Leopold-Franzens-Universität

Master Thesis

Matrix-free Leja based exponential integrators in Python

Maximilian Samsinger

December 6, 2019

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 Bachelorarbeit einverstanden.

Datum	Unterschrift

Matrix-free Leja based exponential integrators in Python

Abstract

1 Introduction

Consider the action of the matrix exponential function

$$e^A v, \quad A \in \mathbb{C}^{N\times N}, v \in \mathbb{C}^N.$$

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 seperate step. This is especially true in applications where $N > 10000$ is not uncommon. Furthermore the matrix exponential of a sparse matrix is in general no longer sparse. Therefore it is more feasable 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 advantages. The cost of the computation of $p_m(A)v$ mainly depends on the calculation of n matrix-vector multiplications with A. Furthermore the explicit knowledge of A itself is no longer required. A can be replaced by a linear operator, which can be more convenient and saves memory.

2 The Leja method

This section serves as an introduction to the Leja method for the exponential function. All proofs can be found in a more general form in [4].

2.1 Leja interpolation

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

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

is called a Leja sequence. Due to the maximum principles 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 many beneficial properties.

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

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

Furthermore if f is an entire function, then $(p_m)_{m=0}^{\infty}$ converges superlinearly to f. 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 $A \in \mathbb{C}^{n\times n}, v \in \mathbb{C}^n$.

Recursive constructability: 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.

3 Matrix-free Leja based exponential integratos

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

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

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

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

Similar to Runge-Kutta methods, we replace the integrant with a polynomial approximation. Unlike Runge-Kutta methods, we leave the matrix-exponential untouched and

only replace N. The most well-known Rosenbrock-type exponential integrator, the exponential Rosenbrock-Euler method, can be obtained by using the left hand rule. By replacing N with $N(u_0)$ we get

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

where $\varphi_1(z) = \frac{e^z-1}{z}$. Higher order exponential integrators rely on the efficient computation of the entire functions

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

This can be done by slightly modifying A. For matrices A we have

$$\varphi_1(tA)v = \begin{bmatrix} e^{tA} & v \\ 0 & 0 \end{bmatrix}$$

In the matrix-free case

3.1 Approximating the matrix exponential function:

Inspired by the previous section 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

$$K = [-c, c], \quad f = e^{\cdot}, \quad \xi_0 = 2$$

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

$$e^A v = (e^{s^{-1}A})^s v.$$

for $s \in \mathbb{N}$. By choosing s large enough, we can bound the backward error of This allows us to savely interpolate $e^{s^{-1}A} \approx L_{m,c}(s^{-1}A)$. Now we can approximate the action of the matrix exponential in s substeps

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

This setup was used in [5].

After choosing a fixed tolerance tol they perform a backward error analysis to determine when $\|L_{m,c}(s^{-1}A) - e^{s^{-1}A}\| < \text{tol}/s$. This leads to an optimal choice for s, m and c depending on A, which minimizes the costs of the interpolation.

INSERT TEXT INSERT TEXT INSERT TEXT

For matrix-free linear operators we cannot directly compute the matrix norm $\|A\|$. In general we have

