Tensor-Product Preconditioner For Very High Order Discontinuous Galerkin Discretizations

Tom Feldhausen

December 11, 2025

Abstract

Solving a multi-dimensional PDE by means of a Discontinuous Galerkin method in a higher order solution space requires the solution of large linear systems during time integration. When storing the semi-discrete system, communication dominates the computational costs such that matrix-free implementations [KK19] often turn out to be more efficient on modern architectures. In order to efficiently integrate implicit time integration into this framework, we need matrix-free preconditioners. This project aims to compare the tensor-product preconditioners of Diosady [DM17] and Pazner [PP18] for the example of linear advection and provide a parallelized matrix-free C++ deal.II [ABB$^+$24] implemenation. As a first step, we provide a matrix-based DG implementation of two-dimensional linear advection with upwind flux and verify the efficiency of Pazner's preconditioner for implicit time integration.

A Matrix-Based Discontinuous Galerkin Implementation For Two-Dimensional Linear Advection In Python

The code for all experiments is provided in [Fel25].

As a first step, we provide a matrix-based DG discretization for the two-dimensional linear advection system

$$\begin{aligned}&\partial_t u(x,t) + a(x,t)\cdot\nabla u(x,t) = 0 \quad \forall x\in[0,1]^2, \quad 0\le t\le T\\ &u(x,0) = u_0(x) \quad \forall x\in[0,1]^2\\ &u(x,t) = u_{\text{in}}(x) \quad \forall x\in\partial[0,1]^2\end{aligned}$$

for a solution function u and a domain $\Omega = [0,1]^2$. Following Kronbichler and Persson [KP21], we arrive at the weak formulation that for all $v\in H^1(\Omega)$ it needs to hold that

$$\int_\Omega \partial_t uv - \int_\Omega ua\cdot\nabla v + \int_{\partial\Omega}(a\cdot n)uv = 0, \tag{1}$$

after integrating in space, multiplying by a test function v and integrating by parts.

Now insert basis functions of order p, i.e.

$$u = \sum_{i=0}^{(p+1)^2-1} u_i\Phi_i, v = \sum_{j=0}^{(p+1)^2-1} v_j\Phi_j.$$

Then condition (1) is equivalent to the element-wise formulation for each basis function, i.e.

$$\sum_i \int_{\Omega_e}\partial_t u_i\Phi_i\Phi_j - \int_{\Omega_e} u_i\Phi_i a\cdot\nabla\Phi_j + \int_{\partial\Omega_e}(a\cdot n)u_i\Phi_i\Phi_j = 0 \tag{2}$$

being true for all $0\le j\le (p+1)^2$ and $\Omega_e\subset\Omega$.

Analogously we can write this in matrix form, which yields the system

$$\frac{dU(t)}{dt} = M^{-1}[(B-G)U - G_{\text{bound}}]. \tag{3}$$

We consider a tensor-product mesh with $N = N_x\times N_y$ rectangular elements. Then $x:\Omega_{\text{ref}}\to\Omega_e$ maps from the reference element $\Omega_{\text{ref}} = [-1,1]^2$ to the actual element $\Omega_e = [x_0,x_1]\times[y_0,y_1]$ and can be explicitly written as the affine transformation

$$\mathbf{x}(\boldsymbol{\zeta}) = A\,\boldsymbol{\zeta} + \mathbf{b}, \qquad A = \frac{1}{2}\begin{pmatrix} x_1-x_0 & 0\\ 0 & y_1-y_0\end{pmatrix}, \qquad \mathbf{b} = \begin{pmatrix}\frac{x_1+x_0}{2}\\ \frac{y_1+y_0}{2}\end{pmatrix}.$$

With this at hand, we can explicitly write out the mass matrix as

$$\begin{aligned} M_{ji} &= \int_{\Omega_e}\Phi_j(z)\Phi_i(z)dz\\ &= \int_{x_0}^{x_1}\int_{y_0}^{y_1}\Phi_j(z_0,z_1)\Phi_i(z_0,z_1)dz_0dz_1\\ &= \int_{-1}^{1}\int_{-1}^{1}\Phi_j(\zeta_1,\zeta_2)\Phi_i(\zeta_1,\zeta_2)\|\det(J_e)\|d\zeta_1 d\zeta_2,\end{aligned}$$

where J_e denotes the Jacobian of the reference to element transformation $x(\zeta)$. Using Gauss-Legendre quadrature with one-dimensional weights $w_q = w_{q_0} w_{q_1}$ and points $x_q = (x_{q_0}, x_{q_1})$, we get the approximation

$$\begin{aligned} M_{ji} = {} & \int_{\Omega_e} \Phi_j(z)\Phi_i(z)dz \\ \approx {} & \sum_{q_0}\sum_{q_1} w_{q_0}w_{q_1}\phi_{i_0}(x_{q_0})\phi_{i_1}(x_{q_1})\phi_{j_0}(x_{q_0})\phi_{j_1}(x_{q_1})\frac{1}{4}(x_1-x_0)(y_1-y_0) \end{aligned}$$

when inserting the tensor-product form $\Phi_i = \phi_{i_0}\phi_{i_1}$ and exploiting $\| \det(J_e)\| = \frac{1}{4}(x_1 - x_0)(y_1 - y_0)$.

