Leopold-Franzens-Universität

Master Thesis

Matrix-free Leja based exponential integrators in Python

Maximilian Samsinger

May 11, 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

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

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_n of degree n in A

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

This approach has many advantages. The cost of the computation of $p_n(A)v$ mainly depends on the calculation of $n \in \mathbb{N}$ matrix-vector multiplications with A. Not only can A be sparse, which significantly decreases the costs of the computation, the explicit knowledge of A itself is no longer required. A can be replaced by a linear function, which can be more convenient and saves memory.

2 Numerical experiments for the advection-diffusion equation

We solve the advection-diffusion equation for four different integrators and investigate the respective computational costs necessary to achieve a prescribed tolerance.

- cn2: The Crank-Nicolson method of order 2.
- exprb2: The exponential Rosenbrock-Euler method of order 2.
- rk2: The explicit midpoint method of order 2.
- rk4: The classical Runge-Kutta method of order 4.

The stiffness of this differential equation mainly depends on the advection-diffusion ratio.

Note that this comparison is a bit unfair, since the expleja approximates the matrix exponential function, which returns the exact solution in the linear case. In the nonlinear case, which we consider in Experiment 2, we replace expleja with the exponential Euler method. We expect that the behaviour in the linear case will be similar to the nonlinear case

2.1 Experiment 1

Consider the one-dimensional advection-diffusion equation

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

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

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

where $h = \frac{1}{N}$ is the lenght between grid points. In order to limit numerical instabilities we discretize the advective part with the upwind scheme. Still the resulting system of ordinary differential equation

$$\partial_t u = Au$$

imposes stringent conditions on h and

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 only consider coefficients a, $b \in [0, 1]$ such that $a + b = 1$. We will denote the advection-diffusion ratio scaled by grid size $\Delta x = (N - 1)^{-1}$ by

$$\mathrm{Pe} = \frac{b\Delta x}{2a}$$

.

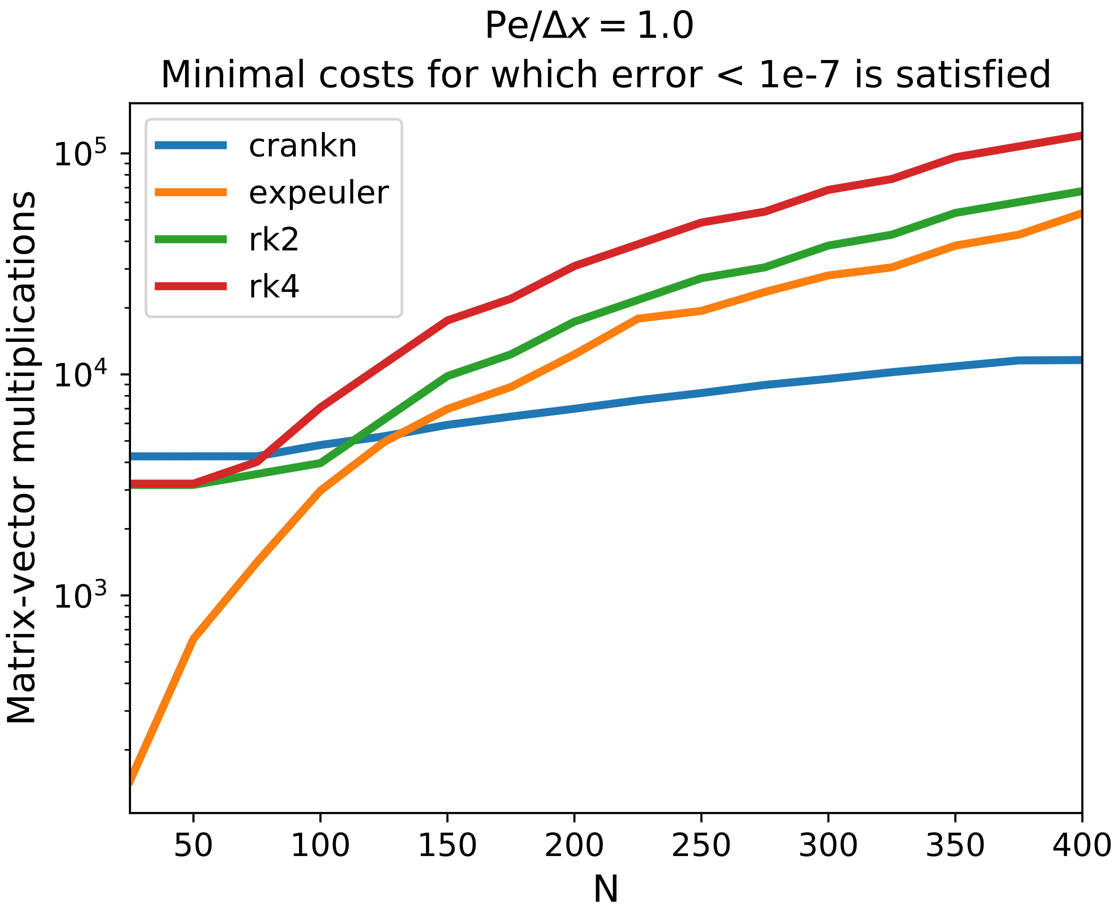

Figure 1: A picture of a gull.

