A semi-intrusive Code Framework for Uncertainty Quantification

Jonas Kusch[a], Jannick Wolters[b], Martin Frank[c]

[a]*Karlsruhe Institute of Technology, Karlsruhe, jonas.kusch@kit.edu*

[b]*Karlsruhe Institute of Technology, Karlsruhe, jannick.wolters@kit.edu*

[c]*Karlsruhe Institute of Technology, Karlsruhe, martin.frank@kit.edu*

Abstract

Methods for quantifying the effects of uncertainties in hyperbolic problems can be divided into intrusive and non-intrusive techniques. A main drawback of intrusive methods is the need to implement new code, whereas collocation methods recycle a given deterministic solver. Furthermore, intrusive methods come with increased computational costs, making it hard to compete with non-intrusive methods. In this work, we present a code framework, which facilitates the implementation of intrusive methods by recycling parts of deterministic codes. Additionally, we introduce methods to decrease numerical costs of intrusive methods for steady problems. We demonstrate the effectiveness of the proposed strategies by comparing results of the uncertain NACA0012 testcase as well as a bent shock tube experiment.

Keywords: uncertainty quantification, conservation laws, hyperbolic, intrusive, stochastic-Galerkin, Collocation, Intrusive Polynomial Moment Method

1. Introduction

Hyperbolic equations play an important role in various research areas such as fluid dynamics or plasma physics. Efficient numerical methods combined with robust implementations are available for these problems, however they do not account for uncertainties which can arise in measurement data or modeling assumptions. Including the effects of uncertainties in differential equations has become an important topic in the last decades.

A general hyperbolic set of equations with random initial data can be written as

$$\partial_t \boldsymbol{u}(t,\boldsymbol{x},\boldsymbol{\xi}) + \nabla\cdot\boldsymbol{f}(\boldsymbol{u}(t,\boldsymbol{x},\boldsymbol{\xi})) = \boldsymbol{0} \quad \text{in } D, \tag{1a}$$

$$\boldsymbol{u}(t=0,\boldsymbol{x},\boldsymbol{\xi}) = \boldsymbol{u}_{\mathrm{IC}}(\boldsymbol{x},\boldsymbol{\xi}), \tag{1b}$$

where the solution $\boldsymbol{u}\in\mathbb{R}^p$ depends on time $t\in\mathbb{R}^+$, spatial position $\boldsymbol{x}\in D\subseteq\mathbb{R}^d$ as well as a vector of random variables $\boldsymbol{\xi}\in\Theta\subseteq\mathbb{R}^s$ with given probability density functions $f_{\Xi,i}(\xi_i)$ for $i=1,\cdots,s$. Hence, the probability density function of $\boldsymbol{\xi}$ is then given by $f_\Xi(\boldsymbol{\xi}) := \prod_{i=1}^s f_{\Xi,i}(\xi_i)$. The physical flux is given by $\boldsymbol{f}:\mathbb{R}^p\to\mathbb{R}^{d\times p}$. To simplify notation, we assume that only the initial condition is random, i.e. $\boldsymbol{\xi}$ enters through the definition of $\boldsymbol{u}_{IC}$. Equations (1) are usually supplemented with boundary conditions, which we will specify later for the individual problems.

Due to the randomness of the solution, one is interested in determining the expectation value or the variance, i.e.

$$\mathrm{E}[\boldsymbol{u}] = \langle\boldsymbol{u}\rangle, \qquad \mathrm{Var}[\boldsymbol{u}] = \langle(\boldsymbol{u}-\mathrm{E}[\boldsymbol{u}])^2\rangle,$$

where we use the bracket operator $\langle\cdot\rangle := \int_\Theta \cdot f_\Xi(\boldsymbol{\xi}) d\xi_1 \cdots d\xi_s$. More generally, one is interested in determining the moments of the solution for a given set of basis functions $\varphi_i : \Theta \to \mathbb{R}$ such that for the multi-index $i = (i_1, \cdots, i_s)$ we have $|i| \leq M$. Note that this yields

$$N := \binom{M+s}{s}$$

basis polynomials when defining $|i| := \sum_{n=1}^{s} |i_n|$. Commonly one chooses orthonormal polynomials as basis functions [?], i.e. $\langle \varphi_i \varphi_j \rangle = \prod_{n=1}^{s} \delta_{i_n j_n}$. The corresponding moments are then given by $\hat{\boldsymbol{u}}_i := \langle \boldsymbol{u} \varphi_i \rangle \in \mathbb{R}^p$. Besides facilitating the computation of the expectation value and variance, the moments can be used to span the solution in $\boldsymbol{\xi}$ [?].

Numerical methods for approximating the moment $\hat{\boldsymbol{u}}_i$ can be divided into intrusive and non-intrusive techniques. A popular non-intrusive method is the stochastic-Collocation (SC) method, see e.g. [? ? ?], which computes the moments with the help of a numerical quadrature rule: For a given set of Q quadrature weights w_k and quadrature points $\boldsymbol{\xi}_k$, the moments are approximated by

$$\hat{\boldsymbol{u}}_i = \langle \boldsymbol{u} \varphi_i \rangle \approx \sum_{k=1}^{Q} w_k \boldsymbol{u}(t, \boldsymbol{x}, \boldsymbol{\xi}_k) \varphi_i(\boldsymbol{\xi}_k) f_\Xi(\boldsymbol{\xi}_k).$$

Since the solution at a fixed quadrature point can be computed by a standard deterministic solver, the SC method does not require a significant implementation effort. Furthermore, SC is embarrassingly parallel, since the required computations can be carried out in parallel on different cores. A downside of collocation methods are aliasing effects, which stem from the inexact approximation of integrals. Furthermore, collocation methods require more runs of the deterministic solver than intrusive methods [? ?]. Despite their easy implementation, collocation methods face challenges when examining unsteady problems, since in this case, the solution needs to be written out, i.e. stored in an external file, at all quadrature points in every time step to compute the time evolution of the moments. This contradicts modern HPC paradigms, which aim at reducing the amount of data produced by numerical methods.

Intrusive methods do not suffer from this problem, since the computation is directly carried out on the moments, i.e. their time evolution can be recorded during the computation. Also for steady problems, the fact that the moments are known in each iteration enables the computation of the stochastic residual, which indicates when to stop the iteration towards the steady state solution. However, intrusive methods are in general more difficult to implement and come along with higher numerical costs. The main idea of these methods is to derive a system of equations for the moments and then implementing a numerical solver for this system: Testing the initial problem (1) with φ_i for $|i| \leq M$ yields

$$\partial_t \hat{\boldsymbol{u}}_i(t, \boldsymbol{x}) + \nabla \cdot \langle \boldsymbol{f}(\boldsymbol{u}(t, \boldsymbol{x}, \cdot)) \varphi_i \rangle = \boldsymbol{0}, \tag{2a}$$

$$\hat{\boldsymbol{u}}_{\boldsymbol{i}}(t = 0, \boldsymbol{x}) = \langle \boldsymbol{u}_{\mathrm{IC}}(\boldsymbol{x}, \cdot) \varphi_i \rangle. \tag{2b}$$

To obtain a closed set of equations, one needs to derive a closure $\mathcal{U}$ such that

$$\boldsymbol{u}(t, \boldsymbol{x}, \boldsymbol{\xi}) \approx \mathcal{U}(\hat{\boldsymbol{u}}; \boldsymbol{\xi}).$$

Here, we collect the N moments for which $|i| \leq M$ holds in the moment matrix $\hat{\boldsymbol{u}} := (\hat{\boldsymbol{u}}_i)_{|i| \leq M} \in \mathbb{R}^{N \times p}$. The dependency of $\mathcal{U}$ on $\boldsymbol{\xi}$ will occasionally be omitted in the following for sake of readability. A commonly used closure is given by stochastic-Galerkin (SG) [?], which represents the solution by a polynomial:

$$\mathcal{U}_{\mathrm{SG}}(\hat{\boldsymbol{u}}; \boldsymbol{\xi}) := \sum_{|i| \leq M} \hat{\boldsymbol{u}}_i \varphi_i(\boldsymbol{\xi}) = \hat{\boldsymbol{u}}^T \boldsymbol{\varphi}(\boldsymbol{\xi}), \tag{3}$$

Corresponding to the definition of $\hat{\boldsymbol{u}}$, we collect the N basis function for which $|i| \leq M$ holds in $\boldsymbol{\varphi} := (\varphi_i)_{|i| \leq M} \in \mathbb{R}^{N+1}$. When using the stochastic-Galerkin method to close (2), the resulting moment system is not necessarily hyperbolic [?] and the solution needs to be manipulated [?] in order to prevent a failure of the method. A generalization of stochastic-Galerkin, which ensures hyperbolicity is the Intrusive Polynomial Moment (IPM) closure [?]: The closure is given by a constraint optimization problem. For a given convex entropy $s : \mathbb{R}^p \to \mathbb{R}$ for the original problem (1), this optimization problem is given by

$$\mathcal{U}(\hat{\boldsymbol{u}}) = \arg\min_{\boldsymbol{u}} \langle s(\boldsymbol{u}) \rangle \quad \text{subject to } \hat{\boldsymbol{u}}_i = \langle \boldsymbol{u}\varphi_i \rangle \text{ for } |i| \leq M. \tag{4}$$

Rewritten in its dual form, (4) is transformed into an unconstraint optimization problem. Defining the variables $\boldsymbol{\lambda}_i \in \mathbb{R}^p$ where i is again a multi index, gives the unconstrained dual problem

$$\hat{\boldsymbol{\lambda}}(\hat{\boldsymbol{u}}) := \arg\min_{\boldsymbol{\lambda} \in \mathbb{R}^{N \times p}} \left\{ \langle s_*(\boldsymbol{\lambda}^T \boldsymbol{\varphi}) \rangle - \sum_{|i| \leq M} \boldsymbol{\lambda}_i^T \hat{\boldsymbol{u}}_i \right\}, \tag{5}$$

where $s_* : \mathbb{R}^p \to \mathbb{R}$ is the Legendre transformation of s, and $\hat{\boldsymbol{\lambda}} := (\hat{\boldsymbol{\lambda}}_i)_{|i| \leq M} \in \mathbb{R}^{N \times p}$ are called the dual variables. The solution to (4) is then given by

$$\mathcal{U}(\hat{\boldsymbol{u}}) = (\nabla_{\boldsymbol{u}} s)^{-1} (\hat{\boldsymbol{\lambda}}(\hat{\boldsymbol{u}})^T \boldsymbol{\varphi}). \tag{6}$$

Plugging the derived closure into the moment system (2), one obtains the IPM moment system

$$\partial_t \hat{\boldsymbol{u}}_i(t, \boldsymbol{x}) + \nabla \cdot \langle \boldsymbol{f}(\mathcal{U}(\hat{\boldsymbol{u}}))\varphi_i \rangle = \boldsymbol{0}, \tag{7a}$$

$$\hat{\boldsymbol{u}}_i(t = 0, \boldsymbol{x}) = \langle \boldsymbol{u}_{\text{IC}}(\boldsymbol{x}, \cdot)\varphi_i \rangle. \tag{7b}$$

with $|i| \leq M$. The IPM method has several advantages: Choosing the entropy $s(\boldsymbol{u}) = \frac{1}{2}\boldsymbol{u}^T\boldsymbol{u}$ yields the stochastic-Galerkin closure, i.e. IPM generalizes different intrusive methods. Furthermore, at least for scalar problems, IPM is significantly less oscillatory compared to SG [?]. Also, as discussed in [?], when choosing $s(\boldsymbol{u})$ to be a physically correct entropy of the deterministic problem, the IPM solution dissipates the expectation value of the entropy, which is

$$S(\hat{\boldsymbol{u}}) := \langle s(\mathcal{U}(\hat{\boldsymbol{u}})) \rangle,$$

i.e. the IPM method yields a physically correct entropy solution. This again underlines a weakness of stochastic-Galerkin: If $s(\boldsymbol{u}) = \frac{1}{2}\boldsymbol{u}^T\boldsymbol{u}$ is not a correct entropy of the original problem, the SG method can lead to non-physical solution values, which can then cause a failure of the method. The main weakness of the IPM method is its run time, since it requires the repeated evaluation of (6), which involves solving the optimization problem (5). Hence, the desirable properties of IPM come along with significantly increased numerical costs. However, IPM and minimal entropy methods in general are well suited for modern HPC architecture, which can be used to reduce the run time [?].