$$\rho(A) \leq \|A\|$$

Shifting the matrix In order to speed up the calculation, we use the relation

$$e^A = e^{\mu} e^{A-\mu I}, \quad \mu \in \mathbb{C}$$

where I is the identity matrix. to center the spectrum of A around 0. For

4 Linear advection diffusion equation

Consider the one-dimensional advection-diffusion equation

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

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

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

The resulting system of ordinary differential equation is given by

$$\partial_t u = Au.$$

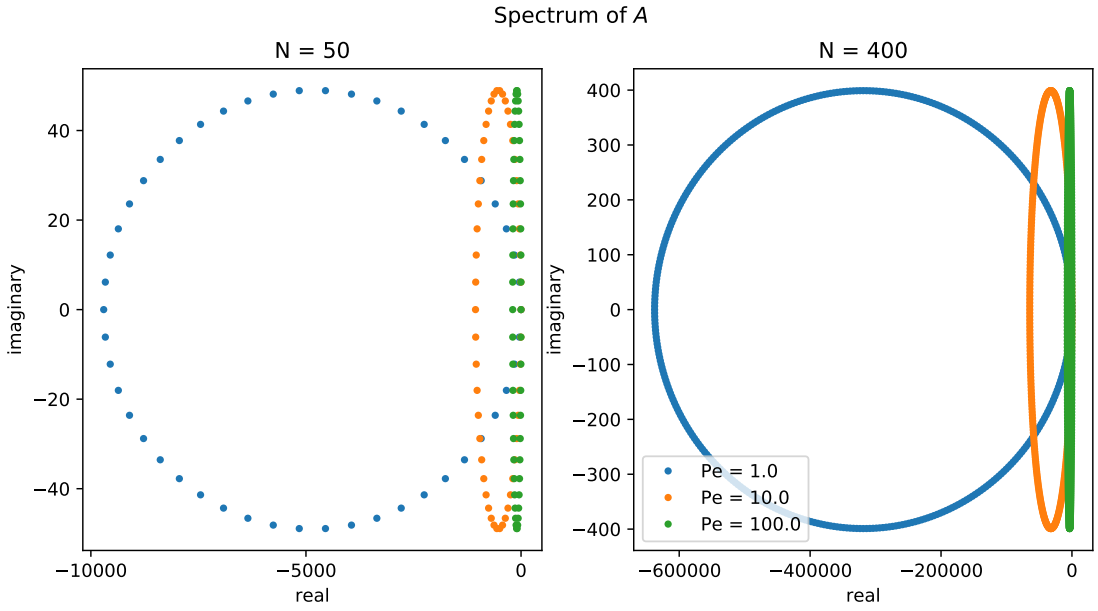


Figure 1: Visualization of the spectrum of A. We assume periodic boundary conditions.