Similarly, we can derive an expression for the volume matrix and calculate

$$\begin{aligned} B_{ji} = {} & \int_{\Omega_e} \Phi_i(z)(a(z,t)\cdot\nabla\Phi_j(z))dz \\ = {} & \int_{x_0}^{x_1}\int_{y_0}^{y_1} \Phi_i(z_0,z_1)(a(z_0,z_1,t)\cdot\nabla\Phi_j(z_0,z_1))dz_0dz_1 \\ = {} & \int_{-1}^{1}\int_{-1}^{1} \Phi_i(\zeta_0,\zeta_1)(a(z_0,z_1,t)\cdot J_e^{-1}\nabla\Phi_j(\zeta_0,\zeta_1))\|\det(J_e)\|d\zeta_0d\zeta_1 \\ = {} & \int_{-1}^{1}\int_{-1}^{1} \Phi_i(\zeta_0,\zeta_1)[2(x_1-x_0)^{-1}a_0(x(\zeta),t)\partial_{\zeta_0}\Phi_j(\zeta_0,\zeta_1) + 2(y_1-y_0)^{-1} \\ & a_1(x(\zeta),t)\partial_{\zeta_1}\Phi_j(\zeta_0,\zeta_1)]\frac{1}{4}(x_1-x_0)(y_1-y_0)d\zeta_0d\zeta_1 \\ = {} & \sum_{q_0}\sum_{q_1} w_{q_0}w_{q_1}\phi_{i_0}(x_{q_0})\phi_{i_1}(x_{q_1})[\frac{1}{2}(y_1-y_0)a_0(x(x_q),t)\phi'_{j_0}(x_{q_0})\phi_{j_1}(x_{q_1}) \\ & +\frac{1}{2}(x_1-x_0)a_1(x(x_q),t)\phi_{j_0}(x_{q_0})\phi'_{j_1}(x_{q_1})]. \end{aligned}$$

The face terms are slightly more difficult to derive as in two dimensions we have four faces that are either interior and face other elements or that are part of the domain boundary. This is why for a specific derivation, we need to choose an explicit flux and face. In this example, we choose an upwind flux for information flow between elements, i.e.

$$\widehat{au} = \begin{cases} (a\cdot n)u^-, & a\cdot n > 0, \\ (a\cdot n)u^+, & a\cdot n \leq 0 \end{cases}$$

and focus on the left face. u^- denotes the exterior value, for the left face also written as u^L, and u^+ denotes the interior value, for the left face also written as u^R. As this is one-dimensional, we will only need the second $x_1(\zeta)$ part of the transformation that concerns the x_1-axis because the x_0-value is fixed. The corresponding determinant of the inverse Jacobian is simply $\frac{1}{2}(y_1 - y_0)$ for our tensor-product grid. Generally it holds that

$$\begin{aligned} & \int_{\partial\Omega_e} \widehat{au}(\Phi_i^R(z),\Phi_i^L(z))\Phi_j(z)dz \\ = {} & \int_{-1}^{1}\frac{1}{2}(y_1-y_0)\phi_{j_0}(-1)\phi_{j_1}(\zeta)\widehat{au}(\Phi_i^R(\zeta),\Phi_i^L(\zeta))d\zeta \\ \approx {} & \sum_q w_q\frac{1}{2}(y_1-y_0)\phi_{j_0}(-1)\phi_{j_1}(x_q)\widehat{au}(u_i^R\phi_{i_0}^R(-1)\phi_{i_1}^R(x_1(x_q)),u_i^L\phi_{i_0}^L(-1)\phi_{i_1}^L(x_1(x_q))), \end{aligned}$$

where w_q, x_q are one-dimensional quadrature weights and points.

The value of $\widehat{au}(u_i^R \phi_{i_0}^R(-1)\phi_{i_1}^R(x_1(x_q)), u_i^L \phi_{i_0}^L(-1)\phi_{i_1}^L(x_1(x_q)))$ depends on the following cases, we write the first and second component of our reference to element mapping as x_0 and x_1.

Case 1A: Inflow Boundary.

$$G_{\text{bound}j} \approx \sum_q w_q \frac{1}{2}(y_1 - y_0)\phi_{j_0}(-1)\phi_{j_1}(x_q)(a(x_0(-1), x_1(x_q), t) \cdot n)u(x_0(-1), x_1(x_q), t)$$

Case 1B: Outflow Boundary.

$$G_{ji^R} \approx \sum_q w_q \frac{1}{2}(y_1 - y_0)\phi_{j_0}(-1)\phi_{j_1}(x_q)(a(x_0(-1), x_1(x_q), t) \cdot n)\phi_{i_0}^R(-1)\phi_{i_1}^R(x_q)u_i^R$$

Case 2A: Inflow Interior Face.

$$G_{ji^L} \approx \sum_q w_q \frac{1}{2}(y_1 - y_0)\phi_{j_0}(-1)\phi_{j_1}(x_q)(a(x_0(-1), x_1(x_q), t) \cdot n)\phi_{i_0}^L(-1)\phi_{i_1}^L(x_q)u_i^L$$

Case 2B: Outflow Interior.

$$G_{ji^R} \approx \sum_q w_q \frac{1}{2}(y_1 - y_0)\phi_{j_0}(-1)\phi_{j_1}(x_q)(a(x_0(-1), x_1(x_q), t) \cdot n)\phi_{i_0}^R(-1)\phi_{i_1}^R(x_q)u_i^R$$