When studying hyperbolic equations, the moment approximations of various methods such as Stochastic Galerkin [?], IPM [?] and stochastic-Collocation [? ?] tend to show incorrect discontinuities in certain regions of the physical space. These non-physical structures dissolve when the number of basis functions is increased [? ?] or when artificial diffusion is added through the spatial numerical method [?] or filters [?]. Also, a multi-element approach which divides the uncertain domain into cells and uses piece-wise polynomial basis functions to represent the solution has proven to mitigate non-physical discontinuities [?]. These structures commonly arise on a small portion of the space-time domain. Therefore, intrusive methods seem to be an adequate choice since they are well suited for adaptive strategies. By locally increasing the polynomial order [? ? ?] or adding artificial viscosity [?] at certain spatial positions and time steps in which complex structures such as discontinuities occur, a given accuracy can be reached with significantly reduced numerical costs. The main task here is to find an adequate refinement indicator.

In this paper, we present a semi-intrusive code framework, which facilitates the task of implementing general intrusive methods. The framework only requires the numerical flux of the deterministic problem (as well as an entropy if IPM is used). We thereby provide the ability to recycle existing implementations of deterministic solvers. Furthermore, we investigate intrusive methods for steady problems and compare them against collocation methods. The steady setting provides different opportunities to take advantage of features of intrusive methods:

- Accelerate the convergence to the IPM steady state solution by applying IPM as a post-processing step for collocation methods: We converge the moments of the solution to a steady state with an inaccurate, but cheap collocation method and then use the resulting collocation moments as starting values for an expensive but accurate intrusive method such as IPM, which we then again converge to steady state.
- Compute inexact dual variables (5) for IPM: Since the moments during the iteration process are inaccurate, i.e. they are not the correct steady state solution, we propose to not fully converge the dual iteration, which computes (5).

In addition to these two strategies for steady problems, we make use of adaptivity: The intrusive nature of SG and IPM can be used to locally increase the number of moments whenever the solution has a complex structure in the random variable (as well as decrease the number of moments if not). We test the effectiveness of these acceleration ideas by comparing results with stochastic-Collocation for the uncertain NACA test case as well as a bent shock tube problem.

The paper is structured as follows: After the introduction, we discuss the numerical discretization as well as the implementation and structure of the semi-intrusive framework in section 2. Section 3.1 discusses the IPM acceleration with a non-intrusive method and in section 3.2, we discuss the idea of not converging the dual iteration. Section 4 extends the presented numerical framework to an algorithm making use of adaptivity. A comparison of results computed with the presented methods is then given in 5, followed by a summary and outlook in section 6.

2. Discretization and code framework

2.1. Discretization

In the following, we discretize the moment system in space and time according to [**?**]. Due to the fact, that stochastic-Galerkin can be interpreted as IPM with a quadratic entropy, it suffices to only derive a discretization of the IPM moment system (7). Omitting initial conditions and assuming a one-dimensional spatial domain, we can write this system as

$$\partial_t \hat{\boldsymbol{u}} + \partial_x \boldsymbol{F}(\hat{\boldsymbol{u}}) = \boldsymbol{0}$$

with the flux $\boldsymbol{F} : \mathbb{R}^{N\times p} \to \mathbb{R}^{N\times p}$, $\boldsymbol{F}(\hat{\boldsymbol{u}}) = \langle \boldsymbol{f}(\mathcal{U}(\hat{\boldsymbol{u}}))\boldsymbol{\varphi}^T\rangle^T$. Due to hyperbolicity of the IPM moment system, one can use a finite-volume method to approximate the time evolution of the IPM moments. First we perform the discretization of the spatial domain: We choose the discrete unknowns which represent the solution to be the spatial averages over each cell at time t_n, given by

$$\hat{\boldsymbol{u}}_{ij}^n \simeq \frac{1}{\Delta x}\int_{x_{j-1/2}}^{x_{j+1/2}} \hat{\boldsymbol{u}}_i(t_n, x)dx.$$

If a moment vector in cell j at time t_n is denoted as $\hat{\boldsymbol{u}}_j^n = (\hat{\boldsymbol{u}}_{0j}^n, \cdots, \hat{\boldsymbol{u}}_{Nj}^n)^T \in \mathbb{R}^{N+1}$, the finite-volume scheme can be written in conservative form with the numerical flux $\boldsymbol{G} : \mathbb{R}^{N\times p} \times \mathbb{R}^{N\times p} \to \mathbb{R}^{N\times p}$ as

$$\hat{\boldsymbol{u}}_j^{n+1} = \hat{\boldsymbol{u}}_j^n - \frac{\Delta t}{\Delta x}\left(\boldsymbol{G}(\hat{\boldsymbol{u}}_j^n, \hat{\boldsymbol{u}}_{j+1}^n) - \boldsymbol{G}(\hat{\boldsymbol{u}}_{j-1}^n, \hat{\boldsymbol{u}}_j^n)\right) \tag{8}$$

for $j = 1, \cdots, N_x$ and $n = 0, \cdots, N_t$, where N_x is the number of spatial cells and N_t is the number of time steps. The numerical flux is assumed to be consistent, i.e. $\boldsymbol{G}(\hat{\boldsymbol{u}}, \hat{\boldsymbol{u}}) = \boldsymbol{F}(\hat{\boldsymbol{u}})$.

When a consistent numerical flux $\boldsymbol{g} : \mathbb{R}^p \times \mathbb{R}^p \to \mathbb{R}^p$, $\boldsymbol{g} = \boldsymbol{g}(\boldsymbol{u}_\ell, \boldsymbol{u}_r)$ is available for the original problem (1), then for the IPM system we can simply take

$$\tilde{\boldsymbol{G}}(\hat{\boldsymbol{u}}_j^n, \hat{\boldsymbol{u}}_{j+1}^n) = \langle \boldsymbol{g}(\mathcal{U}(\hat{\boldsymbol{u}}_j^n), \mathcal{U}(\hat{\boldsymbol{u}}_{j+1}^n))\boldsymbol{\varphi}^T \rangle^T.$$

Commonly, this integral cannot be evaluated analytically and therefore needs to be approximated by a quadrature rule

$$\langle h \rangle \approx \langle h \rangle_Q := \sum_{k=1}^{Q} w_k h(\boldsymbol{\xi}_k) f_\Xi(\boldsymbol{\xi}_k).$$

The approximated numerical flux then becomes

$$\boldsymbol{G}(\hat{\boldsymbol{u}}_j^n, \hat{\boldsymbol{u}}_{j+1}^n) = \langle \boldsymbol{g}(\mathcal{U}(\hat{\boldsymbol{u}}_j^n), \mathcal{U}(\hat{\boldsymbol{u}}_{j+1}^n))\boldsymbol{\varphi}^T \rangle_Q^T. \tag{9}$$

Note that the numerical flux requires evaluating the closure $\mathcal{U}(\hat{\boldsymbol{u}}_j^n)$. To simplify notation, we define $\boldsymbol{u_s} : \mathbb{R}^s \to \mathbb{R}^s$,

$$\boldsymbol{u}_s(\boldsymbol{\Lambda}) := (\nabla_{\boldsymbol{u}} s)^{-1}(\boldsymbol{\Lambda}),$$

meaning that the closure (6) at cell j in timestep n can be written as

$$\mathcal{U}(\hat{\boldsymbol{u}}_j^n) = \boldsymbol{u}_s(\hat{\boldsymbol{\lambda}}(\hat{\boldsymbol{u}}_j^n)^T \boldsymbol{\varphi}).$$

The computation of the so called entropy variables $\hat{\boldsymbol{\lambda}}_j^n := \hat{\boldsymbol{\lambda}}(\hat{\boldsymbol{u}}_j^n)$ requires solving the dual problem (5) for the moment vector $\hat{\boldsymbol{u}}_j^n$. Hence, to determine the dual variables for a given moment vector $\hat{\boldsymbol{u}}$, the cost function

$$L(\boldsymbol{\lambda}; \hat{\boldsymbol{u}}) := \langle s_*(\boldsymbol{\lambda}^T \boldsymbol{\varphi}) \rangle_Q - \sum_{i \leq M} \boldsymbol{\lambda}_i^T \hat{\boldsymbol{u}}_i$$

needs to be minimized, i.e. one needs to find the root of

$$\nabla_{\boldsymbol{\lambda}_i} L(\boldsymbol{\lambda}; \hat{\boldsymbol{u}}) = \langle \nabla s_*(\boldsymbol{\lambda}^T \boldsymbol{\varphi}) \boldsymbol{\varphi}^T \rangle_Q^T - \hat{\boldsymbol{u}}_i,$$

which is usually done by an iterative method. Defining the dual iteration function $\boldsymbol{d} : \mathbb{R}^{N \times p} \times \mathbb{R}^{N \times p} \to \mathbb{R}^{N \times p}$,

$$\boldsymbol{d}(\boldsymbol{\lambda}, \hat{\boldsymbol{u}}) := \boldsymbol{\lambda} - \boldsymbol{B}(\boldsymbol{\lambda}) \cdot \left(\langle \boldsymbol{u}_s(\boldsymbol{\lambda}^T \boldsymbol{\varphi}) \boldsymbol{\varphi}^T \rangle_Q^T - \hat{\boldsymbol{u}} \right), \tag{10}$$

with an adequate preconditioner $\boldsymbol{B}$, the iteration process for spatial cell j is given by

$$\boldsymbol{\lambda}_j^{(m+1)} = \boldsymbol{d}(\boldsymbol{\lambda}_j^{(m)}, \hat{\boldsymbol{u}}_{\boldsymbol{j}}). \tag{11}$$

The exact dual state is then obtained by computing the fix point of $\boldsymbol{d}$, meaning that one converges the iteration (11), i.e. $\hat{\boldsymbol{\lambda}}_j^n := \hat{\boldsymbol{\lambda}}(\hat{\boldsymbol{u}}_{\boldsymbol{j}}^{\boldsymbol{n}}) = \lim_{m \to \infty} \boldsymbol{d}(\boldsymbol{\lambda}_j^{(m)}, \hat{\boldsymbol{u}}_{\boldsymbol{j}}^{\boldsymbol{n}})$. A common preconditioner is the inverse Hessian of the dual problem, i.e.

$$\boldsymbol{B}(\boldsymbol{\lambda}) := \langle \nabla \boldsymbol{u_s}(\boldsymbol{\lambda}^T \boldsymbol{\varphi}) \boldsymbol{\varphi} \boldsymbol{\varphi}^T \rangle_Q^{-T}.$$

To obtain a finite number of iterations for the iteration in cell j, a stopping criterion

$$\sum_{i=0}^{p} \left\| \nabla_{\boldsymbol{\lambda}_i} L(\boldsymbol{\lambda}_j^{(m)}; \hat{\boldsymbol{u}}_j^n) \right\| < \tau \tag{12}$$

is used.