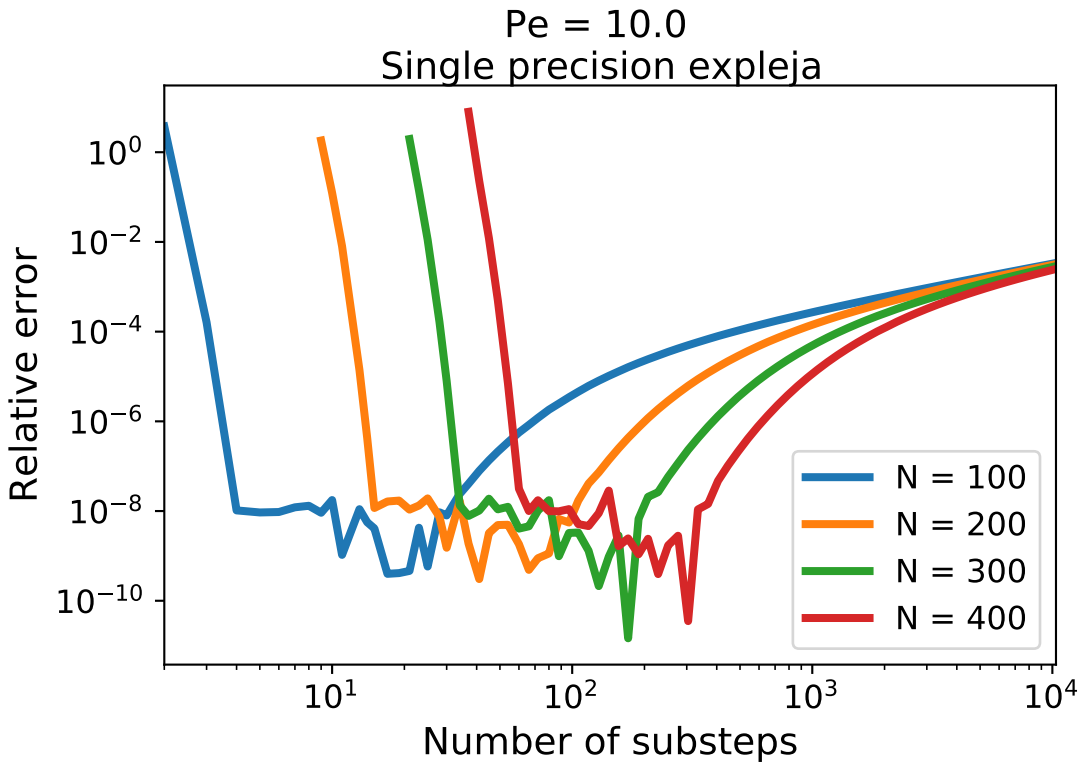


Figure 2: Approximation of $e^A v$

5 Nonlinear Advection-Diffusion-Reaction Equation

Consider the one-dimensional advection-diffusion-reaction equation

$$\partial_t u = \alpha \partial_x((u + \partial_x u)) + \beta \partial_x(u^2) + \gamma u(u - 0.5) \quad \alpha, \beta, \gamma \geq 0$$
$$u_0(t) = e^{-80 \cdot (t-0{,}45)^2} \quad t \in [0, 0.1]$$

on the domain $\Omega = [0, 1]$.

6 Numerical experiments

For the first experiments we will discretize multiple one-dimensional advection-diffusion-reaction equations with hybrid difference schemes.[1] We will always choose an equidistant grid with grid size $h = \frac{1}{N}$, $N \in \mathbb{N}$ and grid points $x_i = ih$ for $i = 0 \ldots, N$ on the domain $\Omega = [0, 1]$. The resulting ordinary differential equations (ODEs) will be solved with four different integrators. Our goal is to investigate the respective computational costs of these methods while achieving a prescribed relative tolerance `tol`.

[1] Need a source, https://en.wikipedia.org/wiki/Hybrid_difference_scheme

Crank-Nicolson method: We refer to the Crank-Nicolson method of order 2 as `cn2`. In our implementation of `cn2`, we used the SciPy[7] package `scipy.sparse.linalg.gmres` to solve linear equations. We set the relative tolerance to $\texttt{tol}/s$, where s is the total number of substeps taken for solving the ODE. This choice guarantees that the sum of errors made by `gmres` is always lower than our specified tolerance `tol`, since we have to solve exactly one linear equation per substep. 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 imposed by the advective and diffusive part of the differential equations.

Exponential Rosenbrock-Euler method: We refer to the Exponential Rosenbrock-Euler method of order 2 as `exprb2`. The approximate the action of the matrix exponential with the Leja method. No hump reduction is used. The maximal interpolation degree is set to 100. Note that the total number of matrix-vector multiplication per time step can still exeed 100 since we have to compute a single matrix norm. This typically happens for $s = 1$.

Explicit midpoint method: We refer to the explicit midpoint method of order 2 as `rk2`.

Classical Runge kutta: We refer to the classical Runge-Kutta method of order 4 as `rk4`.

For our experiments we will often fix one of two different Péclet numbers

$$\texttt{Pe} = \frac{b}{a}, \quad \texttt{pe} = \frac{hb}{2a},$$

The Péclet numbers are dimensionless quantities representing the ratio of the advective velocity b to the diffusive velocity a. While `Pe` characterizes the original partial differential equation, the grid Péclet number `pe` is the dimensionless quantity for the resulting ODE after discretization. Note that by fixing `pe` for varying grid sizes, we have to change the original partial differential equantion. Unless otherwise noted we accomplish that by replacing b with $2b$ and a with ah.

