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 the shallow water equations.

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 $|i| \leq M$. Commonly one chooses orthonormal polynomials as basis functions [1], i.e. $\langle \varphi_n \varphi_m \rangle = \delta_{nm}$. The moments are then given by $\hat{\boldsymbol{u}}_{\boldsymbol{i}} := \langle \boldsymbol{u} \varphi_i \rangle$. Besides facilitating the computation of the expectation value and variance, the moments can be used to span the solution in $\boldsymbol{\xi}$ [2].

Numerical methods for approximating the moment $\hat{\boldsymbol{u}}_{\boldsymbol{i}}$ can be divided into intrusive and non-intrusive methods. A popular non-intrusive method is the stochastic-Collocation (SC) method, see e.g. [3, 4, 5], which computes the moments with the help of a numerical quadrature rule: For a given set of N_q quadrature weights w_k and quadrature points $\boldsymbol{\xi}_k$, the moments are approximated by

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

Since the solution at a fixed quadrature point can be computed by a standard deterministic solver, the SC method does not require a big 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 [6, 7]. Despite their easy implementation, collocation methods can become cumbersome for 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

$$\boldsymbol{u}(t, \boldsymbol{x}, \boldsymbol{\xi}) \approx \mathcal{U}(\hat{\boldsymbol{u}}_0, \cdots, \hat{\boldsymbol{u}}_N; \boldsymbol{\xi}).$$

A commonly used closure is given by stochastic-Galerkin (SG) [8], which represents the solution by a polynomial:

$$\mathcal{U}_{\mathrm{SG}}(\hat{\boldsymbol{u}}_0, \cdots, \hat{\boldsymbol{u}}_N; \boldsymbol{\xi}) := \sum_{i=0}^{N} \hat{\boldsymbol{u}}_i \varphi_i(\boldsymbol{\xi}) = \hat{\boldsymbol{u}}^T \boldsymbol{\varphi}(\boldsymbol{\xi}),$$

Here, we collect the moments for which $|i| \leq M$ holds in the moment matrix $\hat{\boldsymbol{u}} := (\hat{\boldsymbol{u}}_0, \cdots, \hat{\boldsymbol{u}}_N)^T \in \mathbb{R}^{(N+1) \times p}$ and the corresponding basis functions in $\boldsymbol{\varphi} := (\varphi_0, \cdots, \varphi_N)^T \in \mathbb{R}^{N+1}$. When using the stochastic-Galerkin method to close (2), the resulting moment system is not necessarily hyperbolic [9] and the solution needs to be manipulated [10] in order to prevent a failure of the method. A generalization of stochastic-Galerkin, which ensures hyperbolicity is the Intrusive Polynomial Moment (IPM) closure [9]: 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

$$\mathcal{U}(\hat{\boldsymbol{u}}) = \arg\min_{\boldsymbol{u}} \langle s(\boldsymbol{u}) \rangle \quad \text{subject to } \hat{\boldsymbol{u}}_i = \langle \boldsymbol{u} \varphi_i \rangle \text{ for } i = 0, \cdots, N. \tag{3}$$

Rewritten in its dual form, (3) is transformed into an unconstraint optimization problem, given by

$$\hat{\boldsymbol{\lambda}}(\hat{\boldsymbol{u}}) := \underset{\boldsymbol{\lambda}\in\mathbb{R}^{(N+1)\times p}}{\arg\min} \left\{ \langle s_*(\boldsymbol{\lambda}^T\boldsymbol{\varphi})\rangle - \sum_{i=0}^{N} \boldsymbol{\lambda}_i^T \hat{\boldsymbol{u}}_i \right\}, \tag{4}$$

where $s_* : \mathbb{R}^p \to \mathbb{R}$ is the Legendre transformation of s, and $\hat{\boldsymbol{\lambda}} := (\hat{\boldsymbol{\lambda}}_0, \cdots, \hat{\boldsymbol{\lambda}}_N)^T \in \mathbb{R}^{(N+1)\times p}$ are called the dual variables. The solution to (3) is then obtained by

$$\mathcal{U}(\hat{\boldsymbol{u}}) = (\nabla_{\boldsymbol{u}} s)^{-1} (\hat{\boldsymbol{\lambda}}(\hat{\boldsymbol{u}})^T \boldsymbol{\varphi}). \tag{5}$$