We now write down the entire scheme: To obtain a more compact notation, one defines

$$\boldsymbol{c}\left(\boldsymbol{\lambda}_\ell, \boldsymbol{\lambda}_c, \boldsymbol{\lambda}_r\right) := \langle \boldsymbol{u}_s(\boldsymbol{\lambda}_c^T\boldsymbol{\varphi})\boldsymbol{\varphi}^T\rangle_Q^T - \frac{\Delta t}{\Delta x}\left(\langle \boldsymbol{g}(\boldsymbol{u}_s(\boldsymbol{\lambda}_c^T\boldsymbol{\varphi}), \boldsymbol{u}_s(\boldsymbol{\lambda}_r^T\boldsymbol{\varphi}))\boldsymbol{\varphi}^T\rangle_Q^T - \langle \boldsymbol{g}(\boldsymbol{u}_s(\boldsymbol{\lambda}_\ell^T\boldsymbol{\varphi}), \boldsymbol{u}_s(\boldsymbol{\lambda}_c^T\boldsymbol{\varphi}))\boldsymbol{\varphi}^T\rangle_Q^T\right). \tag{13}$$

The moment iteration is then given by

$$\hat{\boldsymbol{u}}_j^{n+1} = \boldsymbol{c}\left(\hat{\boldsymbol{\lambda}}(\hat{\boldsymbol{u}}_{j-1}^n), \hat{\boldsymbol{\lambda}}(\hat{\boldsymbol{u}}_j^n), \hat{\boldsymbol{\lambda}}(\hat{\boldsymbol{u}}_{j+1}^n)\right), \tag{14}$$

where the map from the moment vector to the dual variables, i.e. $\boldsymbol{\lambda}(\hat{\boldsymbol{u}}_j^n)$, is obtained by iterating

$$\boldsymbol{\lambda}_j^{(m+1)} = \boldsymbol{d}(\boldsymbol{\lambda}_j^{(m)}; \hat{\boldsymbol{u}}_j^n). \tag{15}$$

until condition (12) is fulfilled. This gives Algorithm 1.

Algorithm 1 IPM implementation

```
1:  for $j = 0$ to $N_x + 1$ do
2:      $\boldsymbol{u}_j^0 \leftarrow \frac{1}{\Delta x}\int_{x_{j-1/2}}^{x_{j+1/2}} \langle u_{\text{IC}}(x,\cdot)\boldsymbol{\varphi}\rangle_Q dx$
3:  for $n = 0$ to $N_t$ do
4:      for $j = 0$ to $N_x + 1$ do
5:          $\boldsymbol{\lambda}_j^{(0)} \leftarrow \hat{\boldsymbol{\lambda}}_j^n$
6:          while (12) is violated do
7:              $\boldsymbol{\lambda}_j^{(m+1)} \leftarrow \boldsymbol{d}(\boldsymbol{\lambda}_j^{(m)}; \hat{\boldsymbol{u}}_j^n)$
8:              $m \leftarrow m + 1$
9:          $\hat{\boldsymbol{\lambda}}_j^{n+1} \leftarrow \boldsymbol{\lambda}_j^{(m)}$
10:     for $j = 1$ to $N_x$ do
11:         $\hat{\boldsymbol{u}}_j^{n+1} \leftarrow \boldsymbol{c}(\hat{\boldsymbol{\lambda}}_{j-1}^{n+1}, \hat{\boldsymbol{\lambda}}_j^{n+1}, \hat{\boldsymbol{\lambda}}_{j+1}^{n+1})$
```

2.2. Costs of the numerical flux

A straight forward implementation is ensured by the choice of the numerical flux (9), which has been proposed in [**?**]. By simply taking moments of a given deterministic flux, the software can easily be extended to various physical problems whenever an implementation of $\boldsymbol{g} = \boldsymbol{g}(\boldsymbol{u}_\ell, \boldsymbol{u}_r)$ is available. However, the need to approximate the flux by an integral seems to be numerically expensive, especially when considering that the stochastic-Galerkin method often allows an analytic computation of all arising integrals. In the following, we discuss this issue and show the advantages of our numerical flux choice by considering the stochastic Burgers' equation

$$\partial_t u + \partial_x \frac{u^2}{2} = 0,$$
$$u(t = 0, x, \xi) = u_{IC}(x, \xi).$$

The scalar random variable ξ is uniformly distributed in the interval $[-1, 1]$, hence the gPC basis functions $\boldsymbol{\varphi} = (\varphi_0, \cdots, \varphi_M)^T$ are the Legendre polynomials. Testing with these basis functions and choosing the SG closure (3) yields the closed SG moment system

$$\partial_t \hat{u}_i + \partial_x \frac{1}{2} \sum_{n,m=0}^{M} \hat{u}_n \hat{u}_m \langle \varphi_n \varphi_m \varphi_i \rangle = 0.$$

Defining the matrices $\boldsymbol{C}_i := \langle \boldsymbol{\varphi}\boldsymbol{\varphi}^T \varphi_i \rangle \in \mathbb{R}^{M\times M}$ gives

$$\partial_t \hat{\boldsymbol{u}} + \partial_x \boldsymbol{F}(\hat{\boldsymbol{u}}) = 0$$

with $F_i(\hat{\boldsymbol{u}}) = \frac{1}{2}\hat{\boldsymbol{u}}^T \boldsymbol{C}_i \hat{\boldsymbol{u}}$. Note that $\boldsymbol{C}_i$ can be computed analytically, hence choosing a Lax-Friedrichs flux

$$G_i^{(LF)}(\hat{\boldsymbol{u}}_\ell, \hat{\boldsymbol{u}}_r) = \frac{1}{4}\left(\hat{\boldsymbol{u}}_\ell^T \boldsymbol{C}_i \hat{\boldsymbol{u}}_\ell + \hat{\boldsymbol{u}}_r^T \boldsymbol{C}_i \hat{\boldsymbol{u}}_r\right) - \frac{\Delta x}{2\Delta t}(\hat{\boldsymbol{u}}_r - \hat{\boldsymbol{u}}_\ell)_i \tag{16}$$

requires no integral evaluations. Recall, that the numerical flux choice made in this work gives

$$\boldsymbol{G}(\hat{\boldsymbol{u}}_\ell, \hat{\boldsymbol{u}}_r) = \sum_{k=1}^{Q} w_k g(\mathcal{U}_{SG}(\hat{\boldsymbol{u}}_\ell; \xi_k), \mathcal{U}_{SG}(\hat{\boldsymbol{u}}_r; \xi_k))\boldsymbol{\varphi}(\xi_k) f_\Xi(\xi_k). \tag{17}$$

When the chosen deterministic flux g is Lax-Friedrichs, the order of the polynomials inside the sum is of order $3M = 3(N-1)$. Choosing a Gauss-Lobatto quadrature rule, $Q = \frac{3}{2}N - 1$ quadrature points suffice for an exact computation of the numerical flux. Indeed, with this choice of quadrature points, the numerical fluxes (16) and (17) are equivalent. Counting the number of operations, one observes that our choice of the numerical flux (17) is significantly cheaper: When computing and storing the values in a matrix $\boldsymbol{A} \in \mathbb{R}^{Q\times N}$ with entries $a_{ki} = \varphi_i(\xi_k)$ before running the program, the numerical flux (17) can be split into two parts. First, we determine the SG solution at all quadrature points, i.e. we compute $\boldsymbol{u}^{(\ell)} := \boldsymbol{A}\hat{\boldsymbol{u}}_\ell$ and $\boldsymbol{u}^{(r)} := \boldsymbol{A}\hat{\boldsymbol{u}}_r$ which requires $O(N\cdot Q)$ operations. These solution values are then used to compute the numerical flux

$$G_i(\hat{\boldsymbol{u}}_\ell, \hat{\boldsymbol{u}}_r) = \sum_{k=1}^{Q} w_k g(u_k^{(\ell)}, u_k^{(r)}) a_{ki} f_\Xi(\xi_k),$$

which again requires $O(N\cdot Q)$ operations, i.e. the costs are $O(N^2)$. The evaluation of (16) however requires $O(N^3)$ operations.

2.3. Code framework

TODO: Code structure, why is it easy to include different equations?

3. Strategies for steady problems

In the following, we look at steady state problems, i.e. we have

$$\nabla \cdot \boldsymbol{f}(\boldsymbol{u}(\boldsymbol{x}, \boldsymbol{\xi})) = \boldsymbol{0} \quad \text{in } D \tag{18}$$

with adequate boundary conditions. A general strategy for computing the steady state solution to (18) is to introduce a pseudo-time and numerically treat (18) as an unsteady problem. A steady state solution is then obtained by iterating in pseudo-time until the solution remains constant. It is important to point out that the time it takes to converge to a steady state solution is crucially affected by the chosen initial condition and its distance to the steady state solution. Similar to the unsteady case (1), we can again derive a moment system for (18) given by

$$\nabla \cdot \langle \boldsymbol{f}(\boldsymbol{u}(\boldsymbol{x}, \boldsymbol{\xi}))\boldsymbol{\varphi}^T \rangle^T = \boldsymbol{0} \quad \text{in } D \tag{19}$$

which is again needed for the construction of intrusive methods. By adding a pseudo-time and using the IPM closure, we obtain the same system as in (7), i.e. Algorithm 1 can be used to iterate to a steady state solution. Note that now, the time iteration is not performed for a fixed number of time steps N_t, but until the residual

$$\sum_{j=1}^{N_x} \Delta x_j \|\hat{\boldsymbol{u}}_j^n - \hat{\boldsymbol{u}}_j^{n-1}\| \leq \varepsilon \tag{20}$$

is fulfilled. Since one is generally interested in low order moments such as the expectation value, this residual can be modified by only accounting for the zero order moments.