Notice that the outflow cases 1B and 2B yield the same formula. Now that we have set up the semi-discretization in space, we can turn our attention to time integration. We can evolve (3) in time by applying a Runge–Kutta method. In our experiments, we set $N_x = 10, N_y = 10$ and $p = 3$. We choose Gauss-Legendre quadrature with $N_q = p+1$ 1D quadrature nodes for Lagrange elements

$$\phi_i(x) = \prod_{j \neq i} \frac{x - x_j}{x_i - x_j}$$

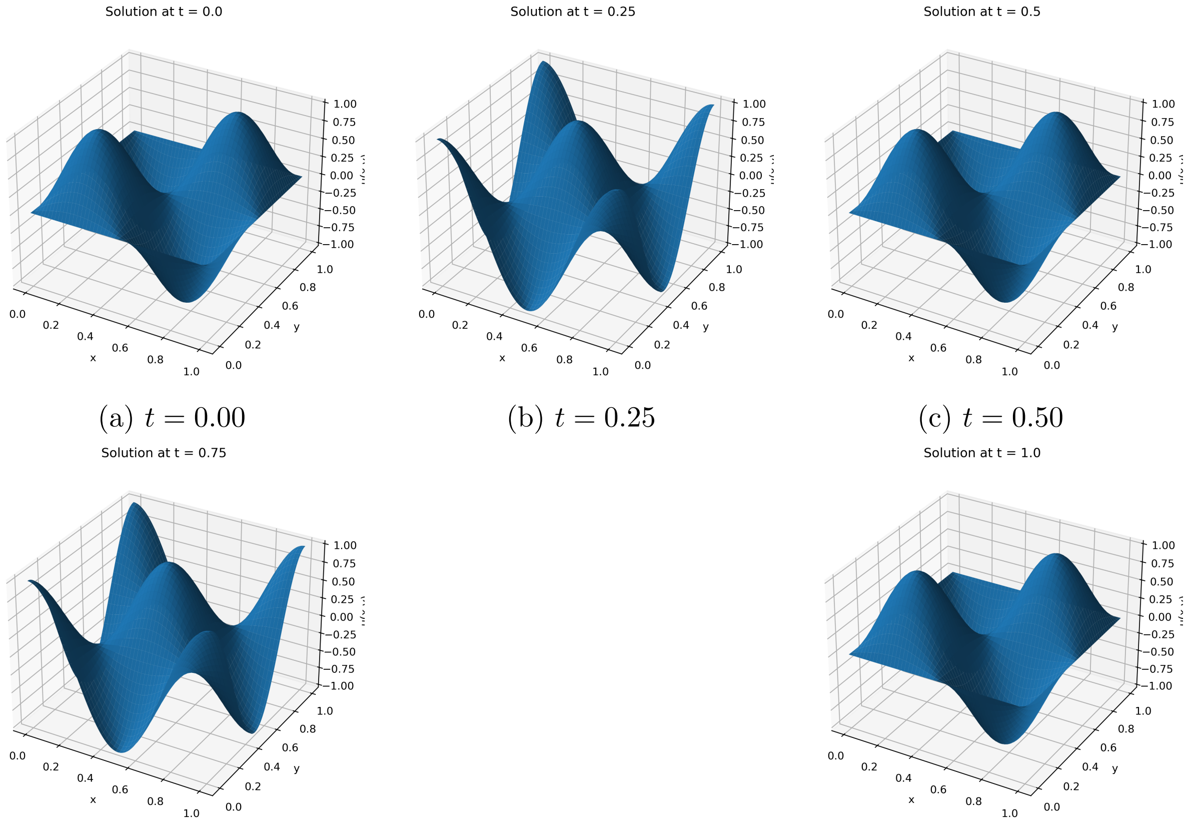
and define the exact solution to be

$$u(x_0, x_1, t) = sin(2\pi(x_0 - t))sin(2\pi(x_1 - t)),$$

with the corresponding initial and boundary Dirichlet conditions. Figure 1 shows different snapshots of the exact solution. It is evident that the solution at final time and initial time is the same for $T = 1$. For our experiments, we chose $T = 0.1$ and measure the error in the Euclidean norm.

Figure 3 shows the convergence behaviour for the different explicit integrators of order 1 to 4 listed in figure 2. As expected, the higher the order of the method, the more accurate is the solution with respect to the step size. It is also no surprise that the runtime of higher order methods is larger for a fixed step size as shown in figure 4.

A DIRK scheme is an implicit Runge–Kutta method with $a_{ij} = 0$ for all $j > i$. SDIRK methods further fulfill $a_{ii} = a_{jj}$ for all $i, j \leq s$, where s denotes the number of stages.



(a) $t = 0.00$ (b) $t = 0.25$ (c) $t = 0.50$

(d) $t = 0.75$ (e) $t = 1.00$

Figure 1: Snapshots of the exact solution at selected times.

$$\begin{array}{c|c} 0 & 0 \\ \hline & 1 \end{array}$$

(a) Explicit Euler (order 1)

$$\begin{array}{c|cc} 0 & 0 & 0 \\ 1 & 1 & 0 \\ \hline & \frac{1}{2} & \frac{1}{2} \end{array}$$

(b) Heun / Explicit RK2 (order 2)

$$\begin{array}{c|ccc} 0 & 0 & 0 & 0 \\ \frac{1}{2} & \frac{1}{2} & 0 & 0 \\ 1 & -1 & 2 & 0 \\ \hline & \frac{1}{6} & \frac{2}{3} & \frac{1}{6} \end{array}$$

(c) Classical explicit RK3 (order 3)

$$\begin{array}{c|cccc} 0 & 0 & 0 & 0 & 0 \\ \frac{1}{2} & \frac{1}{2} & 0 & 0 & 0 \\ \frac{1}{2} & 0 & \frac{1}{2} & 0 & 0 \\ 1 & 0 & 0 & 1 & 0 \\ \hline & \frac{1}{6} & \frac{1}{3} & \frac{1}{3} & \frac{1}{6} \end{array}$$

(d) Classical explicit RK4 (order 4)

Figure 2: Butcher tableaus of explicit Runge–Kutta schemes of order 1–4.

