\varphi_k(z)-1}{z}, \quad \varphi_0(z) = e^z, \quad k \in \mathbb{N}.$$

For example consider the embedded method

$$\begin{array}{c|ccc} 0 & & & \\ \frac{1}{2} & \frac{1}{2}\varphi_1(\frac{1}{2}\cdot) & & \\ 1 & 0 & \varphi_1 & \\ \hline \texttt{exprb3} & \varphi_1 - 14\varphi_3 & 16\varphi_3 & -2\varphi_3 \\ \texttt{exprb4} & \varphi_1 - 14\varphi_3 + 36\varphi_4 & 16\varphi_3 - 48\varphi_4 & -2\varphi_3 + 12\varphi_4 \end{array},$$

i.e.

$$
\begin{aligned}
U_1 &= u_0, \\
U_2 &= e^{\frac{\tau}{2}A}u_0 + \tau\varphi_1\left(\frac{\tau}{2}A\right)g(U_1) = u_0 + \frac{\tau}{2}\varphi_1\left(\frac{\tau}{2}A\right)F(u_0), \\
U_3 &= e^{\tau A}u_0 + \tau\varphi_1(\tau A)g(U_2) = e^{\tau A}F(U_2), \\
\tilde{u}_1 &= e^{\tau A}u_0 + (\varphi_1 - 14\varphi_3)(\tau A)g(u_0) + 16\varphi_3(\tau A)g(U_2) - 2\varphi_3(\tau A)g(U_3), \\
\hat{u}_1 &= e^{\tau A}u_0 + (\varphi_1 - 14\varphi_3 + 36\varphi_4)(\tau A)g(u_0) + (16\varphi_3 - 48\varphi_4)(\tau A)g(U_2) \\
&\quad + (-2\varphi_3 + 12\varphi_4)(\tau A)g(U_3),
\end{aligned}
$$

where $\tilde{u}_1$ and $\hat{u}_1$ is the numerical solution given by `exprb3` and `exprb4` respectively. This scheme is known as `exprb43` [6, Example 2.24]. It uses `exprb3` as a third-order estimator for its fourth-order method `exprb4`. Both integrators are well suited for numerical computations since all internal stages can be cheaply computed using the exponential Euler method.
Under the simplifying assumptions

$$
\sum_{j=1}^{s} b_j = \varphi_1, \quad \sum_{j=1}^{s} a_{ij} = c_i\varphi_1(c_i\cdot)
$$

for $1 \le i \le s$ the scheme (5) can be expressed as

$$
\begin{aligned}
U_i &= u_0 + c_i\tau\varphi_1(c_i\tau A)F(u_0) + \tau\sum_{j=2}^{i-1} a_{ij}(\tau A)D_j, \\
D_j &= g(U_j) - g(u_0), \quad 2 \le j \le s, \\
u_1 &= u_0 + \tau\varphi_1(\tau A)F(u_0) + \tau\sum_{i=2}^{s} b_i(\tau A)D_i.
\end{aligned} \tag{6}
$$

The main advantage of this reformulation lies in the fact that the norm of all D_j is expected to be small. This can be exploited by the Leja method by allowing an early termination of the Newton interpolation.
For an efficient implementation of exponential Rosenbrock integrators it is crucial to compute only a single action of a matrix function per stage U_i and for solution u_1. Since the most frequently employed methods depend on linear combinations of φ_k-functions this can be done using the matrix exponential function.

5.2 Computing the action of the φ-functions

Exponential integrators rely on the efficient computation of φ_k-functions. In the matrix case $A \in \mathbb{C}^{N\times N}$ this can be done by slightly expanding A, see [2, Theorem 2.1].
Let $V = [V_p \dots V_2, V_1] \in \mathbb{C}^{N\times p}$, $u \in \mathbb{C}^{N\times 1}$, $\tau \in \mathbb{C}$ and

$$
\tilde{A} = \begin{bmatrix} A & V \\ 0 & J \end{bmatrix}, \quad J = \begin{bmatrix} 0 & I_{p-1} \\ 0 & 0 \end{bmatrix},
$$

where I_n is the $n \times n$ identity matrix. Let e_n denote the n-th $p \times 1$ unity vector. Then

$$\begin{bmatrix} I_N & 0 \end{bmatrix} e^{\tau \tilde{A}} \begin{bmatrix} u \\ e_j \end{bmatrix} = e^{\tau A} u + \sum_{k=1}^{j} \tau^k \varphi_k(\tau A) V_{p-j+k}, \quad j \in \{1, \dots, p\}.$$

In particular for $j = p$ we have

$$\begin{bmatrix} I_N & 0 \end{bmatrix} e^{\tau \tilde{A}} \begin{bmatrix} u \\ e_p \end{bmatrix} = e^{\tau A} u + \sum_{k=1}^{p} \tau^k \varphi_k(\tau A) V_k.$$

This formulation can be directly applied to each stage in (6) assuming a_{ij} and b_j are linear combinations of φ_k-functions. Therefore for each stage only a single action of an expanded matrix exponential has to be evaluated. In total this has to be done s times for an exponential Rosenbrock method with s stages.
For a matrix-free implementation of $\tilde{A}$ given an operator A we can simply compute the action of $\tilde{A}$ as follows

$$\tilde{A} \begin{bmatrix} v \\ w \end{bmatrix} = \begin{bmatrix} Av \\ 0 \end{bmatrix} + \begin{bmatrix} Vw \\ Jw \end{bmatrix}, \quad v \in \mathbb{C}^{N \times 1}, w \in \mathbb{C}^{p \times 1}.$$

The Leja method only relies on matrix-vector multiplications with $\tilde{A}$ and therefore the explicit knowledge of A is not required. To summarize, an efficient matrix-free implementation of exponential Rosenbrock-methods can be achieved using the Leja method. In particular `exprb3` and `exprb4` can be evaluated by computing three actions of matrix exponentials.