3.1. Collocation accelerated IPM

Commonly, a great amount of iterations in pseudo-time are needed to converge to a steady state solution. Consequently, the IPM method which requires solving the dual problem (5) in every spacial cell in each iteration becomes prohibitively expensive. We tackle this problem by using IPM only as a postprocesssing step for the steady solution obtained by a cheap method. (Or vice-versa, we use a cheap method as a preprocessing step for IPM). In our case, we perform the preprocessing step with stochastic-Collocation, i.e. we converge the moments to a steady state solution by applying collocation steps. The obtained moments are then used as initial condition for the IPM moment system (for which the moments are no longer a steady state solution). After applying a significantly reduced number of IPM iterations, we obtain a steady state IPM solution. In our numerical experiments presented in section 5, we can show that the overall costs are dominated by the large number of cheap collocation steps and not by the small number of expensive IPM steps, while the solution shows the expected desirable properties of the IPM solution.

Different variants of this method are possible:

- Since the IPM iterations will again modify the steady state Collocation solution, it is not necessary to converge Collocation to the exact steady state solution before starting IPM. Here, one needs to determine an indicator to choose at which residual the collocation iteration is sufficiently accurate and can therefore be switched to IPM.
- The main idea is to use a cheap but inexact method as a preconditioner for an expensive but accurate method. Here, one is not limited in choosing SC and IPM, but one can for example choose Monte Carlo methods as an accelerator or SC with an increased number of quadrature points as the expensive method. Note that in the latter case, a map from the moments to the solution at the increased quadrature set is required, for which the IPM reconstruction (6) would be a suitable choice.

3.2. One-shot IPM

In this section we aim at breaking up the inner loop in the IPM algorithm 1, i.e. to just perform one iteration of the dual problem in each time step. Consequently, the IPM reconstruction given by (4) is not done exactly, meaning that the reconstructed solution does not minimize the entropy while not fulfilling the moment constraint. However, the fact that the moment vectors are not yet converged to the steady solution seems to permit such an inexact reconstruction. Hence, we aim at iterating the moments to steady state and the dual variables to the exact solution of the IPM optimization problem (4) simultaneously. Hence, by successively performing one update of the moment iteration and one update of the dual iteration, we obtain

$$\boldsymbol{\lambda}_j^{n+1} = \boldsymbol{d}(\boldsymbol{\lambda}_j^n, \boldsymbol{u}_j^n) \quad \text{for all j} \tag{21a}$$
$$\boldsymbol{u}_j^{n+1} = \boldsymbol{c}\left(\boldsymbol{\lambda}_{j-1}^{n+1}, \boldsymbol{\lambda}_j^{n+1}, \boldsymbol{\lambda}_{j+1}^{n+1}\right). \tag{21b}$$

This gives algorithm

Algorithm 2 One-shot IPM implementation

```
1: for $j = 0$ to $N_x + 1$ do
2:     $\boldsymbol{u}_j^0 \leftarrow \frac{1}{\Delta x}\int_{x_{j-1/2}}^{x_{j+1/2}} \langle u_{\text{IC}}(x,\cdot)\boldsymbol{\varphi}\rangle_Q dx$
3: while (20) is violated do
4:     for $j = 1$ to $N_x$ do
5:         $\boldsymbol{\lambda}_j^{n+1} \leftarrow \boldsymbol{d}(\boldsymbol{\lambda}_j^n; \hat{\boldsymbol{u}}_j^n)$
6:         $\hat{\boldsymbol{u}}_j^{n+1} \leftarrow \boldsymbol{c}(\boldsymbol{\lambda}_{j-1}^{n+1}, \boldsymbol{\lambda}_j^{n+1}, \boldsymbol{\lambda}_{j+1}^{n+1})$
7:     $n \leftarrow n + 1$
```

We call this method One-Shot IPM, since it is inspired by One-shot optimization, see for example [?], which uses only a single iteration of the primal and dual step in order to update the design variables. Note that the dual variables from the One-Shot iteration are written without a hat to indicate that they are not the exact solution of the dual problem.

In the following, we will show that this iteration converges, if the chosen initial condition is sufficiently close to the steady state solution. For this we take an approach commonly chosen to prove local convergence properties of Newton's method: In Theorem 1, we show that the iteration function is contractive at its fix point and conclude in Theorem 2 that this yields local convergence:

Theorem 1. *Assume that the classical IPM iteration is contractive at its fix point $\hat{\boldsymbol{u}}^*$. Then the Jacobi matrix $\boldsymbol{J}$ of the One-Shot IPM iteration (21) has a spectral radius $\rho(\boldsymbol{J}) < 1$ at the fix point $(\boldsymbol{\lambda}^*, \hat{\boldsymbol{u}}^*)$ if the preconditioner $\boldsymbol{B} = \langle \nabla \boldsymbol{u}_s(\boldsymbol{\varphi}^T \boldsymbol{\lambda}_j^n)\boldsymbol{\varphi}\boldsymbol{\varphi}^T\rangle^{-1}$, i.e. the Hessian of the dual problem is used.*

Proof. First, to understand what contraction of the classical IPM iteration implies, we rewrite the moment iteration (14) of the classical IPM scheme: When defining the update function

$$\tilde{\boldsymbol{c}}\left(\hat{\boldsymbol{u}}_\ell, \hat{\boldsymbol{u}}_c, \hat{\boldsymbol{u}}_r\right) := \boldsymbol{c}\left(\hat{\boldsymbol{\lambda}}(\hat{\boldsymbol{u}}_\ell), \hat{\boldsymbol{\lambda}}(\hat{\boldsymbol{u}}_c), \hat{\boldsymbol{\lambda}}(\hat{\boldsymbol{u}}_r)\right)$$

we can rewrite the classical moment iteration as

$$\hat{\boldsymbol{u}}_j^{n+1} = \tilde{\boldsymbol{c}}\left(\hat{\boldsymbol{u}}_{j-1}^n, \hat{\boldsymbol{u}}_j^n, \hat{\boldsymbol{u}}_{j+1}^n\right). \tag{22}$$

Since we assume that the classical IPM scheme is contractive at its fix point, we have $\rho(\nabla_{\hat{\boldsymbol{u}}}\tilde{\boldsymbol{c}}(\hat{\boldsymbol{u}}^*)) < 1$ with $\nabla_{\hat{\boldsymbol{u}}}\tilde{\boldsymbol{c}} \in \mathbb{R}^{N\cdot N_x \times N \cdot N_x}$ defined by

$$\nabla_{\hat{\boldsymbol{u}}}\tilde{\boldsymbol{c}} = \begin{pmatrix} \partial_{\hat{\boldsymbol{u}}_c}\tilde{\boldsymbol{c}}_1 & \partial_{\hat{\boldsymbol{u}}_r}\tilde{\boldsymbol{c}}_1 & 0 & 0 & \dots \\ \partial_{\hat{\boldsymbol{u}}_\ell}\tilde{\boldsymbol{c}}_2 & \partial_{\hat{\boldsymbol{u}}_c}\tilde{\boldsymbol{c}}_2 & \partial_{\hat{\boldsymbol{u}}_r}\tilde{\boldsymbol{c}}_2 & 0 & \dots \\ 0 & \partial_{\hat{\boldsymbol{u}}_\ell}\tilde{\boldsymbol{c}}_3 & \partial_{\hat{\boldsymbol{u}}_c}\tilde{\boldsymbol{c}}_3 & \partial_{\hat{\boldsymbol{u}}_r}\tilde{\boldsymbol{c}}_3 & \\ \vdots & & & \ddots & \\ 0 & \cdots & 0 & \partial_{\hat{\boldsymbol{u}}_\ell}\tilde{\boldsymbol{c}}_{N_x} & \partial_{\hat{\boldsymbol{u}}_c}\tilde{\boldsymbol{c}}_{N_x} \end{pmatrix},$$

where we define $\tilde{\boldsymbol{c}}_j := \tilde{\boldsymbol{c}}\left(\hat{\boldsymbol{u}}_{j-1}^*, \hat{\boldsymbol{u}}_j^*, \hat{\boldsymbol{u}}_{j+1}^*\right)$ for all j. Now for each term inside the matrix $\nabla_{\hat{\boldsymbol{u}}}\tilde{\boldsymbol{c}}$ we have

$$\partial_{\hat{\boldsymbol{u}}_\ell}\tilde{\boldsymbol{c}}_j = \frac{\partial \boldsymbol{c}_j}{\partial \hat{\boldsymbol{\lambda}}_\ell}\frac{\partial \hat{\boldsymbol{\lambda}}(\hat{\boldsymbol{u}}_{j-1}^*)}{\partial \hat{\boldsymbol{u}}}, \quad \partial_{\hat{\boldsymbol{u}}_c}\tilde{\boldsymbol{c}}_j = \frac{\partial \boldsymbol{c}_j}{\partial \hat{\boldsymbol{\lambda}}_c}\frac{\partial \hat{\boldsymbol{\lambda}}(\hat{\boldsymbol{u}}_{j}^*)}{\partial \hat{\boldsymbol{u}}}, \quad \partial_{\hat{\boldsymbol{u}}_r}\tilde{\boldsymbol{c}}_j = \frac{\partial \boldsymbol{c}_j}{\partial \hat{\boldsymbol{\lambda}}_r}\frac{\partial \hat{\boldsymbol{\lambda}}(\hat{\boldsymbol{u}}_{j+1}^*)}{\partial \hat{\boldsymbol{u}}}. \tag{23}$$

We first wish to understand the structure of the terms $\partial_{\hat{\boldsymbol{u}}}\hat{\boldsymbol{\lambda}}(\hat{\boldsymbol{u}})$. For this, we note that the exact dual variables fulfill

$$\hat{\boldsymbol{u}} = \langle \boldsymbol{u}_s(\hat{\boldsymbol{\lambda}}^T\boldsymbol{\varphi})\boldsymbol{\varphi}^T\rangle =: \boldsymbol{h}(\hat{\boldsymbol{\lambda}}), \tag{24}$$

which is why we have the mapping $\hat{\boldsymbol{u}} : \mathbb{R}^{N+1} \to \mathbb{R}^{N+1}$, $\hat{\boldsymbol{u}}(\hat{\boldsymbol{\lambda}}) = \boldsymbol{h}(\hat{\boldsymbol{\lambda}})$. Since the solution of the dual problem for a given moment vector is unique, this mapping is bijective and therefore we have an inverse function

$$\hat{\boldsymbol{\lambda}} = \boldsymbol{h}^{-1}(\hat{\boldsymbol{u}}(\hat{\boldsymbol{\lambda}})). \tag{25}$$

Now we differentiate both sides w.r.t. $\hat{\boldsymbol{\lambda}}$ to get

$$\boldsymbol{I}_d = \frac{\partial \boldsymbol{h}^{-1}(\hat{\boldsymbol{u}}(\hat{\boldsymbol{\lambda}}))}{\partial \hat{\boldsymbol{u}}}\frac{\partial \hat{\boldsymbol{u}}(\hat{\boldsymbol{\lambda}})}{\partial \hat{\boldsymbol{\lambda}}}.$$