For the implicit SDIRK schemes [KC16] with 1 to 4 stages from figure 5, we get the convergence results illustrated in figure 6. Clearly, the implicit methods deliver more accurate results for larger step sizes. However, figure 7 shows that this comes at the expense of a longer runtime.

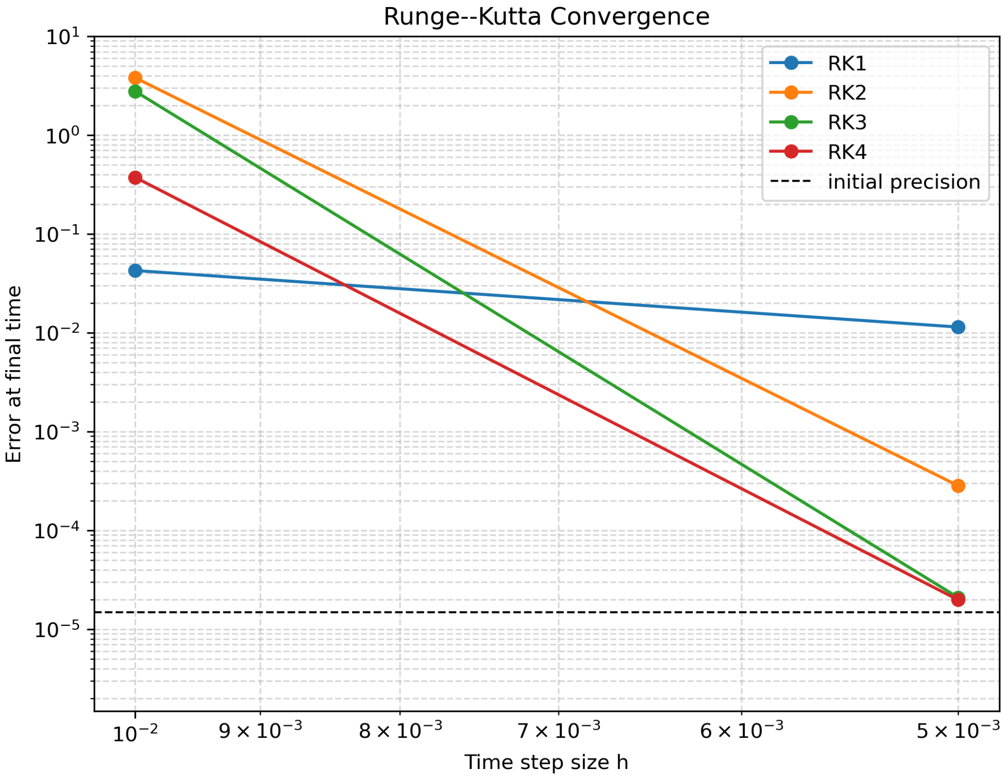


Figure 3: Convergence of explicit Runge–Kutta methods with 1 to 4 stages.

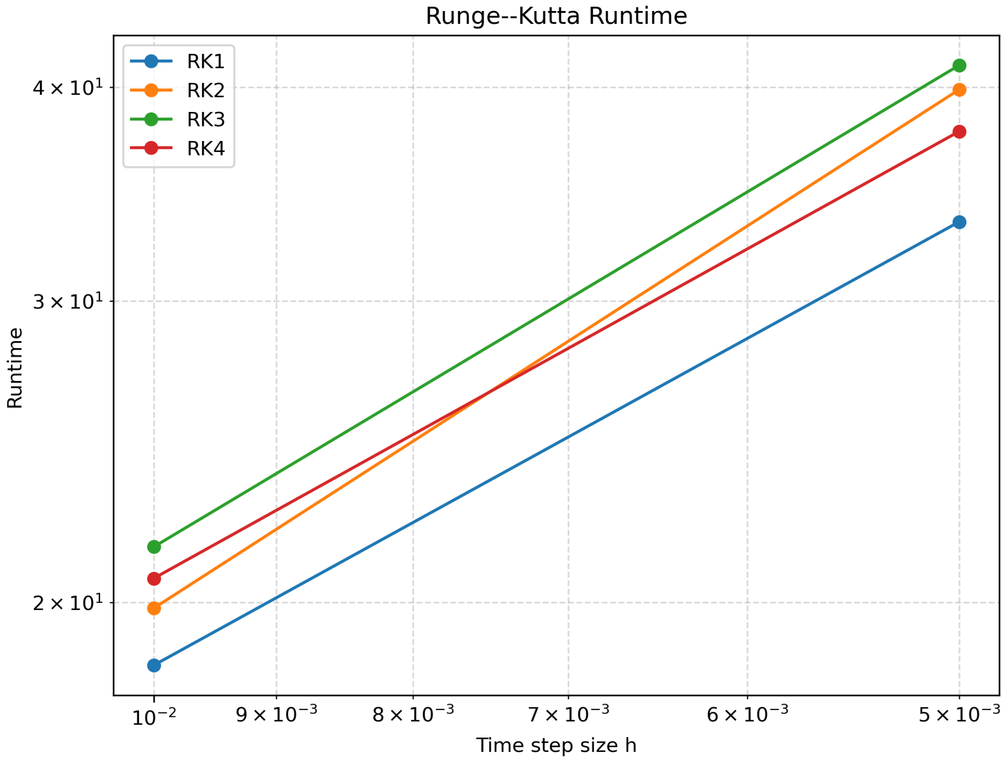


Figure 4: Runtime of explicit Runge–Kutta methods with 1 to 4 stages.