6.1 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.[2]

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

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.[3] 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 funcion with the Leja method. We write `expleja` for the single precision Leja method approximation. Note that 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

6.2 Experiment 2: 1D Nonlinear advection diffusion equation

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

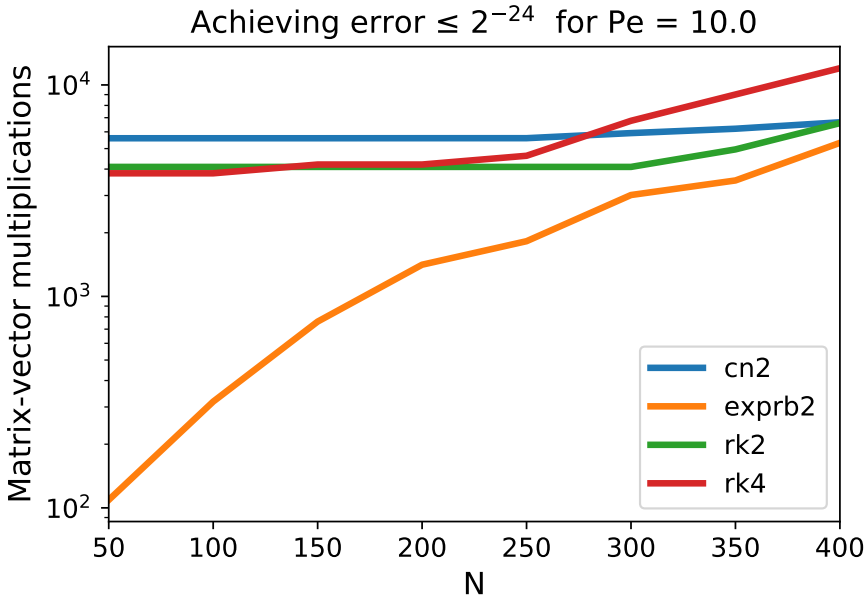
We discretize, solve again with rk2, rk4, cn2 and exprb2.

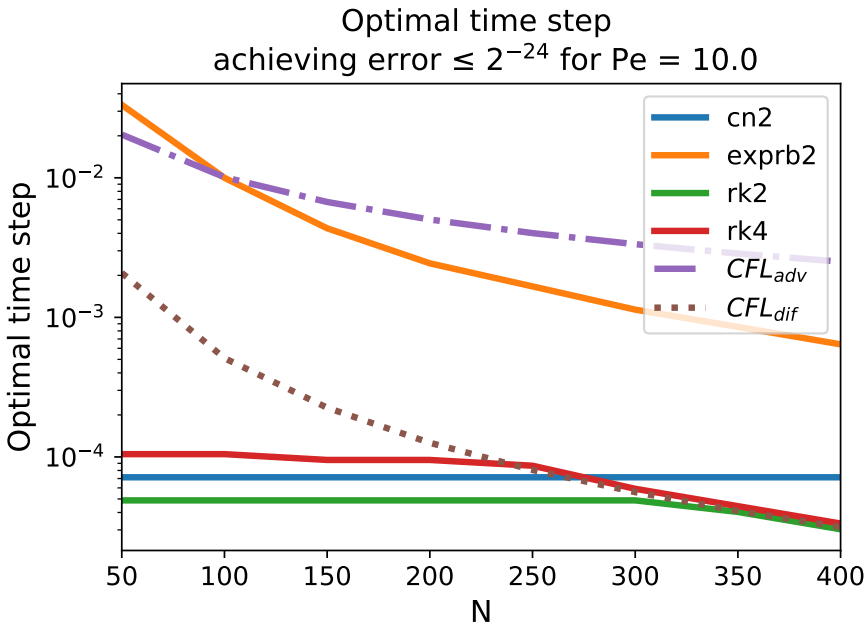
[2] Maybe create a seperate section on hybrid difference schemes? There we can also analyze the resulting matrix A itself and plot the eigenvalues. I need sources for that though.