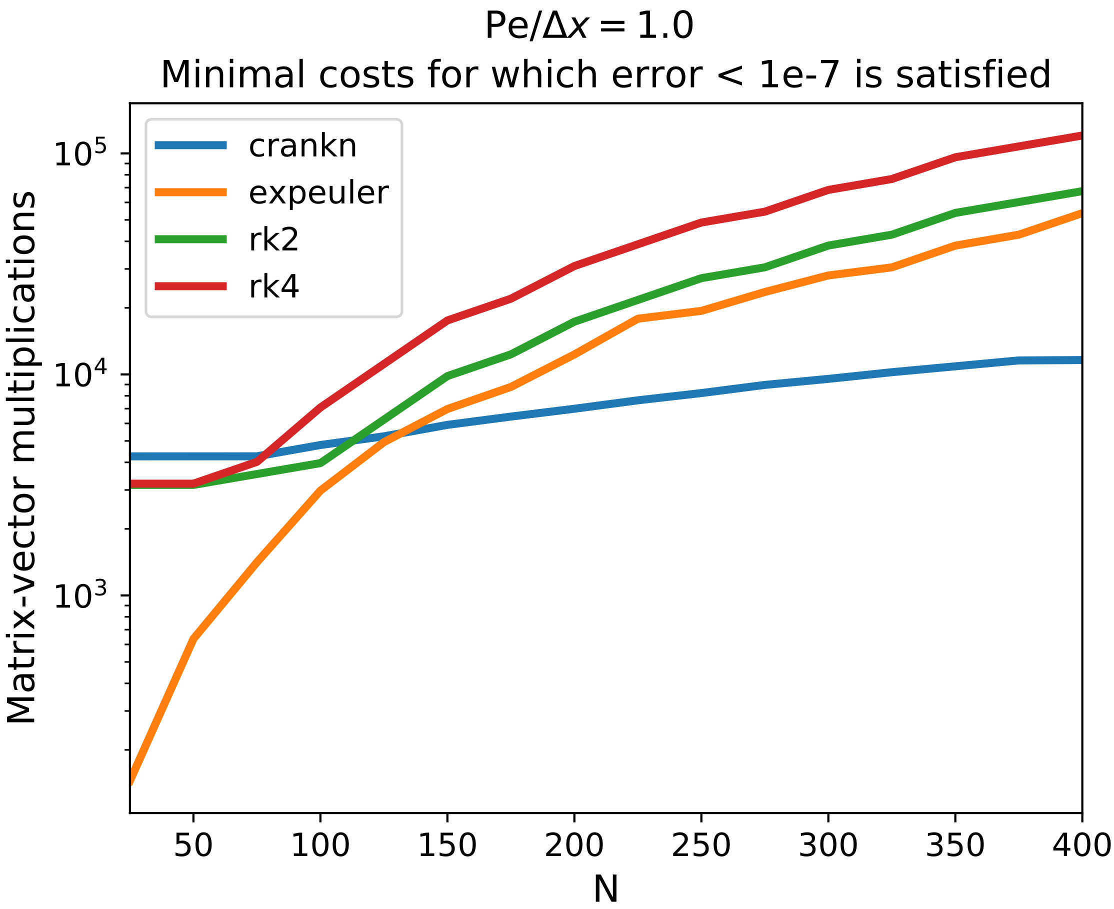

Figure 2: A picture of a gull.