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

$$\partial_t \hat{\boldsymbol{u}}_i(t,\boldsymbol{x}) + \nabla \cdot \langle \boldsymbol{f}(\mathcal{U}(\hat{\boldsymbol{u}}))\varphi_i\rangle = \boldsymbol{0}, \tag{6a}$$

$$\hat{\boldsymbol{u}}_{\boldsymbol{i}}(t=0,\boldsymbol{x}) = \langle \boldsymbol{u}_{\mathrm{IC}}(\boldsymbol{x},\cdot)\varphi_i\rangle. \tag{6b}$$

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 [11]. Also, as discussed in [9], 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 (5), which involves solving the optimization problem (4). Hence, the desirable properties of IPM come along with a significantly increased run time. However, IPM and minimal entropy methods in general are well suited for modern HPC architecture, which can be used to significantly reduce runtime [12].

When studying hyperbolic equations, the moment approximations of various methods such as Stochastic Galerkin [13], IPM [14] and stochastic-Collocation [15, 16] tend to show incorrect discontinuities in certain regions of the physical space. These non-physical structures tend to dissolve when the number of basis functions is increased [17, 18] or when artificial diffusion is added through the spatial numerical method [18] or filters [14]. Here, intrusive methods seem to be an adequate choice since they are well suited for adaptive strategies which locally increase the polynomial order [19, 20, 21] or add artificial viscosity [14] at certain spatial positions and time steps in which complex structures such as discontinuities occur. The main task here is to find an adequate refinement indicator. In this paper we compare methods without using local refinement strategies.

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 to collocation methods. The steady setting provides different opportunities to take advantage of features of intrusive methods:

- Accelerate 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 (4) 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 (4).

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. A comparison of results computed with the presented methods is given in 4, followed by a summary and outlook in section 5.

2. Discretization and code framework

2.1. Discretization

In the following, we discretize the moment system in space and time according to [11]. 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 (6). 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 $\boldsymbol{F} : \mathbb{R}^{(N+1)\times p} \to \mathbb{R}^{(N+1)\times p}$, $\boldsymbol{F}(\hat{\boldsymbol{u}}) = \langle \boldsymbol{f}(\mathcal{U}(\hat{\boldsymbol{u}}))\boldsymbol{\varphi}^T\rangle^T$. 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 $\boldsymbol{G} : \mathbb{R}^{(N+1)\times p} \times \mathbb{R}^{(N+1)\times p} \to \mathbb{R}^{(N+1)\times p}$ as

$$\hat{\boldsymbol{u}}_j^{n+1} = \hat{\boldsymbol{u}}_j^n - \frac{\Delta t}{\Delta x}\left(\boldsymbol{G}(\hat{\boldsymbol{u}}_j^n, \hat{\boldsymbol{u}}_{j+1}^n) - \boldsymbol{G}(\hat{\boldsymbol{u}}_{j-1}^n, \hat{\boldsymbol{u}}_j^n)\right) \tag{7}$$

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., that $\boldsymbol{G}(\hat{\boldsymbol{u}}, \hat{\boldsymbol{u}}) = \boldsymbol{F}(\hat{\boldsymbol{u}})$.

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

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

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

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

meaning that the closure (5) 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 (4) 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

$$L(\boldsymbol{\lambda}; \hat{\boldsymbol{u}}) := \langle s_*(\boldsymbol{\lambda}^T\boldsymbol{\varphi})\rangle - \sum_{i=0}^{N} \boldsymbol{\lambda}_i^T \hat{\boldsymbol{u}}_i$$

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

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

which is usually done by an iterative method. Defining the dual iteration function $\boldsymbol{d} : \mathbb{R}^{(N+1)\times p} \times \mathbb{R}^{(N+1)\times p} \to \mathbb{R}^{(N+1)\times p}$,