6 Numerical experiments

In this section we will apply matrix-free exponential Rosenbrock integrators to multiple advection-diffusion-reaction equations. All experiments are conducted in Python 3.7 [11] with NumPy 1.18.1 [12] and SciPy 1.4.1 [12]. We use an AMD Ryzen 7 2700 Processor on a Windows machine. We consider d-dimensional nonlinear variants of the initial value problem in section 4

$$\begin{aligned} \partial_t u &= F(u) && \text{with} \quad t \in [0, 0.1], \\ u_0(x) &= e^{-80 \cdot (\|x\|_2^2 - 0{,}45)^2} && \text{with} \quad x \in [0, 1]^d. \end{aligned}$$

For all experiments we assume Dirichlet boundary conditions.
We compare the behavior of these methods with other matrix-free ODE solvers.

Crank-Nicolson method: We refer to the Crank-Nicolson method of order 2 as `cn2`. In our implementation of `cn2` we use the Newton-Raphson method to solve the nonlinear system of equations. We treat the Jacobian-vector product $v \mapsto F'(u)v$ as a linear operator. For the resulting system of linear equations we use the SciPy package `scipy.sparse.linalg.gmres`. For all experiments the relative tolerance is set to

`tol`$/N_\tau$, where N_τ is the total number of time steps used for solving the ODE. This choice guarantees that the sum of errors made by `gmres` is always lower than our specified tolerance `tol`. No preconditioner was used for `gmres`. The Crank-Nicolson method is unconditionally stable and therefore does not have to satisfy the Courant-Friedrichs-Lewy (CFL) conditions for the differential equations we investigate in our experiments.

Explicit Runge-Kutta method: We refer to the explicit midpoint method of order 2 as `rk2` and refer to the classical Runge-Kutta method of order 4 as `rk4`. No explicit Runge-Kutta method is A-stable. Therefore small time step sizes have to be chosen when solving stiff differential equations. In our experiments this is the case when the considered differential equation is diffusion-dominated.

Exponential Rosenbrock methods: We refer to the exponential Rosenbrock methods of order 2, 3 and 4 discussed in 5 as `exprb2`, `exprb3` and `exprb4` respectively. The action of the matrix exponential is approximated with the matrix-free `expleja` algorithm. At each time step we compute the optimal interpolation degree and substep parameter according to (2). The maximal interpolation degree is set to 100. Note that the total number of matrix-vector multiplication per time step can still exceed 100 since we have to compute the spectral radius. This typically happens for $s = 1$. At each time step we compute the spectral radius using the power method with at most four power iterations.

Our goal is to investigate the respective computational costs of these methods while achieving a prescribed relative tolerance `tol`.

6.1 Cost analysis

Let $N_\tau \in \mathbb{N}$ be the total number of time steps. We investigate the cost per time step incurred by each integrator in terms of function evaluations. Both `rk2` and `rk4` require one function evaluation for each stage. For `cn2` and all exponential integrators the cost analysis is much more involved. In addition to function evaluations these methods use expensive subroutines which incur costs mainly depending on the number of Jacobian-vector products. For $v \in \mathbb{R}^N$ the directional derivative can be approximated using two function evaluations

$$F'(u)v \approx \frac{F(u + \epsilon v) - F(u)}{\epsilon} \quad \text{with} \quad \epsilon > 0.$$

However, $F(u)$ only needs to be computed once per time step. Therefore, after evaluating and storing $F(u)$ the directional derivative $F'(u)v$ costs approximately one function evaluation.

6.2 One-dimensional advection-diffusion equation

Let $\alpha, \beta, \geq 0$ and

$$F(u) = \alpha\partial_x((u + 1)\partial_x u) + \beta\partial_x(u^2) + u(u - 0.5).$$

With α and β we control the relative strength of the diffusion compared to the advection respectively. The differential equation can be rewritten using the product rule. The This leads to

$$F(u) = \alpha((u+1)\partial_{xx}u + (\partial_x u)^2) + 2\beta u\partial_x(u) + u(u-0.5)$$

The total derivative is given by

$$\partial_u F = \alpha((u+1)\partial_{xx} + (\partial_{xx}u)\,\mathrm{Id} + (\partial_x u)\,\mathrm{Id} + u\partial_x) + 2\beta(\partial_x u\,\mathrm{Id} + u\partial_x) + (2u-0.5)\,\mathrm{Id},$$

where Id is the identity function. We will treat $A = \partial_u F$ as a linear operator for all calculations. All instances of ∂_{xx} are discretized with centred finite differences. By considering only the terms depending ∂_{xx} we get the modified problem

$$\partial_t u = \alpha(u+1)\partial_{xx}u.$$

Since num

$$\tau \leq \frac{h^2}{2\alpha} =: C_{dif}$$

restricts the maximum possible step size for explicit Runge-Kutta methods. This constraint can be seen in Figure 4 on the bottom row for the original problem.