We multiply with the matrix inverse of $\frac{\partial \hat{\boldsymbol{u}}(\hat{\boldsymbol{\lambda}})}{\partial \hat{\boldsymbol{\lambda}}}$ to get

$$\left(\frac{\partial \hat{\boldsymbol{u}}(\hat{\boldsymbol{\lambda}})}{\partial \hat{\boldsymbol{\lambda}}}\right)^{-1} = \frac{\partial \boldsymbol{h}^{-1}(\hat{\boldsymbol{u}}(\hat{\boldsymbol{\lambda}}))}{\partial \hat{\boldsymbol{u}}}.$$

Note that on the left-hand-side we have the inverse of a matrix and on the right-hand-side, we have the inverse of a multi-dimensional function. By rewriting $\boldsymbol{h}^{-1}(\hat{\boldsymbol{u}}(\hat{\boldsymbol{\lambda}}))$ as $\hat{\boldsymbol{\lambda}}(\hat{\boldsymbol{u}})$ and simply computing the term $\frac{\partial \hat{\boldsymbol{u}}(\hat{\boldsymbol{\lambda}})}{\partial \hat{\boldsymbol{\lambda}}}$ by differentiating (24) w.r.t. $\hat{\boldsymbol{\lambda}}$, one obtains

$$\partial_{\hat{\boldsymbol{u}}} \hat{\boldsymbol{\lambda}}(\hat{\boldsymbol{u}}) = \langle \nabla \boldsymbol{u}_s(\hat{\boldsymbol{\lambda}}^T \varphi) \varphi \varphi^T \rangle^{-T}. \tag{26}$$

Now we begin to derive the spectrum of the *One-Shot IPM* iteration (21). Note that in its current form this iteration is not really a fix point iteration, since it uses the time updated dual variables in (21b). To obtain a fix point iteration, we plug the dual iteration step (21a) into the moment iteration (21b) to obtain

$$\begin{aligned} \boldsymbol{\lambda}_j^{n+1} &= \boldsymbol{d}(\boldsymbol{\lambda}_j^n, \hat{\boldsymbol{u}}_j^n) \quad \text{for all j} \\ \hat{\boldsymbol{u}}_j^{n+1} &= \boldsymbol{c}\left(\boldsymbol{d}(\boldsymbol{\lambda}_{j-1}^n, \hat{\boldsymbol{u}}_{j-1}^n), \boldsymbol{d}(\boldsymbol{\lambda}_j^n, \hat{\boldsymbol{u}}_j^n), \boldsymbol{d}(\boldsymbol{\lambda}_{j+1}^n, \hat{\boldsymbol{u}}_{j+1}^n)\right) \end{aligned}$$

The Jacobian $\boldsymbol{J} \in \mathbb{R}^{2N \cdot N_x \times 2N \cdot N_x}$ has the form

$$\boldsymbol{J} = \begin{pmatrix} \partial_{\boldsymbol{\lambda}} \boldsymbol{d} & \partial_{\hat{\boldsymbol{u}}} \boldsymbol{d} \\ \partial_{\boldsymbol{\lambda}} \boldsymbol{c} & \partial_{\hat{\boldsymbol{u}}} \boldsymbol{c} \end{pmatrix}, \tag{27}$$

where each block has entries for all spatial cells. We start by looking at $\partial_{\boldsymbol{\lambda}} \boldsymbol{d}$. For the columns belonging to cell j, we have

$$\begin{aligned} \partial_{\boldsymbol{\lambda}} \boldsymbol{d}(\boldsymbol{\lambda}_j^n, \hat{\boldsymbol{u}}_j^n) &= \boldsymbol{I}_d - \boldsymbol{B} \cdot \langle \nabla \boldsymbol{u}_s(\varphi^T \boldsymbol{\lambda}_j^n) \varphi \varphi^T \rangle^T - \partial_{\boldsymbol{\lambda}} \boldsymbol{B} \cdot \left(\langle \boldsymbol{u}_s(\varphi^T \boldsymbol{\lambda}_j^n) \varphi^T \rangle^T - \hat{\boldsymbol{u}} \right) \\ &= -\partial_{\boldsymbol{\lambda}} \boldsymbol{B} \cdot \left(\langle \boldsymbol{u}_s(\varphi^T \boldsymbol{\lambda}_j^n) \varphi^T \rangle^T - \hat{\boldsymbol{u}} \right), \end{aligned}$$

where we used $\boldsymbol{B} = \langle \nabla \boldsymbol{u}_s(\varphi^T \boldsymbol{\lambda}_j^n) \varphi \varphi^T \rangle^{-T}$. Recall that at the fix point $(\boldsymbol{\lambda}^*, \hat{\boldsymbol{u}}^*)$, we have $\langle \boldsymbol{u}_s(\varphi^T \boldsymbol{\lambda}_j^n) \varphi^T \rangle^T = \hat{\boldsymbol{u}}$, hence one obtains $\partial_{\boldsymbol{\lambda}} \boldsymbol{d} = \boldsymbol{0}$. For the block $\partial_{\hat{\boldsymbol{u}}} \boldsymbol{d}$, we get

$$\partial_{\hat{\boldsymbol{u}}} \boldsymbol{d}(\boldsymbol{\lambda}_j^n, \hat{\boldsymbol{u}}_j^n) = \boldsymbol{B},$$

hence $\partial_{\hat{\boldsymbol{u}}} \boldsymbol{d}$ is a block diagonal matrix. Let us now look at $\partial_{\boldsymbol{\lambda}} \boldsymbol{c}$ at a fixed spatial cell j:

$$\frac{\partial \boldsymbol{c}}{\partial \boldsymbol{\lambda}_\ell} \frac{\partial \boldsymbol{d}(\boldsymbol{\lambda}_{j-1}^n, \hat{\boldsymbol{u}}_{j-1}^n)}{\partial \boldsymbol{\lambda}} = \boldsymbol{0},$$

since we already showed that by the choice of $\boldsymbol{B}$ the term $\partial_{\boldsymbol{\lambda}} \boldsymbol{d}$ is zero. We can show the same result for all spatial cells and all inputs of $\boldsymbol{c}$ analogously, hence $\partial_{\boldsymbol{\lambda}} \boldsymbol{c} = \boldsymbol{0}$. For the last block, we have that

$$\frac{\partial \boldsymbol{c}}{\partial \boldsymbol{\lambda}_\ell} \frac{\partial \boldsymbol{d}(\boldsymbol{\lambda}_{j-1}^n, \hat{\boldsymbol{u}}_{j-1}^n)}{\partial \hat{\boldsymbol{u}}} = \frac{\partial \boldsymbol{c}}{\partial \boldsymbol{\lambda}_\ell} \boldsymbol{B} = \frac{\partial \boldsymbol{c}}{\partial \boldsymbol{\lambda}_\ell} \langle \nabla \boldsymbol{u}_s(\varphi^T \boldsymbol{\lambda}_{j-1}^n) \varphi \varphi^T \rangle^{-T} = \partial_{\hat{\boldsymbol{u}}_\ell} \tilde{\boldsymbol{c}}_j$$

by the choice of $\boldsymbol{B}$ as well as (23) and (26). We obtain an analogous result for the second and third input. Hence, we have that $\partial_{\hat{\boldsymbol{u}}} \boldsymbol{c} = \nabla_{\hat{\boldsymbol{u}}} \tilde{\boldsymbol{c}}$, which only has eigenvalues between -1 and 1 by the assumption that the classical IPM iteration is contractive. Since $\boldsymbol{J}$ is an upper triangluar block matrix, the eigenvalues are given by $\lambda\left(\partial_{\boldsymbol{\lambda}} \boldsymbol{d}\right) = 0$ and $\lambda\left(\partial_{\hat{\boldsymbol{u}}} \boldsymbol{c}\right) \in (-1, 1)$, hence the One-Shot IPM is contractive around its fix point. □

Theorem 2. *With the assumptions from Theorem 1, the One-Shot IPM converges locally, i.e. there exists a $\delta > 0$ s.t. for all starting points $(\boldsymbol{\lambda}^0, \hat{\boldsymbol{u}}^0) \in B_\delta(\boldsymbol{\lambda}^*, \hat{\boldsymbol{u}}^*)$ we have*

$$\|(\boldsymbol{\lambda}^n, \hat{\boldsymbol{u}}^n) - (\boldsymbol{\lambda}^*, \hat{\boldsymbol{u}}^*)\| \to 0 \qquad \textit{for } n \to \infty.$$

Proof. By Theorem 1, the One-Shot scheme is contractive at its fix point. Since we assumed convergence of the classical IPM scheme, we can conclude that all entries in the Jacobian $\boldsymbol{J}$ are continuous functions. Furthermore, the determinant of $\tilde{\boldsymbol{J}} := \boldsymbol{J} - \lambda \boldsymbol{I}_d$ is a polynomial of continuous functions, since

$$\det(\tilde{\boldsymbol{J}}) = \sum_\sigma \operatorname{sgn}(\sigma) \prod_{i=1}^{2N_x N} \tilde{J}_{\sigma(i),i}.$$

Since the roots of a polynomial vary continuously with its coefficients, the eigenvalues of $\boldsymbol{J}$ are continuous w.r.t $(\boldsymbol{\lambda}, \hat{\boldsymbol{u}})$. Hence there exists an open ball with radius δ around the fix point in which the eigenvalues remain in the interval $(-1, 1)$. $\square$

Remark 3. *Since the preconditioning step of the Collocation-accelerated IPM method generates initial conditions which are close to the steady state solution, using One-Shot IPM instead of classical IPM is well suited. However, our numerical calculations show that one-shot IPM converges even if the solution is far away from its steady state.*

4. Adaptivity

The following section presents the adaptivity strategy used in this work. Since stochastic hyperbolic problems generally experience shocks in a small portion of the space-time domain, the idea is to perform arising computations on a high accuracy level in this small area, while keeping a low level of accuracy in the remainder. The hope is to achieve the finest level accuracy with this adaptive strategy, i.e. the same error is obtained while using a significantly reduced number of unknowns.

In the following, we discuss the building blocks of the IPM method for accuracy levels $\ell = 1, \cdots, N_{\text{ad}}$. At a given level ℓ, the total degree of the basis function is given by M_ℓ with a corresponding number of moments N_ℓ. The number of quadrature points at level ℓ is denoted by Q_ℓ. To determine the accuracy level of a given moment vector $\hat{\boldsymbol{u}}$ we choose techniques used in discontinuous Galerkin (DG) methods. Adaptivity is a common strategy to accelerate this class of methods and several indicators to determine the smoothness of the solution exist. We make use of a strategy from [**?**], which uses the highest degree moments as indicator. Translating the idea of the discontinuity sensor used in [**?**] to uncertainty quantification, we define the polynomial approximation at level ℓ as

$$\tilde{\boldsymbol{u}}_\ell := \sum_{|i| \leq M_\ell} \hat{\boldsymbol{u}}_i \varphi_i.$$

Now the indicator for a moment vector at level ℓ is defined as

$$\boldsymbol{S}_\ell := \frac{\langle (\tilde{\boldsymbol{u}}_\ell - \tilde{\boldsymbol{u}}_{\ell-1})^2 \rangle}{\langle \tilde{\boldsymbol{u}}_\ell^2 \rangle}, \tag{28}$$

where divisions and multiplications are performed element-wise. In this work, we use the first entry in $\boldsymbol{S}_\ell$ to determine the refinement level, i.e. in the case of gas dynamics, the regularity of the density is chosen to indicate an adequate refinement level. If the moment vector in a given cell at a certain timestep is initially at refinement level ℓ, this level is kept if the error indicator (28) lies in the interval $I_\delta := [\delta_-, \delta_+]$. Here $\delta_\pm$ are user determined parameters. If the indicator is smaller than δ_-, the refinement level is decreased, if it lies above δ_+, it is increased.