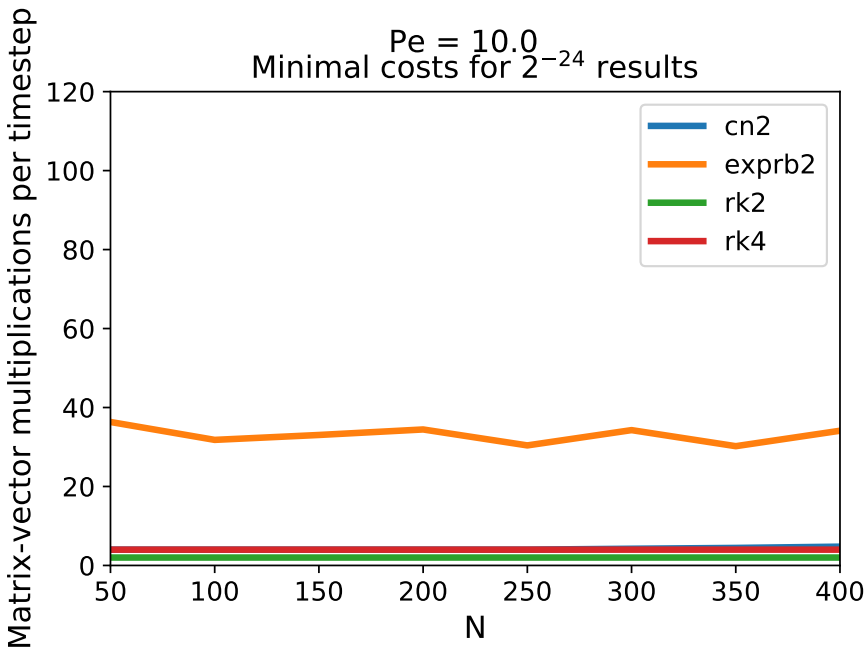
[3] See section **??**

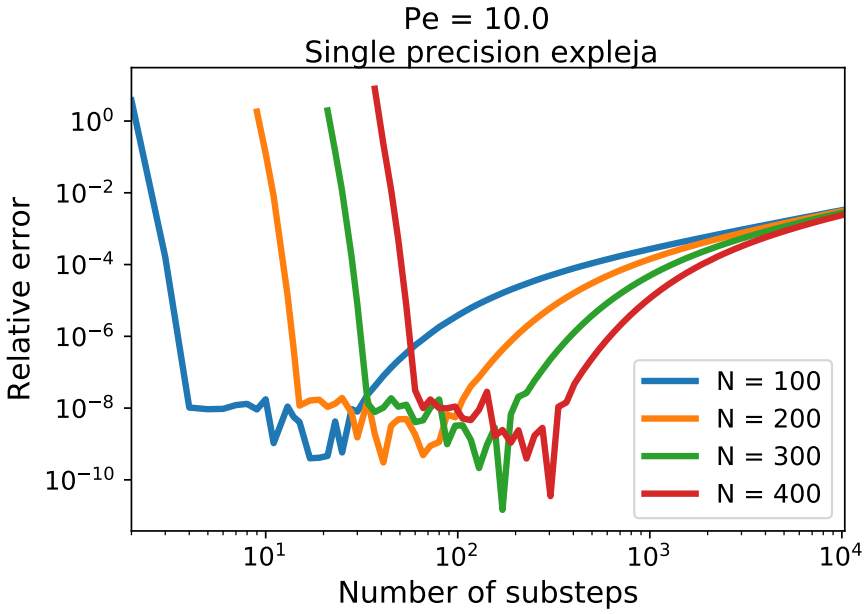
7 Appendix

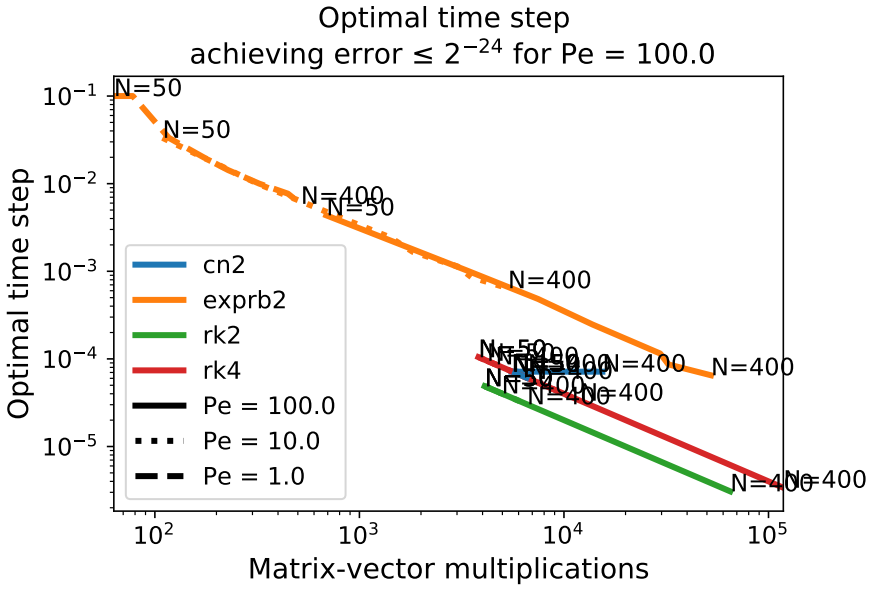
7.1 Experiment 1






Pe = 10.0
Minimal costs for 2^{-24} results
Matrix-vector multiplications per timestep
N
cn2
exprb2
rk2
rk4
0
20
40
60
80
100
120
50
100
150
200
250
300
350
400


Pe = 10.0
Single precision expleja
Relative error
Number of substeps
N = 100
N = 200
N = 300
N = 400
10^{0}
10^{-2}
10^{-4}
10^{-6}
10^{-8}
10^{-10}
10^{1}
10^{2}
10^{3}
10^{4}



7.2 Experiment 1.5

In the matrix-free case the linear operator A is not explicitly given. In order to compute the matrix norm $\|A\|_2$ we use power iterations to estimate the absolutely largest eigenvalue of A. A priory it is not clear how many power iterations *it* are neccessary for a good approximation.