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

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

$$\boldsymbol{\lambda}_j^{(m+1)} = \boldsymbol{d}(\boldsymbol{\lambda}_j^{(m)}, \hat{\boldsymbol{u}}_{\boldsymbol{j}}). \tag{8}$$

The exact dual state is then obtained by computing the fix point of $\boldsymbol{d}$, meaning that one converges the iteration (8), 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.

$$\boldsymbol{B}(\boldsymbol{\lambda}) := \langle \nabla \boldsymbol{u}_{\boldsymbol{s}}(\boldsymbol{\lambda}^T \boldsymbol{\varphi}) \boldsymbol{\varphi} \boldsymbol{\varphi}^T \rangle^{-T}.$$

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

$$\sum_{i=0}^{p} \left\| \nabla_{\boldsymbol{\lambda_i}} L(\boldsymbol{\lambda}_j^{(m)}; \hat{\boldsymbol{u}}_j^n) \right\| < \tau \tag{9}$$

is used.

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

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

The moment iteration is then given by

$$\hat{\boldsymbol{u}}_j^{n+1} = \boldsymbol{c}\left(\hat{\boldsymbol{\lambda}}(\hat{\boldsymbol{u}}_{j-1}^n), \hat{\boldsymbol{\lambda}}(\hat{\boldsymbol{u}}_j^n), \hat{\boldsymbol{\lambda}}(\hat{\boldsymbol{u}}_{j+1}^n) \right), \tag{10}$$

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

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

until condition (9) is fulfilled. This gives Algorithm 1.

Algorithm 1 IPM implementation

for $j = 0$ to $N_x + 1$ **do**
  $\boldsymbol{u}_j^0 \leftarrow \frac{1}{\Delta x} \int_{x_{j-1/2}}^{x_{j+1/2}} \langle u_{\mathrm{IC}}(x, \cdot) \boldsymbol{\varphi} \rangle dx$
for $n = 0$ to N_t **do**
  for $j = 0$ to $N_x + 1$ **do**
    $\boldsymbol{\lambda}_j^{(0)} \leftarrow \hat{\boldsymbol{\lambda}}_j^n$
    while (9) is violated **do**
      $\boldsymbol{\lambda}_j^{(m+1)} \leftarrow \boldsymbol{d}(\boldsymbol{\lambda}_j^{(m)}; \hat{\boldsymbol{u}}_j^n)$
      $m \leftarrow m + 1$
    $\hat{\boldsymbol{\lambda}}_j^{n+1} \leftarrow \boldsymbol{\lambda}_j^{(m)}$
  for $j = 1$ to N_x **do**
    $\hat{\boldsymbol{u}}_j^{n+1} \leftarrow \boldsymbol{c}(\hat{\boldsymbol{\lambda}}_{j-1}^{n+1}, \hat{\boldsymbol{\lambda}}_j^{n+1}, \hat{\boldsymbol{\lambda}}_{j+1}^{n+1})$

2.2. Code framework

3. Strategies for steady problems

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

$$\nabla \cdot \boldsymbol{f}(\boldsymbol{u}(\boldsymbol{x}, \boldsymbol{\xi})) = \boldsymbol{0} \quad \text{in } D \tag{12}$$

with adequate boundary conditions. A general strategy for computing the steady state solution to (12) is to introduce a pseudo-time and numerically treat (12) 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 (12) given by

$$\nabla \cdot \langle \boldsymbol{f}(\boldsymbol{u}(\boldsymbol{x}, \boldsymbol{\xi}))\boldsymbol{\varphi}^T \rangle^T = \boldsymbol{0} \quad \text{in } D \tag{13}$$

which is again needed for the construction of intrusive methdos. By adding a pseudo-time and using the IPM closure, we obtain the same system as in (6), 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

$$\sum_{j=1}^{N_x} \Delta x_j \|\hat{\boldsymbol{u}}_j^n - \hat{\boldsymbol{u}}_j^{n-1}\| \leq \varepsilon \tag{14}$$