In order to address this issue and speed up the implicit integrators, we would like to use the matrix-free tensor-product preconditioners of Pazner [PP18] and Diosady [DM17] for

$$\begin{array}{c|c} 1 & 1 \\ \hline & 1 \end{array}$$

(a) Implicit Euler (order 1)

$$\begin{array}{c|cc} \gamma & \gamma & 0 \\ 1-2\gamma & 1-2\gamma & \gamma \\ \hline & \frac{1}{2} & \frac{1}{2} \end{array}$$

(b) SDIRK2–NCS23 (order 3), $\gamma = (3+\sqrt{3})/6$ [KC16, p. 27]

$$\begin{array}{c|ccc} \gamma & \gamma & 0 & 0 \\ \frac{1}{2}-\gamma & \frac{1}{2}-\gamma & \gamma & 0 \\ 2\gamma & 2\gamma & 1-4\gamma & \gamma \\ \hline & \frac{1}{6(1-2\gamma)^2} & \frac{2(1-6\gamma+6\gamma^2)}{3(2\gamma-1)^2} & \frac{1}{6(1-2\gamma)^2} \end{array}$$

(c) SDIRK3–NC34 (order 4), $\gamma = \frac{3+2\sqrt{3}cos(\Pi/18)}{6}$ [KC16, p. 27]

$$\begin{array}{c|cccc} 0 & 0 & 0 & 0 & 0 \\ 2\gamma & \gamma & \gamma & 0 & 0 \\ 1 & \frac{-4\gamma^2+6\gamma-1}{4\gamma} & \frac{-2\gamma+1}{4\gamma} & \gamma & 0 \\ 1 & \frac{6\gamma-1}{12\gamma} & -\frac{1}{(24\gamma-12)\gamma} & \frac{-6\gamma^2+6\gamma-1}{6\gamma-3} & \gamma \\ \hline & \frac{6\gamma-1}{12\gamma} & -\frac{1}{(24\gamma-12)\gamma} & \frac{-6\gamma^2+6\gamma-1}{6\gamma-3} & \gamma \end{array}$$

(d) 4-stage SDIRK (order 4), $\gamma = 0.435866521508$

Figure 5: Butcher tableaux for implicit Euler and SDIRK methods of order 2–4.

the linear system arising during implicit time integration. We begin with implementing the former for matrix-based code. Notice that we do not exploit the efficiency advantages of a matrix-free tensor-product implementation yet.

A Kronecker-SVD Based Preconditioner

When solving (3) with a DIRK integrator, we get an implicit system of the form

$$(M - ha_{jj}(B-G))U_j = MU_0 - ha_{jj}G_{\text{bound}}(t_j) + h\sum_{l=1}^{j-1} a_{jl}[(B-G)U - G_{\text{bound}}(t_l)]$$

at each stage j. Notice that the right-hand side can be explicitly computed, such that we arrive at a linear system. Notice that for a SDIRK method, it holds that $a_{jj} = a_{ll}$ for all j, l such that we always have the same operator on the left, thus can always use the same preconditioner and precompute it only once.

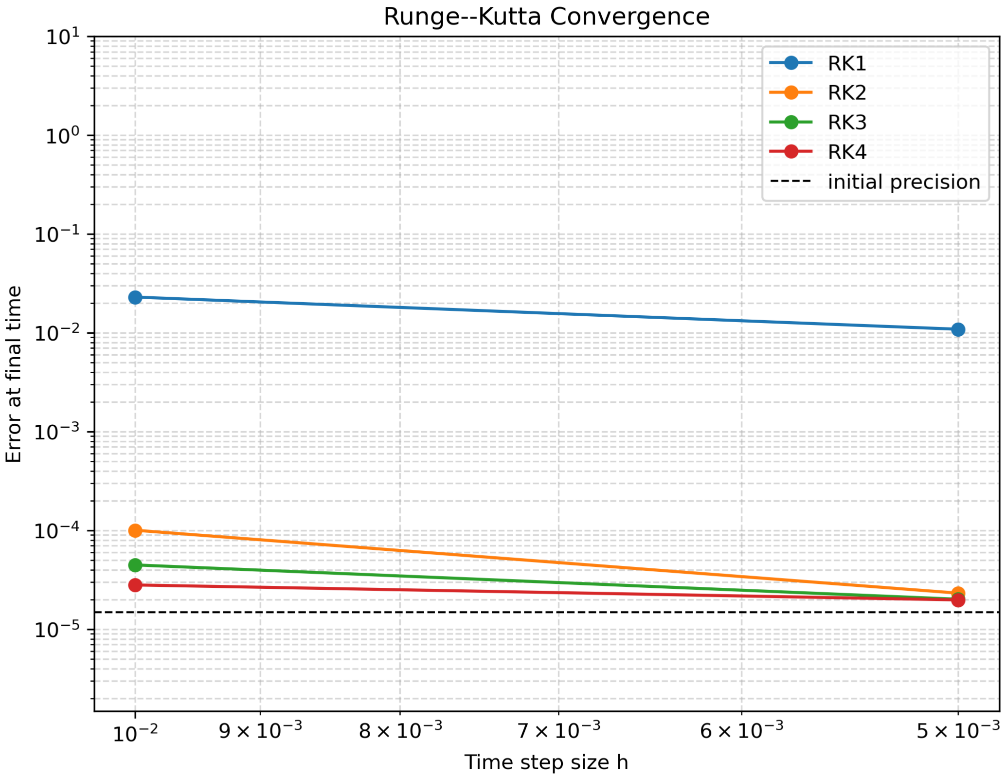


Figure 6: Convergence of implicit Runge–Kutta methods with 1 to 4 stages.