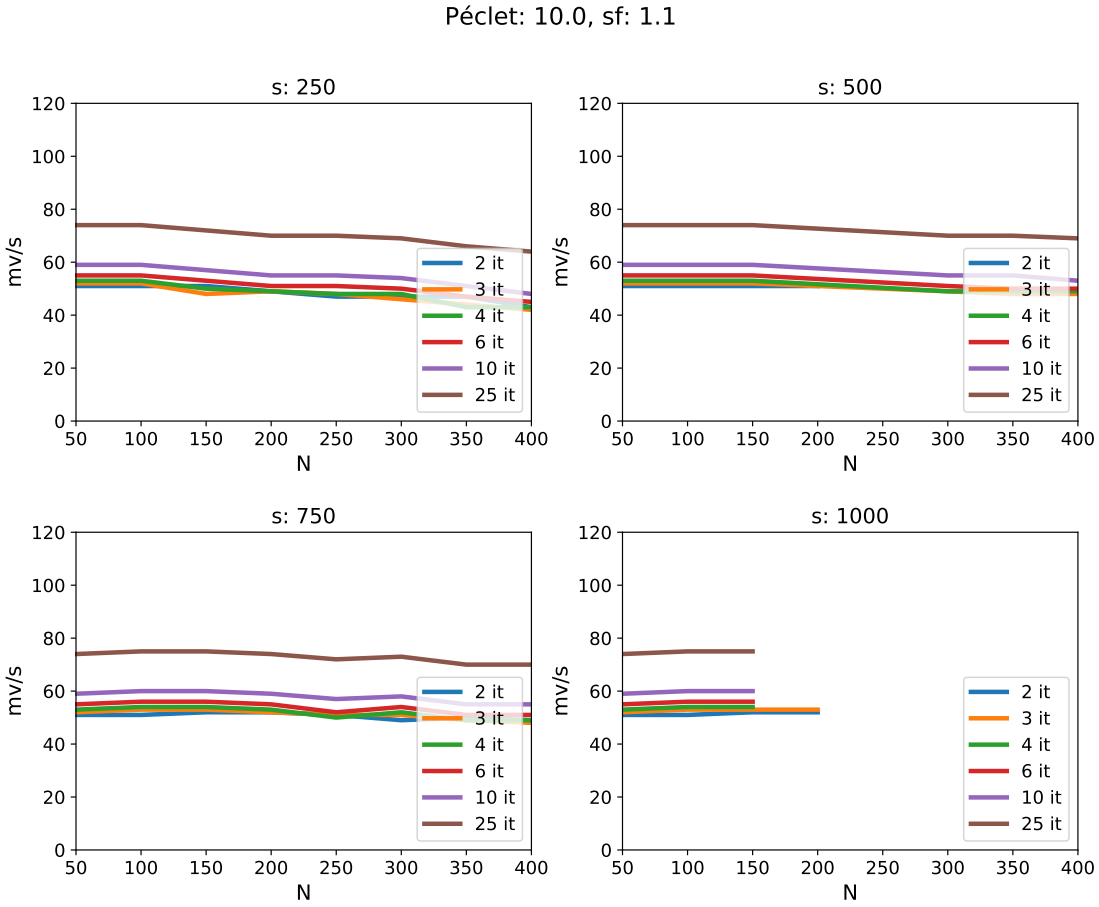
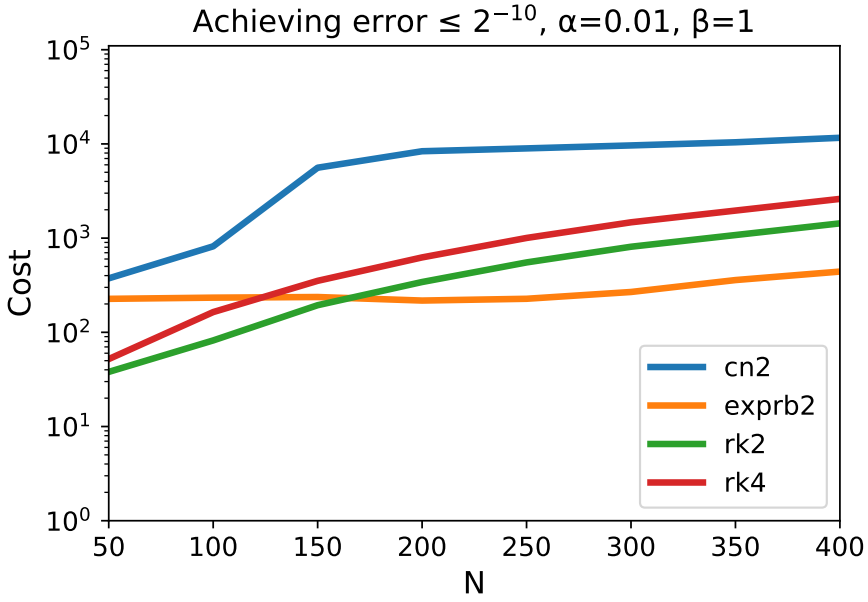