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 (4) 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 4, 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 (5) 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 (3) 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 (3) simultaneously. Hence, by successively performing one update of the moment iteration and one update of the dual iteration, we obtain

$$\boldsymbol{\lambda}_j^{n+1} = \boldsymbol{d}(\boldsymbol{\lambda}_j^n, \boldsymbol{u}_j^n) \quad \text{for all j} \tag{15a}$$

$$\boldsymbol{u}_j^{n+1} = \boldsymbol{c}\left(\boldsymbol{\lambda}_{j-1}^{n+1}, \boldsymbol{\lambda}_j^{n+1}, \boldsymbol{\lambda}_{j+1}^{n+1}\right). \tag{15b}$$

This gives algorithm

Algorithm 2 One-shot IPM implementation

for $j = 0$ to $N_x + 1$ **do**
  $\boldsymbol{u}_j^0 \leftarrow \frac{1}{\Delta x}\int_{x_{j-1/2}}^{x_{j+1/2}} \langle u_{\mathrm{IC}}(x,\cdot)\boldsymbol{\varphi}\rangle dx$
while (14) is violated **do**
  for $j = 1$ to N_x **do**
    $\boldsymbol{\lambda}_j^{n+1} \leftarrow \boldsymbol{d}(\boldsymbol{\lambda}_j^n; \hat{\boldsymbol{u}}_j^n)$
    $\hat{\boldsymbol{u}}_j^{n+1} \leftarrow \boldsymbol{c}(\boldsymbol{\lambda}_{j-1}^{n+1}, \boldsymbol{\lambda}_j^{n+1}, \boldsymbol{\lambda}_{j+1}^{n+1})$
  $n \leftarrow n+1$

We call this method One-Shot IPM, since it is inspired by One-shot optimization, see for example [22], 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* (15) *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, understand what contraction of the classical IPM iteration implies, we rewrite the moment iteration (10) of the classical IPM scheme: When defining the update function

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

we can rewrite the classical moment iteration as

$$\hat{\boldsymbol{u}}_j^{n+1} = \tilde{\boldsymbol{c}}\left(\hat{\boldsymbol{u}}_{j-1}^n, \hat{\boldsymbol{u}}_j^n, \hat{\boldsymbol{u}}_{j+1}^n\right). \tag{16}$$

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

$\nabla_{\hat{\boldsymbol{u}}}\tilde{\boldsymbol{c}} \in \mathbb{R}^{(N+1)\cdot N_x \times (N+1)\cdot N_x}$ 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

$$\partial_{\hat{\boldsymbol{u}}_\ell}\tilde{\boldsymbol{c}}_j = \frac{\partial \boldsymbol{c}_j}{\partial \hat{\boldsymbol{\lambda}}_\ell}\frac{\partial \hat{\boldsymbol{\lambda}}(\hat{\boldsymbol{u}}^*_{j-1})}{\partial \hat{\boldsymbol{u}}}, \;\; \partial_{\hat{\boldsymbol{u}}_c}\tilde{\boldsymbol{c}}_j = \frac{\partial \boldsymbol{c}_j}{\partial \hat{\boldsymbol{\lambda}}_c}\frac{\partial \hat{\boldsymbol{\lambda}}(\hat{\boldsymbol{u}}^*_j)}{\partial \hat{\boldsymbol{u}}}, \;\; \partial_{\hat{\boldsymbol{u}}_r}\tilde{\boldsymbol{c}}_j = \frac{\partial \boldsymbol{c}_j}{\partial \hat{\boldsymbol{\lambda}}_r}\frac{\partial \hat{\boldsymbol{\lambda}}(\hat{\boldsymbol{u}}^*_{j+1})}{\partial \hat{\boldsymbol{u}}}. \tag{17}$$

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

$$\hat{\boldsymbol{u}} = \langle \boldsymbol{u}_s(\hat{\boldsymbol{\lambda}}^T\boldsymbol{\varphi})\boldsymbol{\varphi}^T\rangle =: \boldsymbol{h}(\hat{\boldsymbol{\lambda}}), \tag{18}$$

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

$$\hat{\boldsymbol{\lambda}} = \boldsymbol{h}^{-1}(\hat{\boldsymbol{u}}(\hat{\boldsymbol{\lambda}})). \tag{19}$$