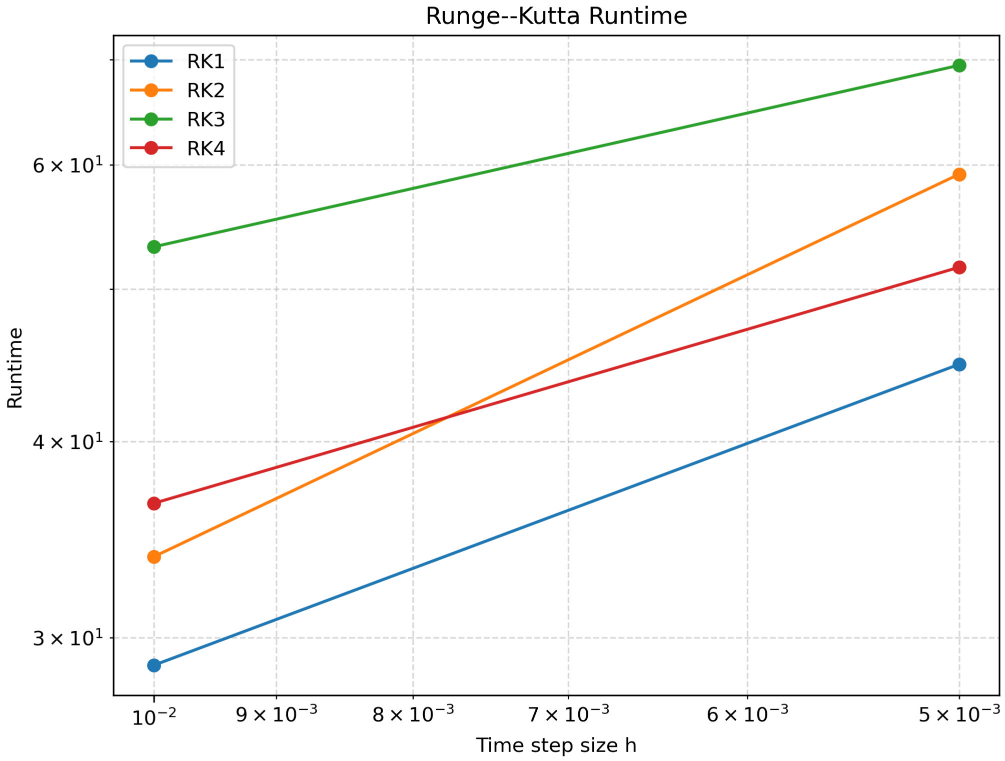


Figure 7: Runtime of implicit Runge–Kutta methods with 1 to 4 stages.

Pazner [PP18] proposes to use an element-wise rank-2 Kronecker SVD to do so, i.e. for

$A_e = M_e - ha_{jj}(G_e - B_e)$ we estimate

$$A_e \approx \sum_{j=1}^{r} A_j \otimes B_j, \tag{4}$$

which can be computed via an SVD of a shuffled version $\tilde{A}_e$ of A_e. The full SVD's cost scales cubicly in the size of the matrix to be estimated, see [GVL13, Ch. 8.6]. Luckily, there are ways to avoid this, namely the Lanczos SVD and the Randomized SVD [TW23]. Both reduce the runtime to near squared complexity and deliver near-optimal accuracy with high probability. In his paper, Pazner decides to use the Lanczos algorithm. For $r = 2$, the approximation (4) can be inverted effiently by analytical means, see [PP18, Ch. 4].
In our experiments in the same setting as earlier, this yields the following convergence behaviour. Accuracy-wise, we expectedly get the same results as we only changed the preconditioner, see figure 8.

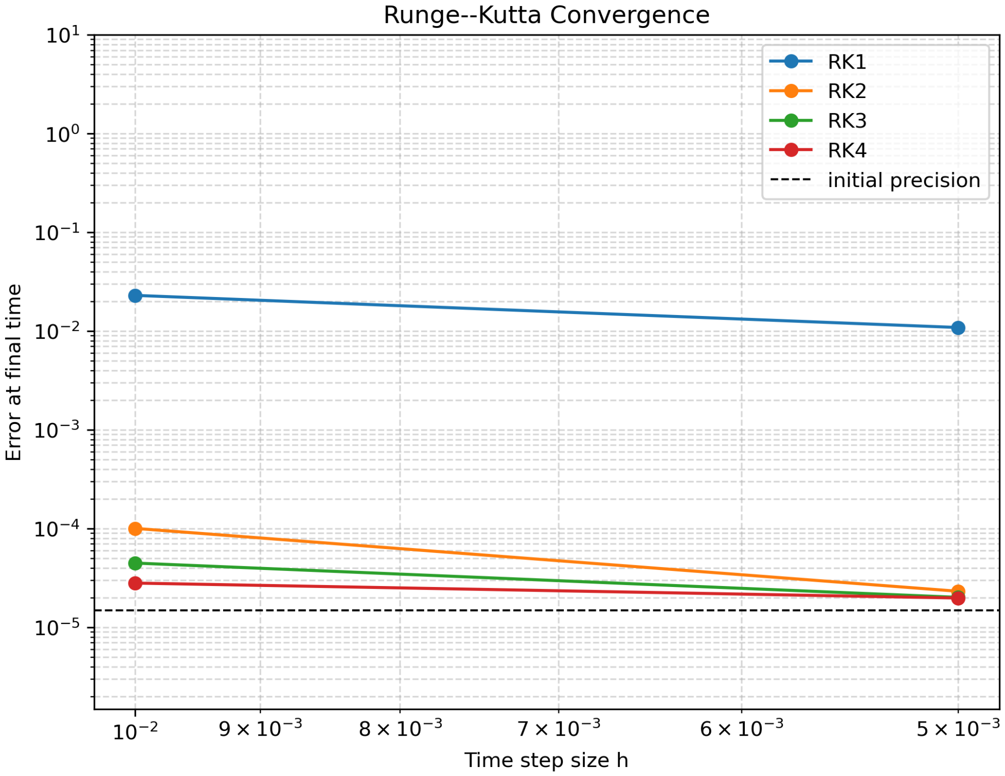


Figure 8: Convergence of implicit Runge–Kutta methods with 1 to 4 stages combined with Pazner's preconditioner.