Now we need to specify how the different building blocks of the IPM method can be modified to work with varying truncation orders in different cells. Let us first add dimensions to the notation of the dual iteration function $\boldsymbol{d}$, which has been defined in (10). Now, we have $\boldsymbol{d}_\ell : \mathbb{R}^{N_\ell \times p} \times \mathbb{R}^{N_\ell \times p} \to \mathbb{R}^{N_\ell \times p}$, given by

$$\boldsymbol{d}_\ell(\boldsymbol{\lambda}, \hat{\boldsymbol{u}}) := \boldsymbol{\lambda} - \boldsymbol{B}_\ell(\boldsymbol{\lambda}) \cdot \left(\langle \boldsymbol{u}_s(\boldsymbol{\lambda}^T \boldsymbol{\varphi}_\ell) \boldsymbol{\varphi}_\ell^T \rangle_{Q_\ell}^T - \hat{\boldsymbol{u}} \right), \tag{29}$$

where $\boldsymbol{\varphi}_\ell \in \mathbb{R}^{N_\ell}$ collects all basis functions with total degree smaller or equal to M_ℓ. The preconditioner $\boldsymbol{B}_\ell$ is given by

$$\boldsymbol{B}_\ell(\boldsymbol{\lambda}) := \langle \nabla \boldsymbol{u_s}(\boldsymbol{\lambda}^T \boldsymbol{\varphi}_\ell) \boldsymbol{\varphi}_\ell \boldsymbol{\varphi}_\ell^T \rangle_{Q_\ell}^{-T}.$$

An adaptive version of the moment iteration (13) is denoted by $\boldsymbol{c}_\ell^{\boldsymbol{\ell}'} : \mathbb{R}^{N_{\ell_1'} \times p} \times \mathbb{R}^{N_{\ell_2'} \times p} \times \mathbb{R}^{N_{\ell_3'} \times p} \to \mathbb{R}^{N_\ell \times p}$ and given by

$$\boldsymbol{c}_\ell^{\boldsymbol{\ell}'}(\boldsymbol{\lambda}_1, \boldsymbol{\lambda}_2, \boldsymbol{\lambda}_3) := \langle \boldsymbol{u}_s(\boldsymbol{\lambda}_2^T \boldsymbol{\varphi}_{\ell_2'}) \boldsymbol{\varphi}_\ell^T \rangle_{Q_\ell}^T - \frac{\Delta t}{\Delta x} \left(\langle \boldsymbol{g}(\boldsymbol{u}_s(\boldsymbol{\lambda}_2^T \boldsymbol{\varphi}_{\ell_2'}), \boldsymbol{u}_s(\boldsymbol{\lambda}_3^T \boldsymbol{\varphi}_{\ell_3'})) \boldsymbol{\varphi}_\ell^T \rangle_{Q_\ell}^T - \langle \boldsymbol{g}(\boldsymbol{u}_s(\boldsymbol{\lambda}_1^T \boldsymbol{\varphi}_{\ell_1'}), \boldsymbol{u}_s(\boldsymbol{\lambda}_2^T \boldsymbol{\varphi}_{\ell_2'})) \boldsymbol{\varphi}_\ell^T \rangle_{Q_\ell}^T \right).$$

Hence, the index vector $\boldsymbol{\ell}' \in \mathbb{N}^3$ denotes the refinement levels of the stencil cells, which are used to compute the time updated moment vector at level ℓ.

The strategy now is to perform the dual update for a set of moment vectors $\hat{\boldsymbol{u}}_j^n$ at refinement levels ℓ_j^n for $j = 1, \cdots, N_x$. Hence, the dual iteration makes use of the iteration function (29) at refinement level ℓ_j^n. After that, the refinement level at the next time step ℓ_j^{n+1} is determined by making use of the smoothness indicator (28). The moment update then computes the moments at the time updated refinement level ℓ_j^{n+1} making use of the dual states at the old refinement levels $\boldsymbol{\ell}' = (\ell_{j-1}^n, \ell_j^n, \ell_{j+1}^n)^T$. The IPM algorithm with adaptivity then reads

Algorithm 3 Adaptive IPM implementation

```
 1: for $j = 0$ to $N_x + 1$ do
 2:     $\ell_j^0 \leftarrow$ choose initial refinement level
 3:     $\boldsymbol{u}_j^0 \leftarrow \frac{1}{\Delta x}\int_{x_{j-1/2}}^{x_{j+1/2}} \langle u_{\text{IC}}(x,\cdot)\boldsymbol{\varphi}_{\ell_j^0}\rangle_{Q_{\ell_j^0}} dx$
 4: for $n = 0$ to $N_t$ do
 5:     for $j = 0$ to $N_x + 1$ do
 6:         $\boldsymbol{\lambda}_j^{(0)} \leftarrow \hat{\boldsymbol{\lambda}}_j^n$
 7:         while (12) is violated do
 8:             $\boldsymbol{\lambda}_j^{(m+1)} \leftarrow \boldsymbol{d}_{\ell_j^n}(\boldsymbol{\lambda}_j^{(m)}; \hat{\boldsymbol{u}}_j^n)$
 9:             $m \leftarrow m + 1$
10:         $\hat{\boldsymbol{\lambda}}_j^{n+1} \leftarrow \boldsymbol{\lambda}_j^{(m)}$
11:         $\ell_j^{n+1} \leftarrow \text{DetermineRefinementLevel}\left(\hat{\boldsymbol{\lambda}}_j^{n+1}\right)$
12:     for $j = 1$ to $N_x$ do
13:         $\boldsymbol{\ell}' \leftarrow (\ell_{j-1}^n, \ell_j^n, \ell_{j+1}^n)^T$
14:         $\hat{\boldsymbol{u}}_j^{n+1} \leftarrow \boldsymbol{c}_{\ell_j^{n+1}}^{\boldsymbol{\ell}'}(\hat{\boldsymbol{\lambda}}_{j-1}^{n+1}, \hat{\boldsymbol{\lambda}}_j^{n+1}, \hat{\boldsymbol{\lambda}}_{j+1}^{n+1})$
```

Adaptivity can be used for intrusive methods in general as well as for steady and unsteady problems. In the case of steady problems, we can make use of a strategy, which we call *refinement retardation.* Recall that the convergence to an admissible steady state solution is expensive and a high accuracy and desirable solution properties are only required at the end of this iteration process. Hence, we propose to iteratively increase the maximal refinement level whenever a certain residual (20) is fulfilled. For a given set of maximal refinement levels ℓ_m^* and a set of residuals ε_m^* at which the refinement level must be increased we can now perform a large amount of th required iterations on a lower but cheaper accuracy level. The same strategy can be applied for one-shot IPM. In this case, the algorithm is given by

Algorithm 4 Adaptive one-shot IPM implementation with refinement retardation

```
1: for $j = 0$ to $N_x + 1$ do
2:     $\boldsymbol{u}_j^0 \leftarrow \frac{1}{\Delta x} \int_{x_{j-1/2}}^{x_{j+1/2}} \langle u_{\mathrm{IC}}(x, \cdot)\boldsymbol{\varphi} \rangle_Q dx$
3: while (20) is violated do
4:     for $j = 1$ to $N_x$ do
5:         $\boldsymbol{\lambda}_j^{n+1} \leftarrow \boldsymbol{d}_{\ell_j^n}(\boldsymbol{\lambda}_j^n; \hat{\boldsymbol{u}}_j^n)$
6:         $\ell_j^{n+1} \leftarrow \max\{\text{DetermineRefinementLevel}\left(\boldsymbol{\lambda}_j^{n+1}\right), \ell_i^*\}$
7:         $\boldsymbol{\ell}' \leftarrow (\ell_{j-1}^n, \ell_j^n, \ell_{j+1}^n)^T$
8:         $\hat{\boldsymbol{u}}_j^{n+1} \leftarrow \boldsymbol{c}_{\ell_j^{n+1}}^{\boldsymbol{\ell}'}(\boldsymbol{\lambda}_{j-1}^{n+1}, \boldsymbol{\lambda}_j^{n+1}, \boldsymbol{\lambda}_{j+1}^{n+1})$
9:     $n \leftarrow n + 1$
10:    if (20) fulfills the stopping criterion $\varepsilon_m^*$ then
11:        $m \leftarrow m + 1$
```

5. Results

5.1. 2D Euler equations

We start by quantifying the effects of an uncertain angle of attack $\phi \sim U(0.75, 1.75)$ for a NACA0012 profile computed with different methods. The stochastic Euler equations in two dimensions are given by

$$\partial_t \begin{pmatrix} \rho \\ \rho v_1 \\ \rho v_2 \\ \rho e \end{pmatrix} + \partial_{x_1} \begin{pmatrix} \rho v_1 \\ \rho v_1^2 + p \\ \rho v_1 v_2 \\ v_1(\rho e + p) \end{pmatrix} + \partial_{x_2} \begin{pmatrix} \rho v_2 \\ \rho v_1 v_2 \\ \rho v_2^2 + p \\ v_2(\rho e + p) \end{pmatrix} = \mathbf{0}.$$

These equations determine the time evolution of the conserved variables $(\rho, rho\boldsymbol{v}, rhoe)$, i.e. density, momentum and energy. A closure for the pressure p is given by

$$p = (\gamma - 1)\rho \left(e - \frac{1}{2}(v_1^2 + v_2^2) \right).$$

Since the the fluid of the following test cases is air, we choose the heat capacity ratio γ to be 1.4. The spatial mesh discretizes the flow domain around the airfoil. At the airfoil boundary Γ_0, we use the Euler slip condition $\boldsymbol{v}^T \boldsymbol{n} = 0$, where $\boldsymbol{n}$ denotes the surface normal. At a sufficiently large distance away from the airfoil, we assume a far field flow with a given Mach number $Ma = 0.8$, pressure $p = 101,325$ Pa and a temperature of 273.15 K. Now the angle of attack ϕ is uniformly distributed in the interval of $[0.75, 1.75]$ degrees. I.e. we choose $\phi(\xi) = 1.25 + 0.5\xi$ where $\xi \sim U(-1, 1)$. The aim is to quantify the effects arising from the one-dimensional uncertainty ξ with different methods. The IPM methods makes use of the acceleration strategies proposed in this work. To be able to measure the quality of the obtained solutions, we compute a reference solution using stochastic-Collocation with 100 Gauss-Lobatto quadrature points.