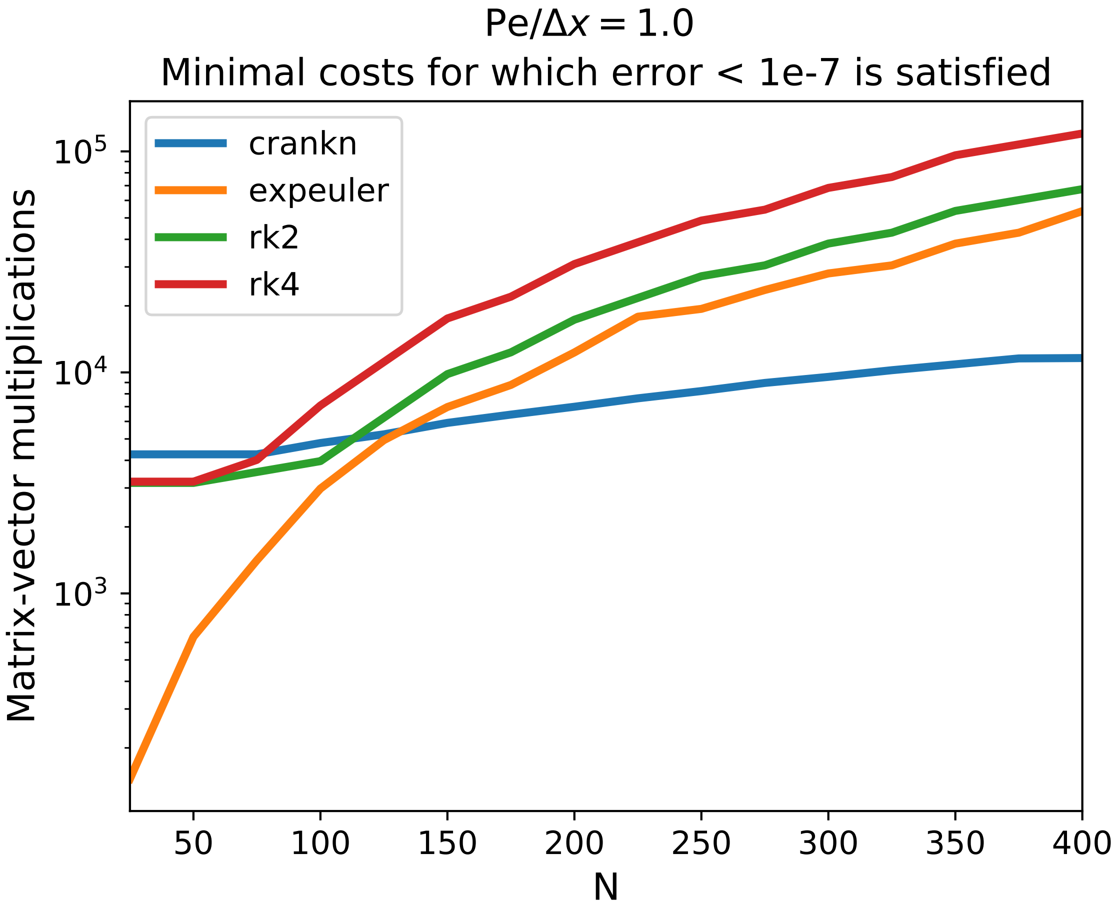

Figure 3: A picture of a gull.

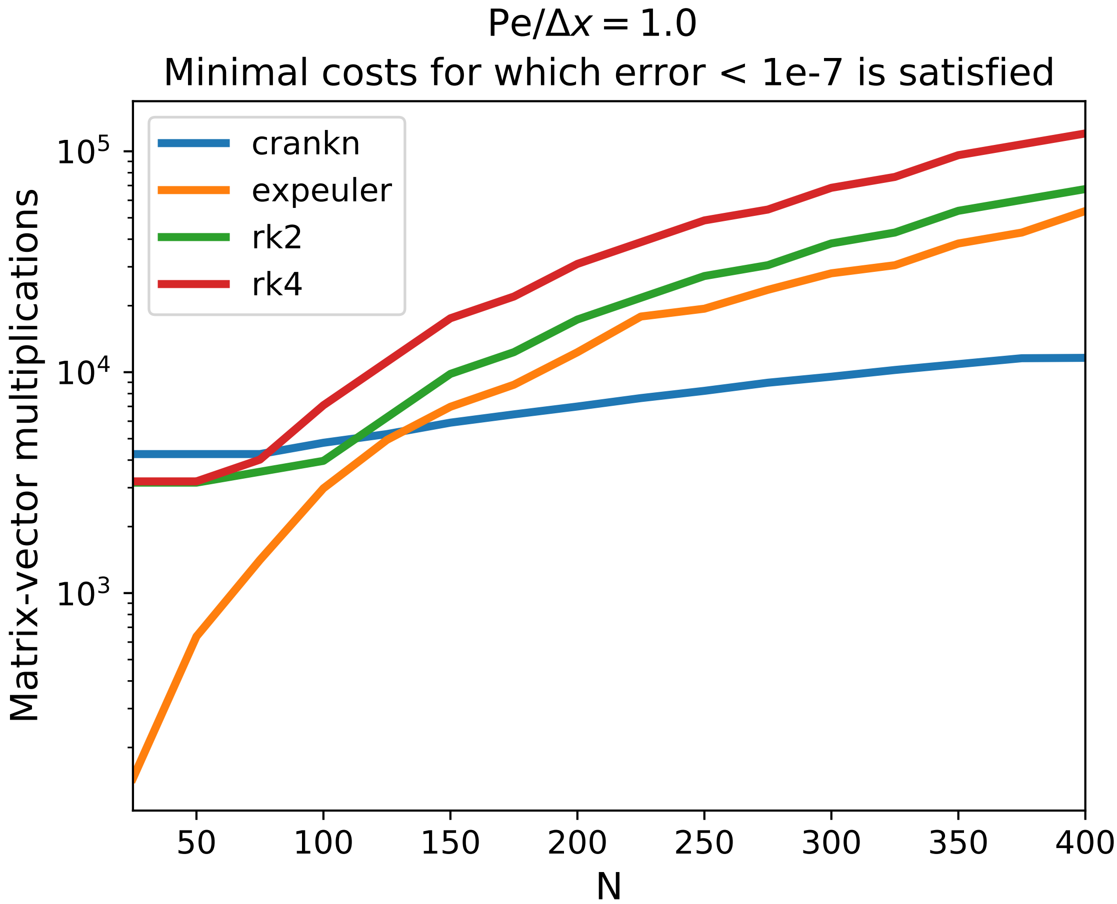

Figure 4: A picture of a gull.

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]