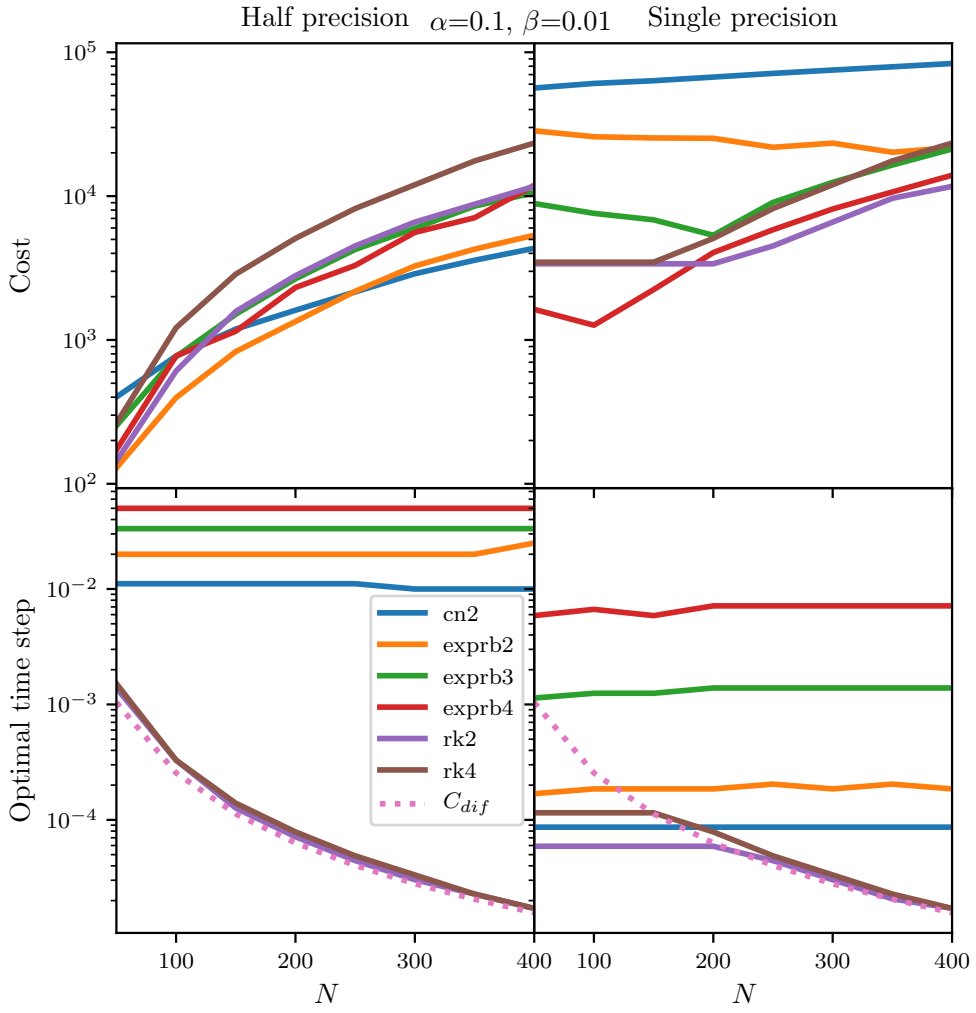


Figure 4

6.3 Two-dimensional advection-diffusion equation

Let $\alpha, \beta, \geq 0$ and

$$\partial_t u = F(u) = \alpha \nabla (u \nabla_x u) + \beta \partial_x (u^2) + u(u - 0.5)$$

With α and β we control the relative strength of the diffusion compared to the advection respectively. The differential equation can be rewritten using $\partial_x(u\partial_x u) = \partial_{xx} u + u\partial_x u$ and $\partial_x(u^2) = 2u\partial_x u$ This leads to

$$F(u) = \alpha(\partial_{xx} u + u\partial_x u) + 2\beta u \partial_x(u) + u(u - 0.5)$$

All instances ∂_{xx} are discretized with centred finite differences. We approximate ∂_x using the upwind scheme for numerical stability.

6.4 Experiment 1: Linear advection diffusion equation

Consider the one-dimensional advection-diffusion equation

$$\begin{aligned} \partial_t u &= a\partial_{xx} u + b\partial_x u \quad a, b \geq 0 \\ u_0(t) &= e^{-80\cdot(t-0{,}45)^2} \quad t \in [0, 0.1] \end{aligned}$$

with homogeneous Dirichlet boundary conditions on the domain $\Omega = [0, 1]$. For a fixed $N \in \mathbb{N}$ we approximate the diffusive part with second-order central differences on an equidistant grid with grid size $h = \frac{1}{N}$ and grid points $x_i = ih$, $i = 0 \ldots, N$.

$$\partial_{xx} u(x_i) = \frac{u(x_{i+1}) - 2u(x_i) + u(x_{i-1})}{h^2} + \mathcal{O}(h^2)$$

In order to limit numerical instabilities we discretize the advective part with forward differences, similar to the upwind scheme.[1]

$$\partial_x u(x_i) = \frac{u(x_{i+1}) - u(x_i)}{h} + \mathcal{O}(h)$$

The resulting system of ordinary differential equation is given by

$$\partial_t u = Au.$$

Some eigenvalues of A can have an extremely large negative real part. Therefore, since no explicit Runge-Kutta method is A-stable, this imposes very stingend conditions on the time step size τ for rk2 and rk4. We will refer to the Courant-Friedrich-Lewy (CFL) conditions imposed by the advective and diffusive part of A respectively by C_{adv} and C_{dif}.

$$C_{adv} = \frac{b\tau}{h} \leq 1, \quad C_{dif} = \frac{a\tau}{h^2} \leq \frac{1}{2}$$

In our case the problem is fully linear and therefore `exprb2` simplifies to the computation of the action of the matrix exponential function with the Leja method. We write `expleja` for the single precision Leja method approximation. The reference solution was computed with double precision and therefore uses different nodes.

In order to keep the solution from vanishing, we fix $b = 1$ and only consider coefficients $a \in [0, 1]$. The advection-diffusion ratio scaled by the grid size h is represented by the grid Péclet number

6.5 Experiment 2: 1D Nonlinear advection diffusion equation

$$\partial_t u = \alpha\partial_x((u + 1)\partial_x u) + \partial_x(u^2) + u(u - 0.5)$$

We discretize, solve again with rk2, rk4, cn2 and exprb2.

[1] Maybe create a seperate section on hybrid difference schemes? There we can also analyze the resulting matrix A itself and plot the eigenvalues. I need sources for that though.