In the following, we compare stochastic-Collocation with stochastic-Galerkin and IPM as well as its proposed acceleration techniques. Note that since IPM generalizes SG, all proposed techniques can be used for this method as well. For more information on the chosen entropy and the resulting solution ansatz for IPM, see Appendix A.

and compare against Collocation with 5 collocation points as well as SG and IPM with 5 moments and 10 quadrature points. Furthermore, we use convergence accelerated (caIPM) as well as convergence accelerated one-shot IPM (caosIPM), which we introduced in Sections 3.1 and 3.2 with 10 moments and 15 quadrature points to converge the collocation solution with 5 quadrature points to an entropy solution with increased accuracy.

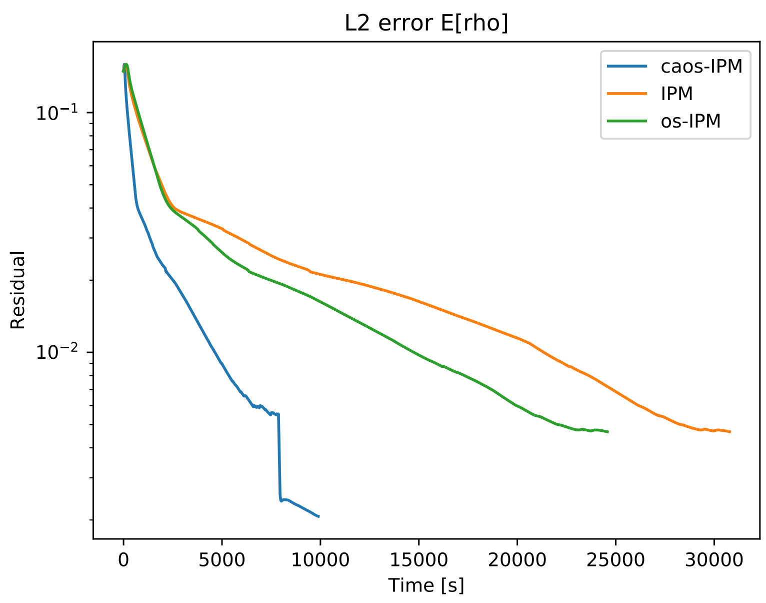

Figure 1: L2 error at airfoil with 5 MPI and 2 OMP threads.

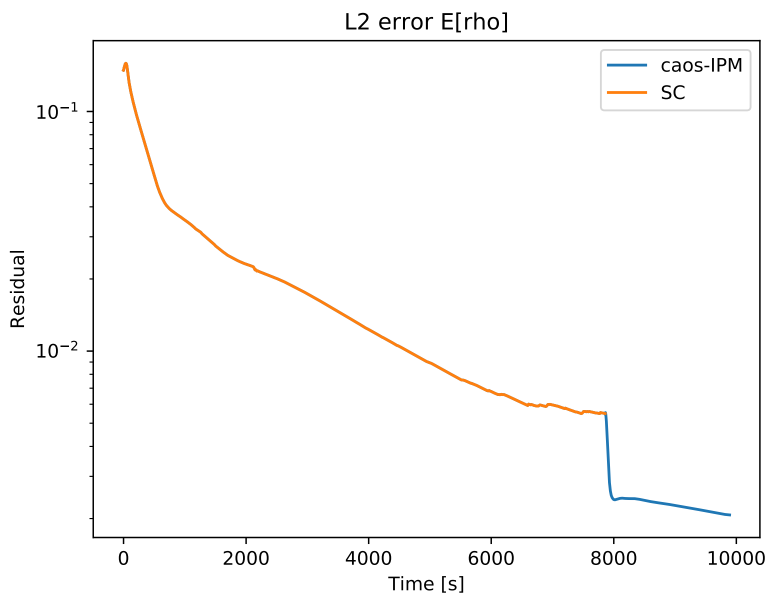

Figure 2: L2 error at airfoil with 5 MPI and 2 OMP threads.

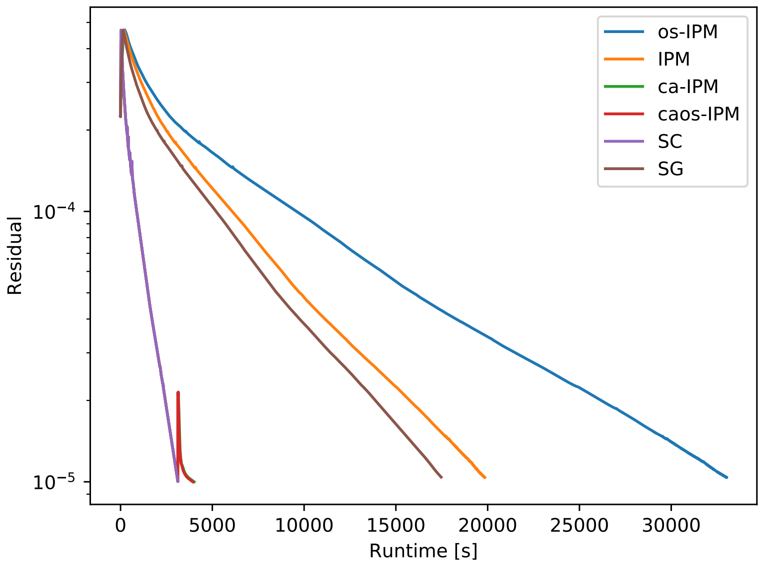

Figure 3: Convergence to steady state with 5 MPI and 2 OMP threads.

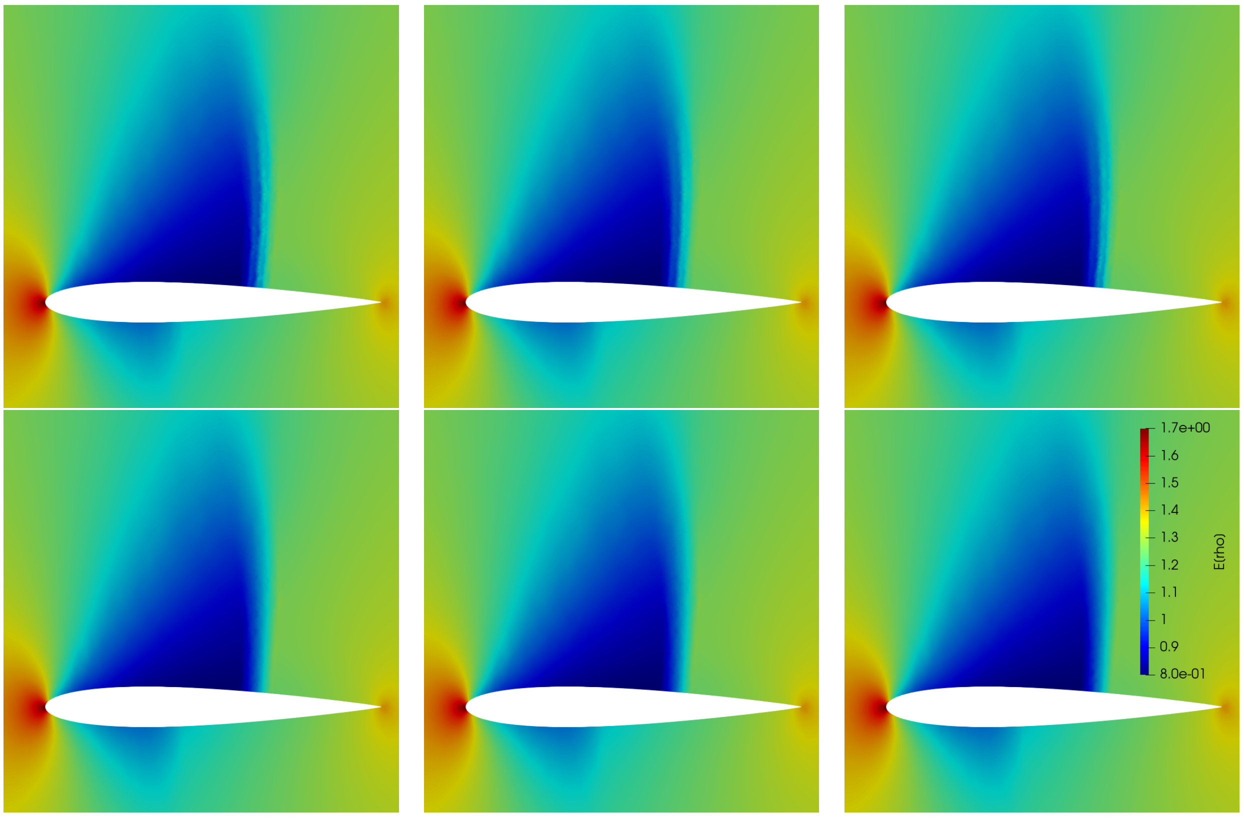

Figure 4: E[ρ] (from top left to bottom right) SC, SG, IPM, caIPM, caosIPM, reference solution.

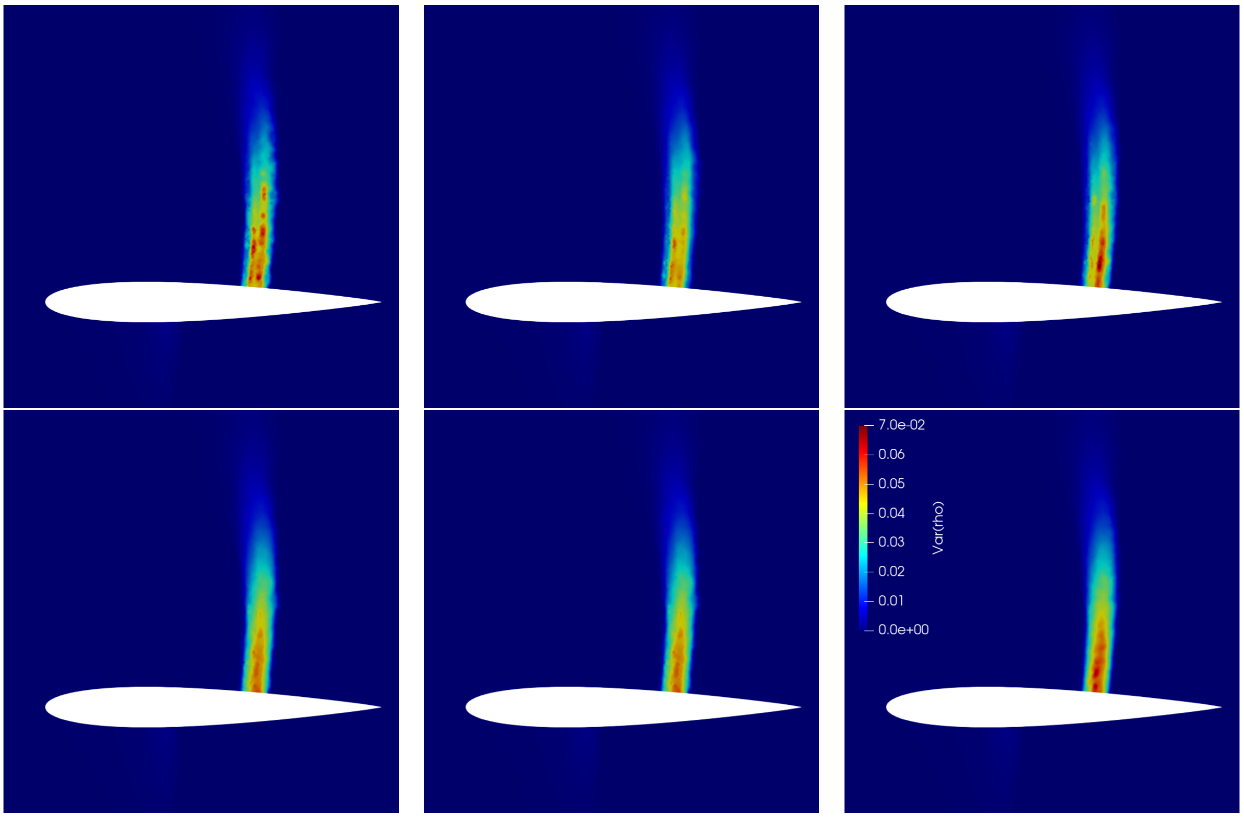

Figure 5: Var[ρ] (from top left to bottom right) SC, SG, IPM, caIPM, caosIPM, reference solution.