In terms of runtime, we cannot recognise a significant speed-up as demonstrated in figure 9.
Even though we could show Pazner's preconditioner works as we have the same convergence behaviour, the results could not convince with respect to runtime. However, this is probably because a final time of 0.1 and a number of elements as small as 100 do not rectify the extra expense of assembling the preconditioner because figure 10 and 11 show that the preconditioner is able to reduce the number of iterations significantly. This is definitely a better indicator as the implementations provided are not optimized, but rather educational.

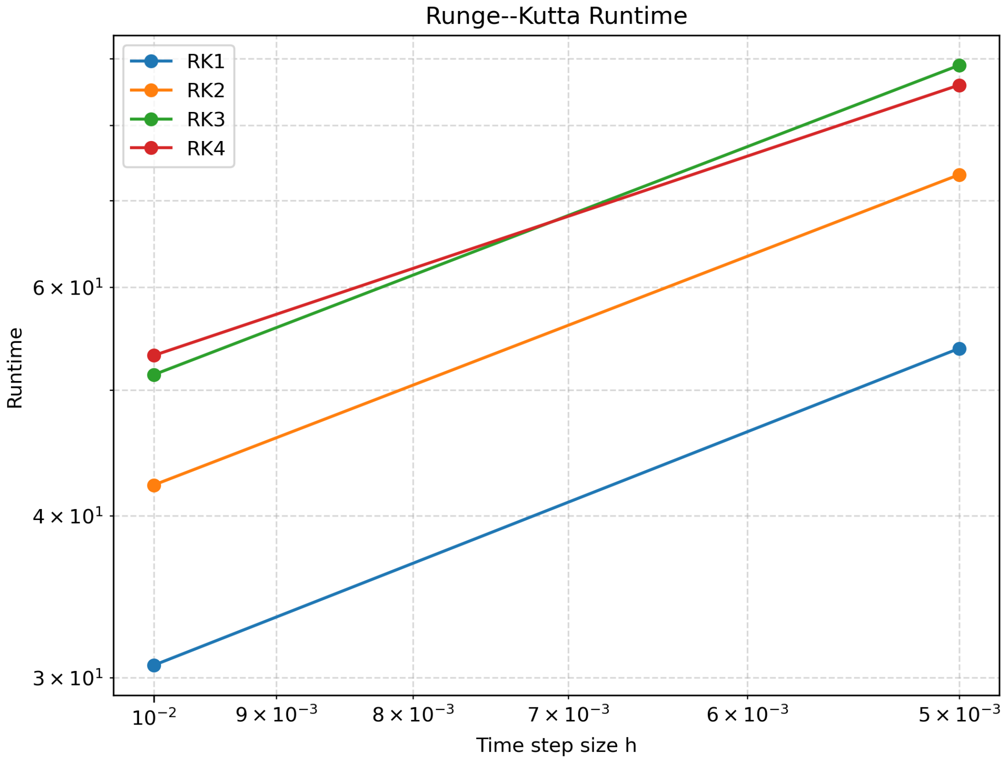


Figure 9: Runtime of implicit Runge–Kutta methods with 1 to 4 stages combined with Pazner's preconditioner.

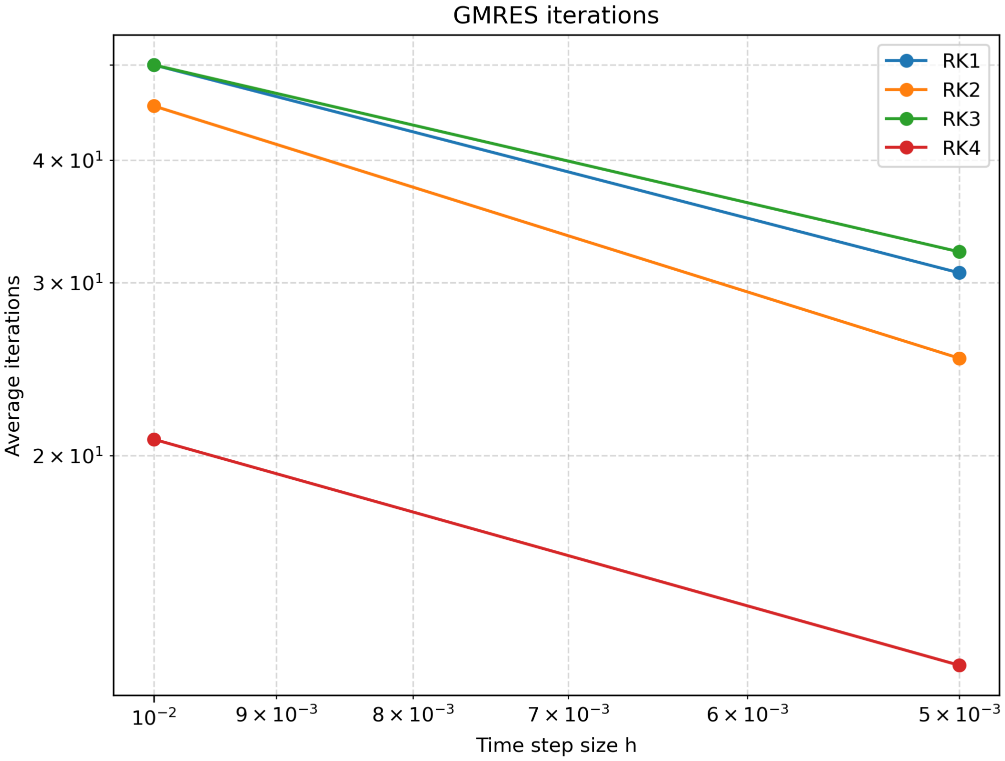


Figure 10: Average number of iterations for implicit Runge–Kutta methods with 1 to 4 stages without preconditioning.