7 Appendix

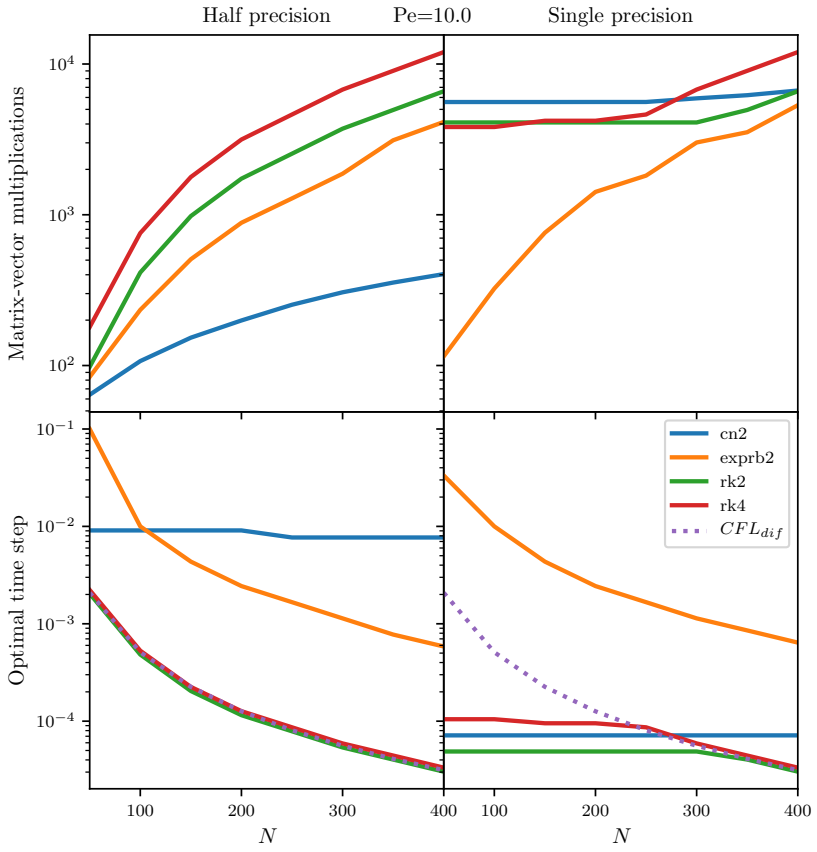
Matrix analysis, Horn and Johnson, Lemma 5.6.10

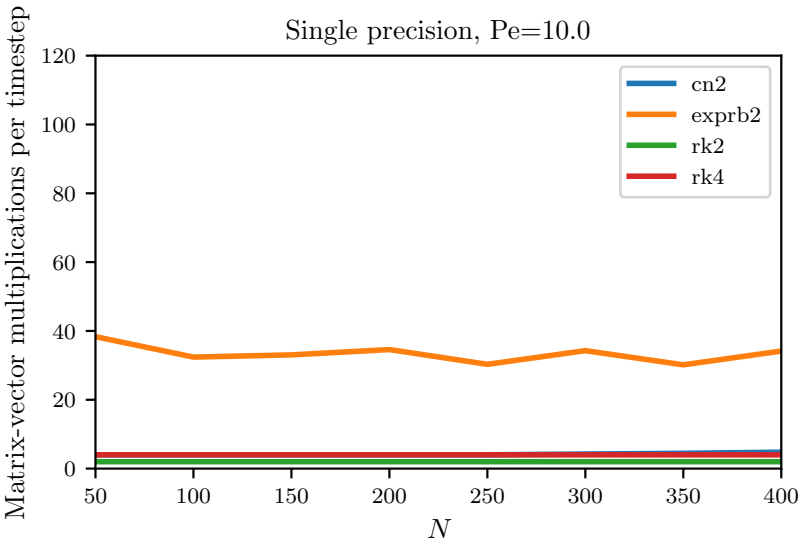
$$\rho(A) = \inf\{\|A\| : \|\cdot\| \text{ is an induced operator norm}\}$$

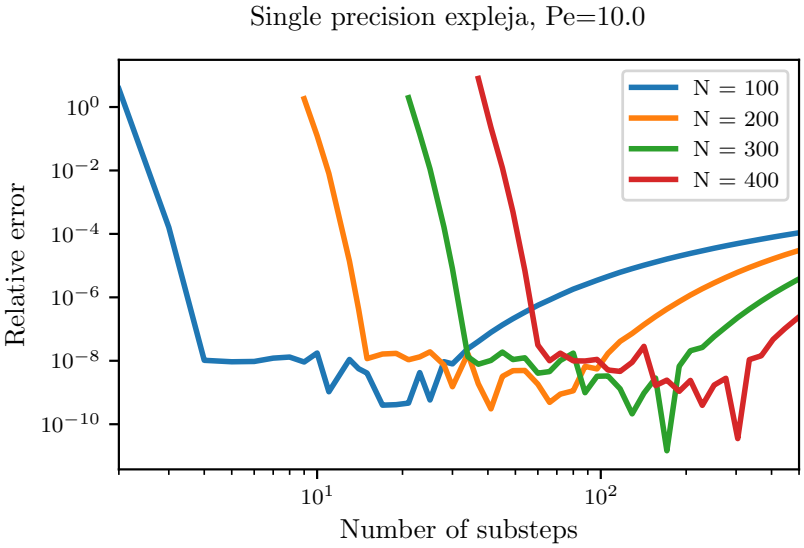
7.1 Experiment Linear

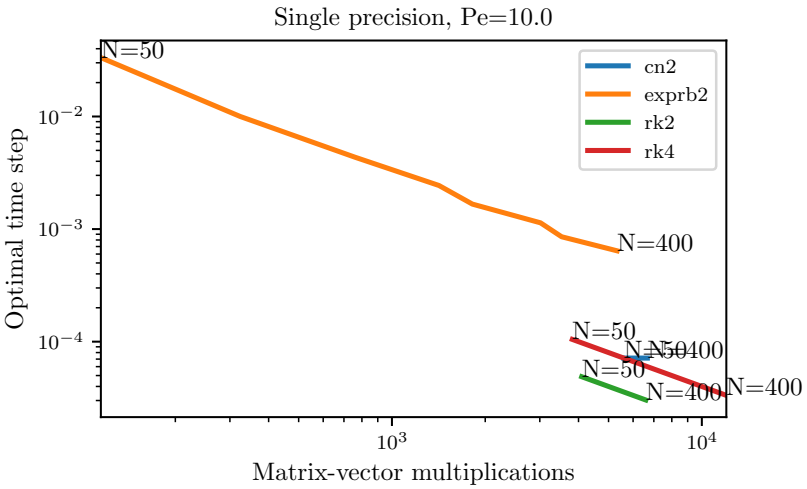
8 TODO

- Section 3, adapt introduction with RAM constraints, not cache
- Section 4.1 Explain $v = \sum_{j=1}^{N} v_j$, Gleichverteilung aller EV
- Section 4.1 Consider special case $b = 0$ for analysis.
- Section 4.1 What is λ_{max}?