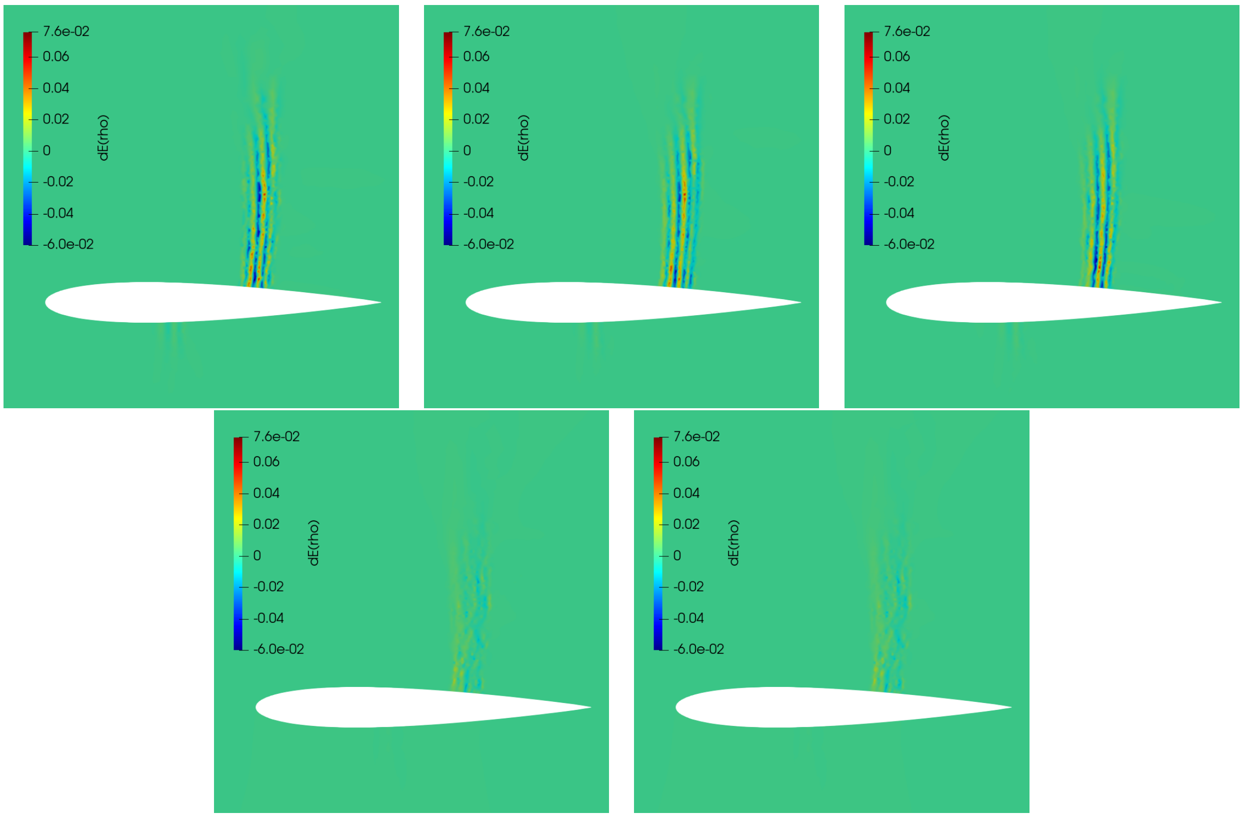

Figure 6: E[ρ] distance to reference solution (from top left to bottom right) SC, SG, IPM, caIPM, caosIPM.

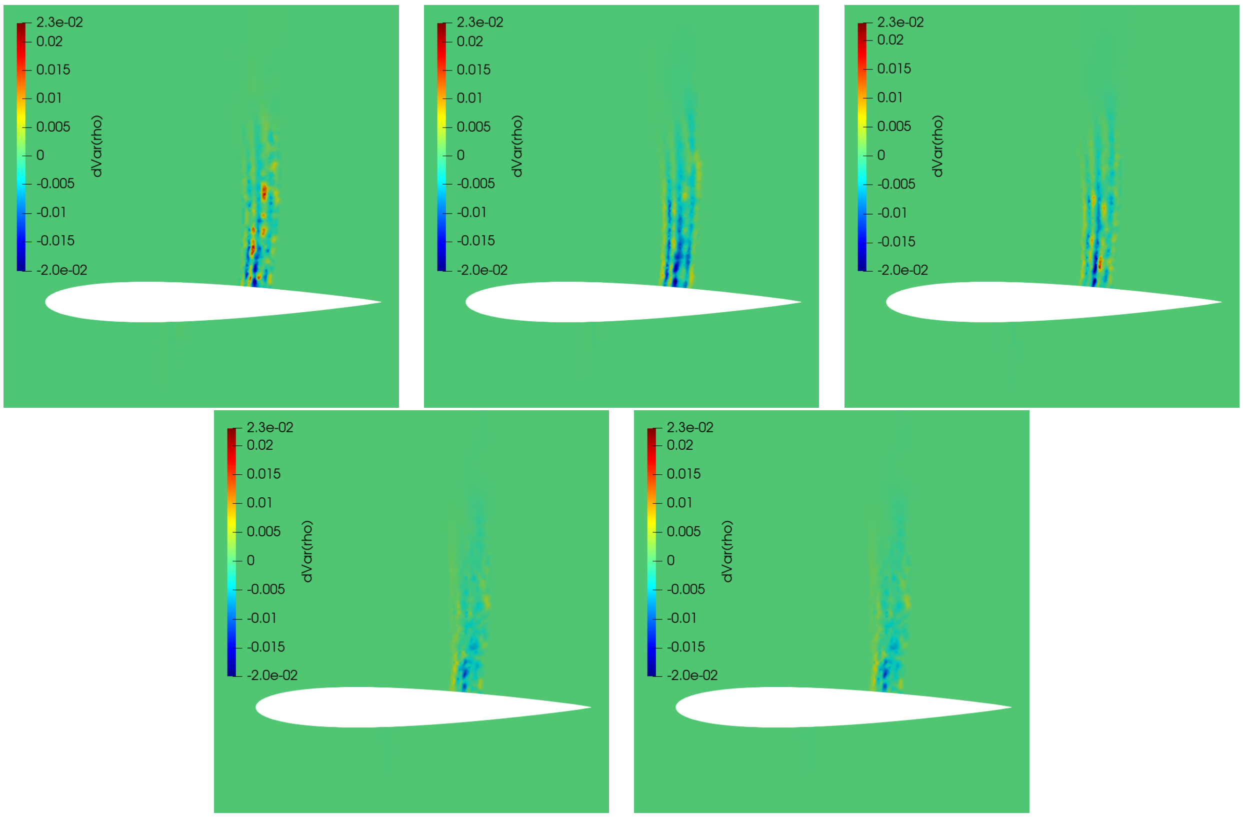

Figure 7: Var[ρ] distance to reference solution (from top left to bottom right) SC, SG, IPM, caIPM, caosIPM.

6. Summary and outlook

Appendix A. IPM for the 2D Euler equations

In the following, we provide details on the implementation of IPM for the 2D Euler equations. We for clarity, we denote the momentum by $m_1 := \rho v_1$ and $m_2 := \rho v_2$ and the energy by $E := \rho e$. Then, the vector of conserved variables is $\boldsymbol{u} = (\rho, m_1, m_2, E)^T$. The entropy used is

$$s(\boldsymbol{u}) = -\rho \ln \left(\rho^{-\gamma} \left(E - \frac{m_1^2 + m_2^2}{2\rho} \right) \right).$$

Now the gradient of the entropy $\nabla_{\boldsymbol{u}} s$ has the components

$$\begin{aligned}
\frac{\partial s}{\partial \rho} &= -\ln \left(\rho^{-\gamma} \left(E - \frac{m_1^2 + m_2^2}{2\rho} \right) \right) + \frac{m_1^2 + m_2^2}{-2\rho E + m_1^2 + m_2^2} + \gamma, \\
\frac{\partial s}{\partial m_i} &= -\frac{2\rho m_i}{-2\rho E + m_1^2 + m_2^2}, \\
\frac{\partial s}{\partial E} &= -\frac{1}{\rho} \left(E - \frac{m_1^2 + m_2^2}{2\rho} \right).
\end{aligned}$$

To compute $\boldsymbol{u}_s(\boldsymbol{\Lambda}) = (\nabla_{\boldsymbol{u}} s)^{-1}(\boldsymbol{\Lambda})$, we set $\boldsymbol{\Lambda} = \nabla_{\boldsymbol{u}} s(\boldsymbol{u})$ and rearrange with respect to $\boldsymbol{u}$. Let us define

$$\alpha(\boldsymbol{\Lambda}) := \exp \left(\frac{\Lambda_2^2 + \Lambda_3^2 - 2\Lambda_1 \Lambda_4 - 2\Lambda_4 \gamma}{2\Lambda_4 (1-\gamma)} \right) \cdot (-\Lambda_4)^{\frac{1}{1-\gamma}}$$

Then the solution ansatz $\boldsymbol{u}_s$ is given by

$$\begin{aligned}
\rho(\boldsymbol{\Lambda}) &= \alpha(\boldsymbol{\Lambda}), \qquad m_1(\boldsymbol{\Lambda}) = -\frac{\Lambda_2 \alpha(\boldsymbol{\Lambda})}{\Lambda_4}, \qquad m_2(\boldsymbol{\Lambda}) = -\frac{\Lambda_3 \alpha(\boldsymbol{\Lambda})}{\Lambda_4}, \\
E(\boldsymbol{\Lambda}) &= -\frac{\alpha(\boldsymbol{\Lambda})(-\Lambda_2^2 - \Lambda_3^2 + 2\Lambda_4)}{2\Lambda_4^2}.
\end{aligned}$$