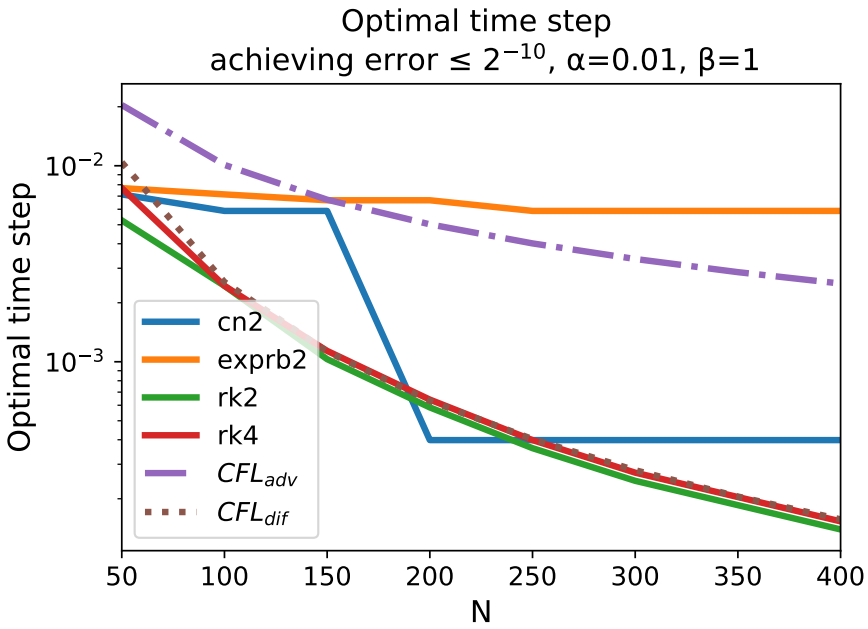