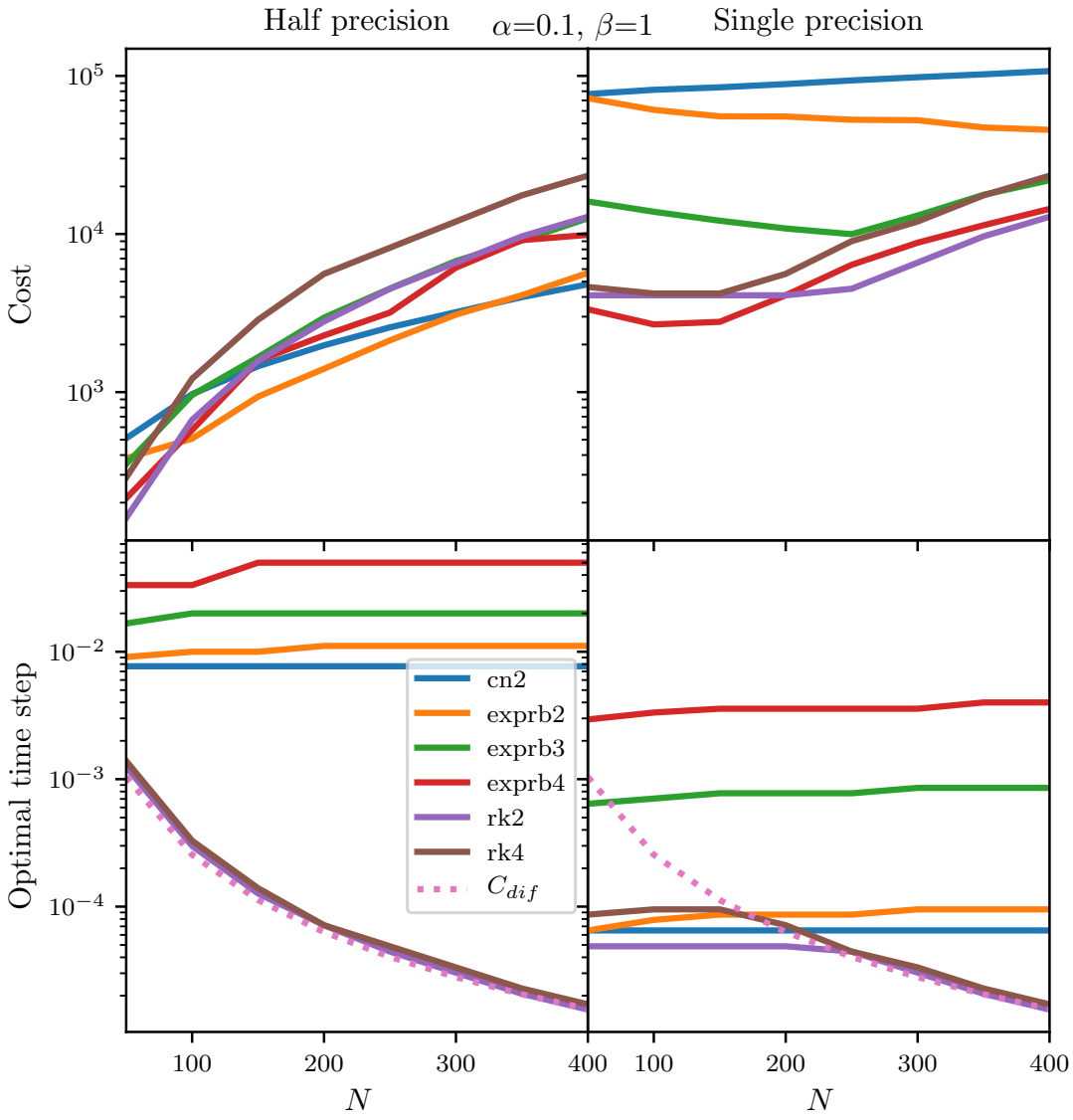

Single precision, Pe=10.0
Matrix-vector multiplications per timestep
120
100
80
60
40
20
0
50
100
150
200
250
300
350
400
N
cn2
exprb2
rk2
rk4

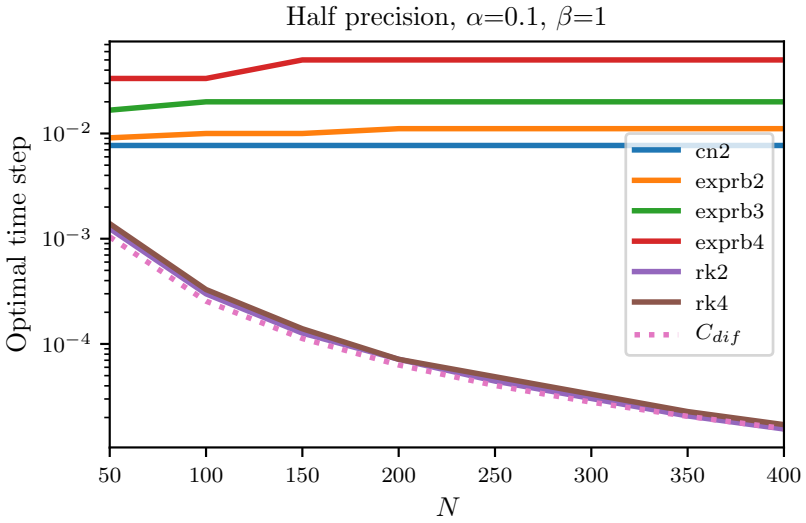

Single precision expleja, Pe=10.0
Relative error
10^{0}
10^{-2}
10^{-4}
10^{-6}
10^{-8}
10^{-10}
10^{1}
10^{2}
Number of substeps
N = 100
N = 200
N = 300
N = 400