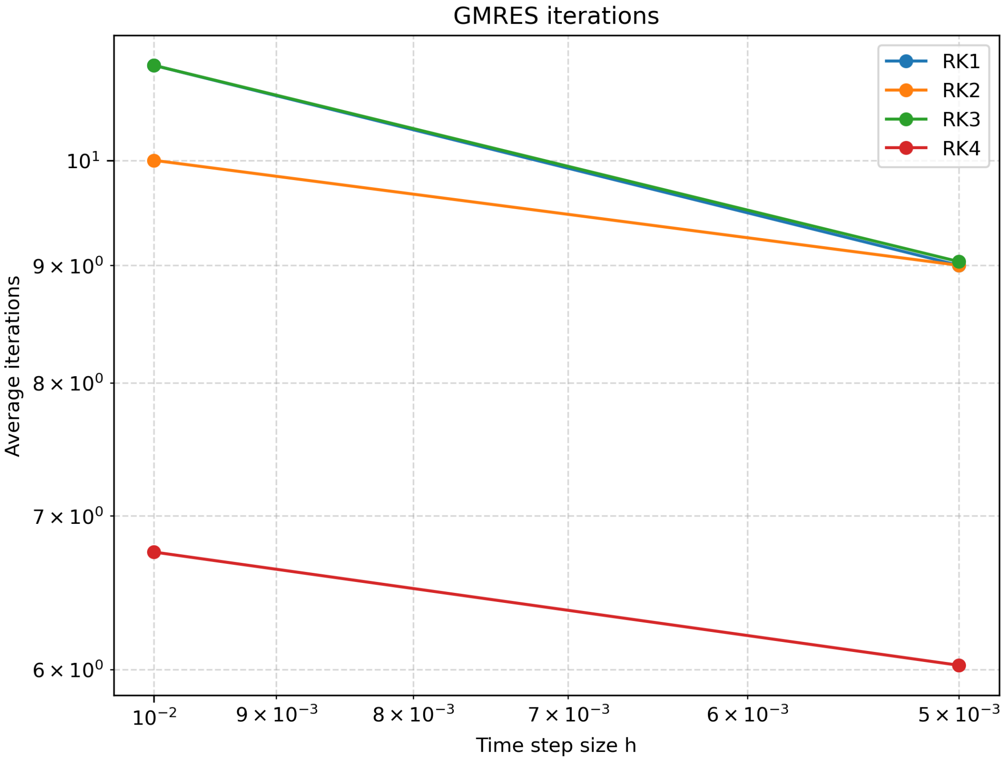


Figure 11: Average number of iterations for implicit Runge–Kutta methods with 1 to 4 stages with Pazner's preconditioner.

Notice that we display the average number of iterations per stage. For the 4-stage ESDIRK method with an explicit first stage, we thus divide by 4 even though we only have to solve an implicit system for 3 stages. This is why GMRES appears to perform much better here.

Outlook

This is a work in progress. Next, we plan on implementing the FDM preconditioner [DM17] for the matrix case in order to conduct a thorough comparison of tensor-product preconditioners. We will then try out a randomized preconditioner based on the RSVD. Depending on the results, an implementation in C++ deal.II will follow as well as an extension to three-dimensional problems.

Bibliography

[ABB^{+}24] Daniel Arndt, Wolfgang Bangerth, Bruno Blais, Marc Fehling, Rene Gassmöller, Timo Heister, Luca Heltai, Sebastian Kinnewig, Martin Kronbichler, Matthias Maier, et al. The deal. ii library, version 9.6. *Journal of Numerical Mathematics*, 32(4):369–380, 2024.

[DM17] Laslo T Diosady and Scott M Murman. Tensor-product preconditioners for higher-order space–time discontinuous galerkin methods. *Journal of Computational Physics*, 330:296–318, 2017.

[Fel25] Tom Feldhausen. 2d_advection_dg: Discontinuous galerkin discretization of 2d linear transport. `https://github.com/tom22-5/2D_advection_dg`, 2025. Accessed: 2025-10-¡day¿.

[GVL13] Gene H Golub and Charles F Van Loan. *Matrix computations*. JHU press, 2013.

[KC16] Christopher A Kennedy and Mark H Carpenter. Diagonally implicit runge-kutta methods for ordinary differential equations. a review. Technical report, 2016.

[KK19] Martin Kronbichler and Katharina Kormann. Fast matrix-free evaluation of discontinuous galerkin finite element operators. *ACM Transactions on Mathematical Software (TOMS)*, 45(3):1–40, 2019.

[KP21] Martin Kronbichler and Per-Olof Persson. *Efficient high-order discretizations for computational fluid dynamics*. Springer, 2021.

[PP18] Will Pazner and Per-Olof Persson. Approximate tensor-product preconditioners for very high order discontinuous galerkin methods. *Journal of computational physics*, 354:344–369, 2018.

[TW23] Joel A. Tropp and Robert J. Webber. Randomized algorithms for low-rank matrix approximation: Design, analysis, and applications. *arXiv preprint arXiv:2306.12418*, 2023.