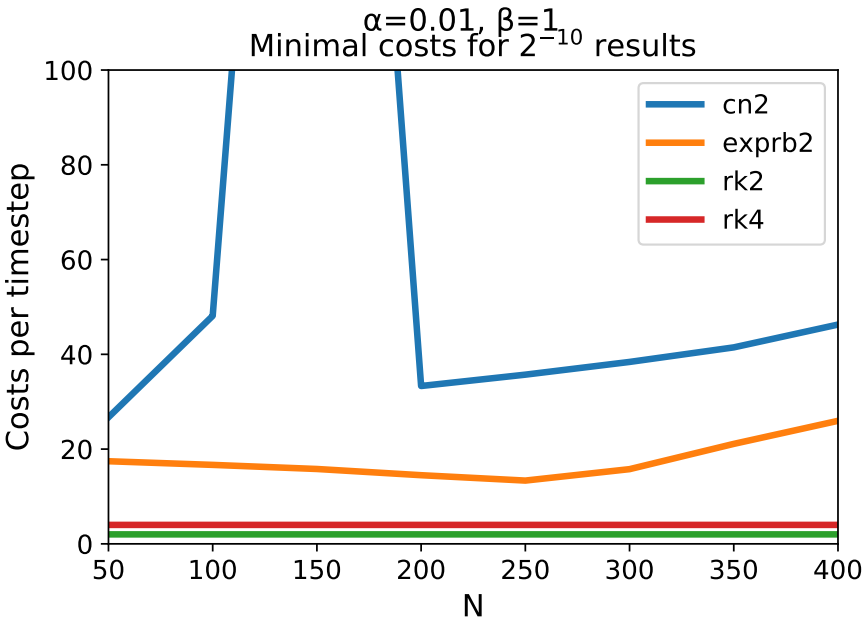
Figure 3: Space dimension N vs costs `mv` per timestep `s` for the exponential Rosenbrock method `exprb2`. Results are only shown if they achieve single precision.

7.3 Experiment 2

1, half, =0.01, =1


Achieving error ≤ 2^{-10}, α=0.01, β=1
Cost
N
cn2
exprb2
rk2
rk4


Optimal time step
achieving error ≤ 2^{-10}, α=0.01, β=1
Optimal time step
N
cn2
exprb2
rk2
rk4
CFL_{adv}
CFL_{dif}


α=0.01, β=1
Minimal costs for 2^{-10} results
Costs per timestep
N
cn2
exprb2
rk2
rk4
0
20
40
60
80
100
50
100
150
200
250
300
350
400

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] Python Software Foundation. Python Language Reference, version 2.7. Available at http://www.python.org. Manual at https://docs.python.org/2/. [Online; accessed 2015-12-14]

[7] E. Jones, E. Oliphant, P. Peterson, SciPy: Open Source Scientific Tools for Python, Available at http://www.scipy.org/. [Online; accessed 2015-12-14]