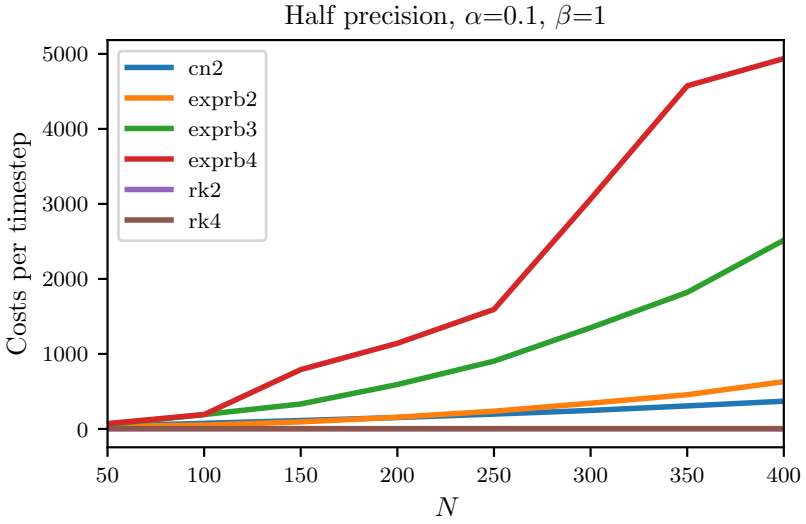

Single precision, Pe=10.0
cn2
exprb2
rk2
rk4
N=50
N=400
N=50
N=400
N=50
N=400
N=400
Optimal time step
10^{-2}
10^{-3}
10^{-4}
10^{3}
10^{4}
Matrix-vector multiplications

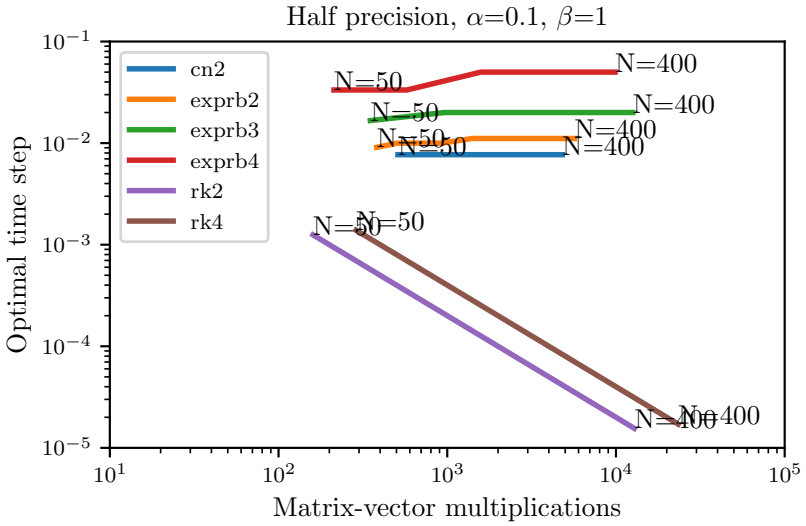
8.1 Experiment Nonlinear 1D

1, half, =0.1, =1



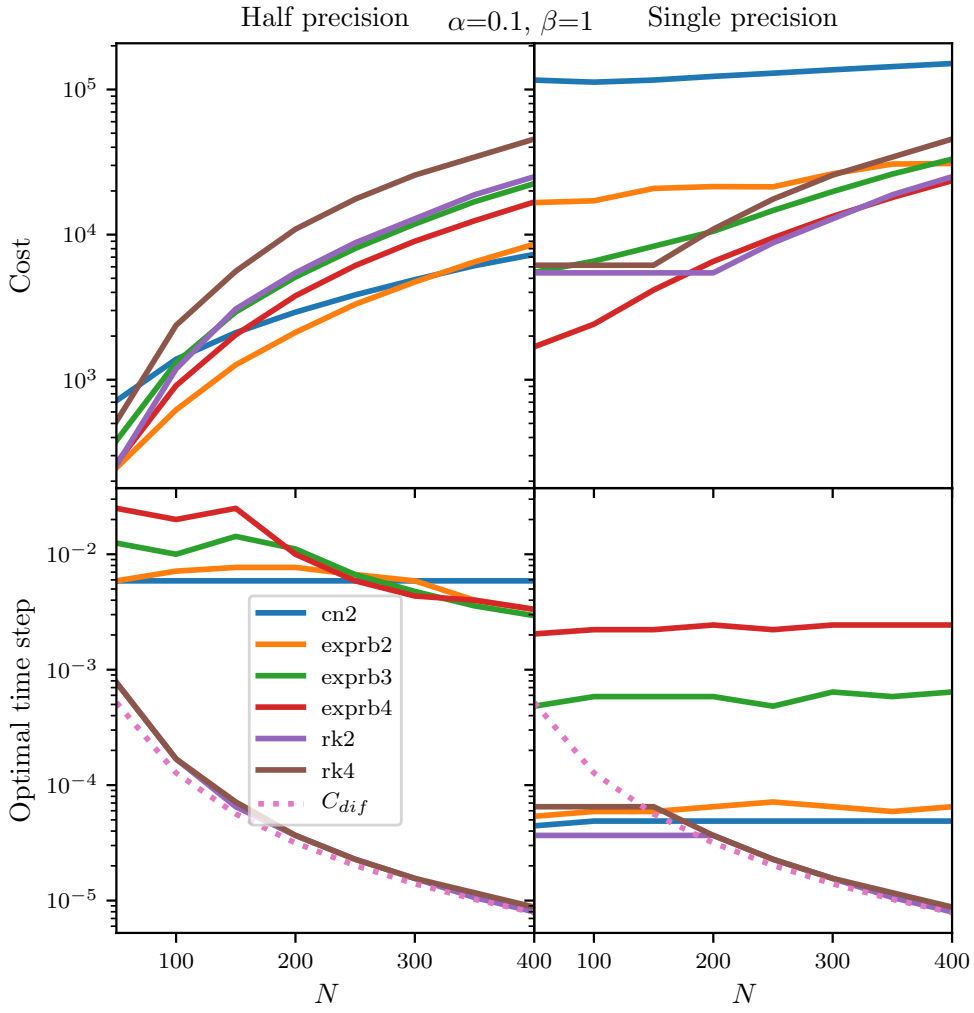

Half precision, α=0.1, β=1
Optimal time step
10^{-2}
10^{-3}
10^{-4}
cn2
exprb2
exprb3
exprb4
rk2
rk4
C_{dif}
50
100
150
200
250
300
350
400
N

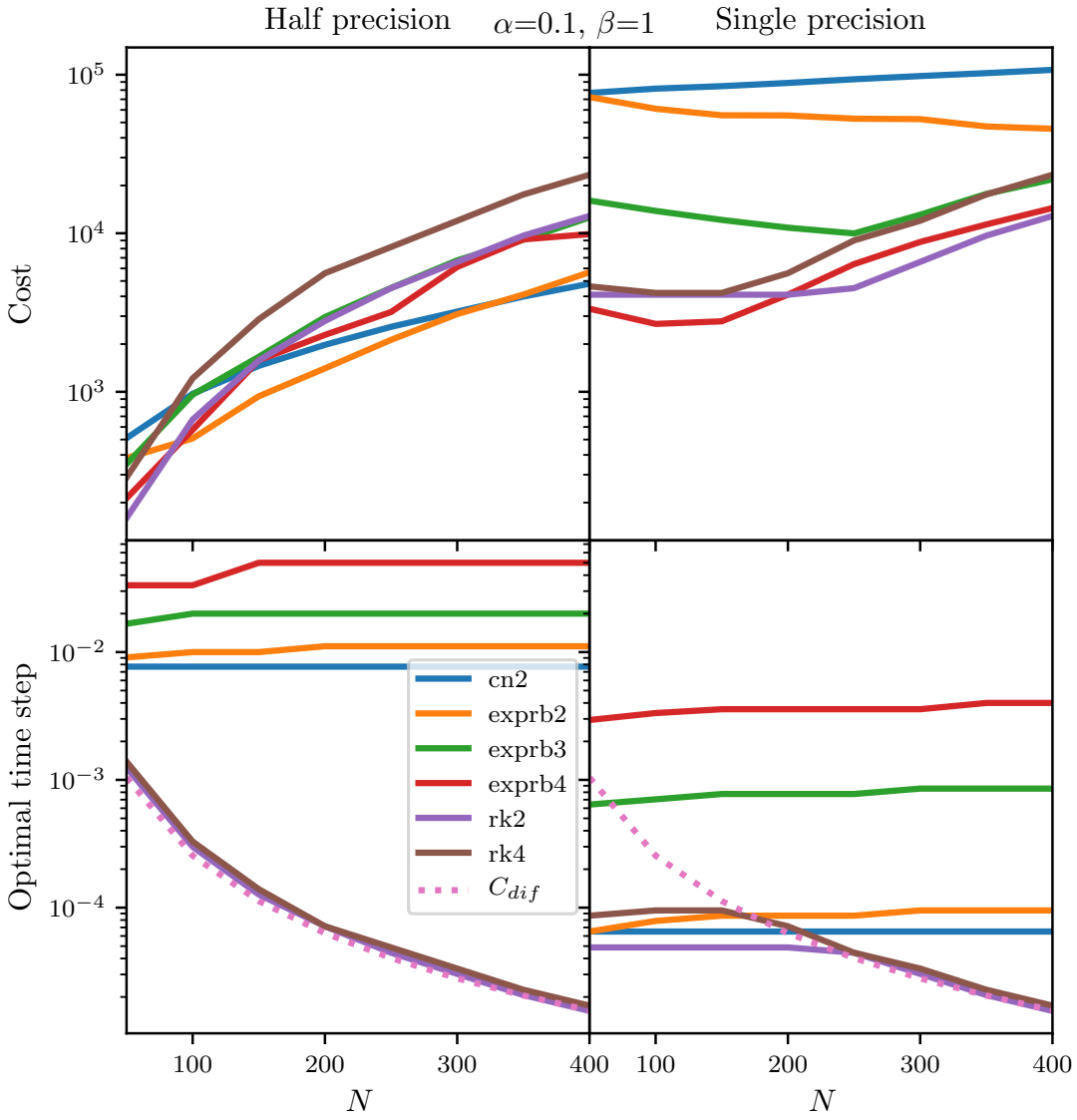

Half precision, α=0.1, β=1
Costs per timestep
5000
4000
3000
2000
1000
0
cn2
exprb2
exprb3
exprb4
rk2
rk4
50
100
150
200
250
300
350
400
N

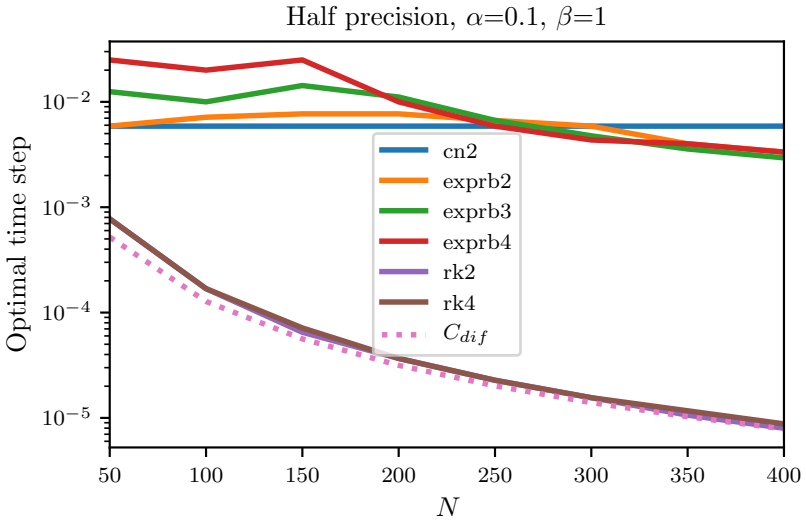


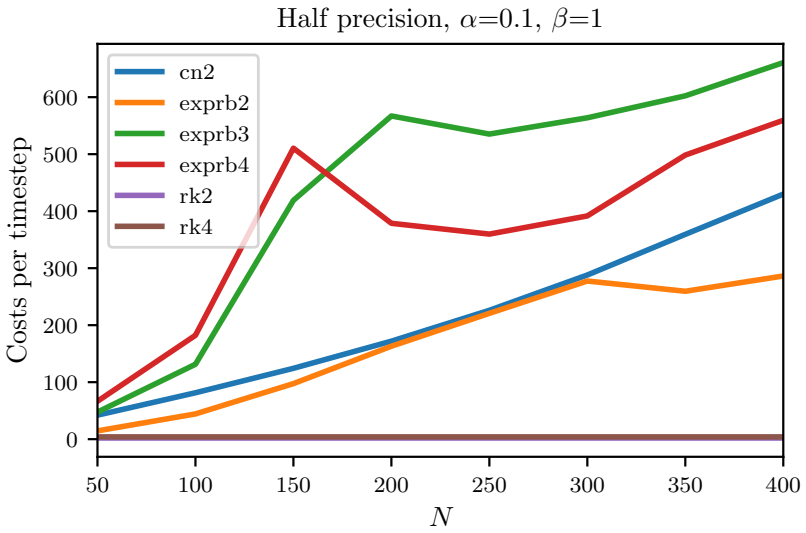
8.2 Experiment Nonlinear 2D

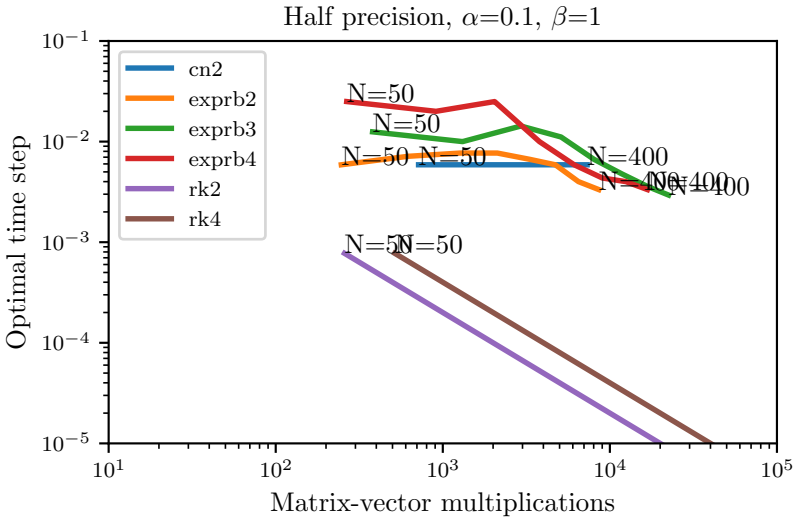
1, half, =0.1, =1


Half precision
α=0.1, β=1
Single precision
Cost
Optimal time step
10^5
10^4
10^3
10^{-2}
10^{-3}
10^{-4}
10^{-5}
cn2
exprb2
exprb3
exprb4
rk2
rk4
C_{dif}
100
200
300
400
N


Half precision
α=0.1, β=1
Single precision
Cost
10^5
10^4
10^3
Optimal time step
10^{-2}
10^{-3}
10^{-4}
cn2
exprb2
exprb3
exprb4
rk2
rk4
C_{dif}
100
200
300
400
N


Half precision, α=0.1, β=1
Optimal time step
10^{-2}
10^{-3}
10^{-4}
10^{-5}
cn2
exprb2
exprb3
exprb4
rk2
rk4
C_{dif}
50
100
150
200
250
300
350
400
N


Half precision, α=0.1, β=1
Costs per timestep
600
500
400
300
200
100
0
cn2
exprb2
exprb3
exprb4
rk2
rk4
50
100
150
200
250
300
350
400
N


Half precision, α=0.1, β=1
cn2
exprb2
exprb3
exprb4
rk2
rk4
N=50
N=50
N=50
N=50
N=400
N=400
N=400
N=50
N=50
Optimal time step
Matrix-vector multiplications

References

[1] M. Caliari, A. Ostermann. Implementation of exponential Rosenbrock-type integrators, Applied Numerical Mathematics 59 (2009), 568-581.

[2] A. Al-Mohy, N. Higham. Computing the action of the matrix exponential, with an application to exponential integrators, SIAM Journal on Scientific Computing 33 (2011), 488-511.

[3] L. Reichel. Newton interpolation at Leja points, BIT Numerical Mathematics 30 (1990), 332-346.

[4] M. Caliari, M. Vianello, L. Bergamaschi. Interpolating discrete advection-diffusion propagators at Leja sequences, Journal of Computational and Applied Mathematics 172 (2004), 79-99.

[5] M. Caliari, P. Kandolf, A. Ostermann, S. Rainer. The Leja method revisited: backward error analysis for the matrix exponential, SIAM Journal on Scientific Computation, Accepted for publication (2016). arXiv:1506.08665.

[6] M. Hochbruck, A. Ostermann. Exponential integrators, Acta Numerica 19 (2010), 209-286

[7] P. Novati, Polynomial methods for the computation of functions of large unsymmetric matrices, Ph.D. Thesis in Computational Mathematics, University of Trieste, advisor I. Moret (2000).

[8] L. Reichel, Newton interpolation at Leja points, BIT 30 (2) (1990), 332–346.

[9] R. Horn, C. Johnson, Matrix Analysis, Cambridge University Press (2012).

[10] L. N. Trefethen, J.A.C. Weideman, The exponentially convergent trapezoidal rule, SIAM Review 56-3 (2014), 385-458.

[11] Python Software Foundation. Python Language Reference, version 3.7. Available at https://www.python.org. Manual at https://docs.python.org/3/. [Online; accessed 2020-02-19]

[12] S. v. d. Walt, C. Colbert, G Varoquaux. The NumPy Array: A Structure for Efficient Numerical Computation, Computing in Science & Engineering, 13 (2011), 22-30.

[13] T. Oliphant. A guide to NumPy, USA: Trelgol Publishing, (2006).

[14] P. Virtanen, R. Gommers, T. Oliphant, M. Haberland, T. Reddy, D. Cournapeau, E. Burovski, P. Peterson, W. Weckesser, J. Bright, S. v. d. Walt, M. Brett, J. Wilson, J. Millman, N. Mayorov, A. Nelson, E. Jones, R. Kern, E. Larson, C. Carey, İ. Polat, Y. Feng, E. Moore, J. VanderPlas, D. Laxalde, J. Perktold, R. Cimrman, I. Henriksen, E. Quintero, C. Harris, A. Archibald, A. Ribeiro, F. Pedregosa, P. v. Mulbregt, and SciPy 1.0 Contributors. SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python. Nature Methods, in press.

[15] J. Hunter. Matplotlib: A 2D Graphics Environment, Computing in Science & Engineering, 9, 90-95 (2007).

[16] W. McKinney. Data Structures for Statistical Computing in Python, Proceedings of the 9th Python in Science Conference, 51-56 (2010).