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 (18) w.r.t. $\hat{\boldsymbol{\lambda}}$, one obtains

$$\partial_{\hat{\boldsymbol{u}}}\hat{\boldsymbol{\lambda}}(\hat{\boldsymbol{u}}) = \langle \nabla \boldsymbol{u}_s(\hat{\boldsymbol{\lambda}}^T\boldsymbol{\varphi})\boldsymbol{\varphi}\boldsymbol{\varphi}^T\rangle^{-T}. \tag{20}$$

Now we begin to derive the spectrum of the *One-Shot IPM* iteration (15). Note that in its current form this iteration is not really a fix point iteration, since it uses the time updated dual variables in (15b). To obtain a fix point iteration, we plug the dual iteration step (15a) into the moment iteration (15b) to obtain

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

The Jacobian $\boldsymbol{J} \in \mathbb{R}^{2(N+1)\cdot N_x \times 2(N+1)\cdot N_x}$ has the form

$$\boldsymbol{J} = \begin{pmatrix} \partial_{\boldsymbol{\lambda}}\boldsymbol{d} & \partial_{\hat{\boldsymbol{u}}}\boldsymbol{d} \\ \partial_{\boldsymbol{\lambda}}\boldsymbol{c} & \partial_{\hat{\boldsymbol{u}}}\boldsymbol{c} \end{pmatrix}, \tag{21}$$

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

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

where we used $\boldsymbol{B} = \langle \nabla \boldsymbol{u}_s(\boldsymbol{\varphi}^T \boldsymbol{\lambda}_j^n) \boldsymbol{\varphi} \boldsymbol{\varphi}^T \rangle^{-T}$. Recall that at the fix point $(\boldsymbol{\lambda}^*, \hat{\boldsymbol{u}}^*)$, we have $\langle \boldsymbol{u}_s(\boldsymbol{\varphi}^T \boldsymbol{\lambda}_j^n) \boldsymbol{\varphi}^T \rangle^T = \hat{\boldsymbol{u}}$, 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

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

by the choice of $\boldsymbol{B}$ as well as (17) and (20). 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 $\lambda(\partial_{\boldsymbol{\lambda}} \boldsymbol{d}) = 0$ and $\lambda(\partial_{\hat{\boldsymbol{u}}} \boldsymbol{c}) \in (-1, 1)$, 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*

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

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

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

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 $(-1, 1)$. □

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. Results

4.1. 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. To be able to measure the quality of the obtained solutions, we compute

a reference solution using stochastic-Collocation with 50 quadrature points 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.

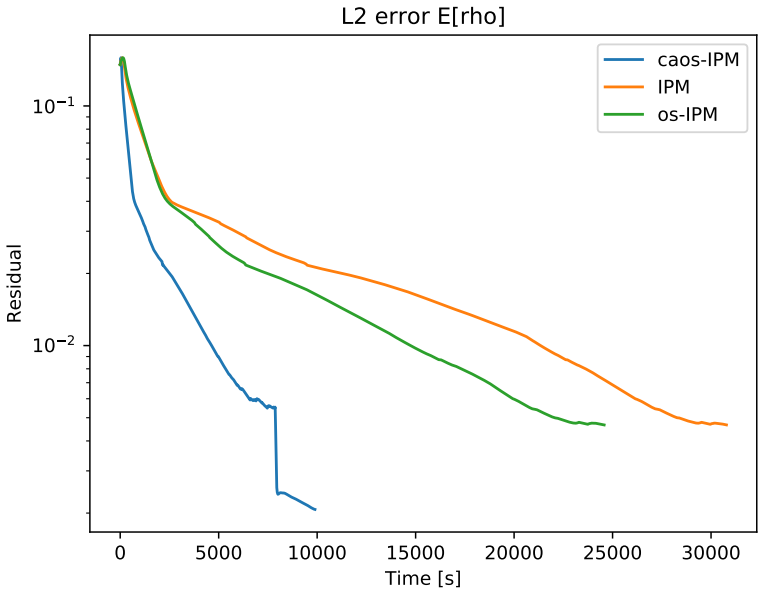

Figure 1: L2 error at airfoil with 5 MPI and 2 OMP threads.

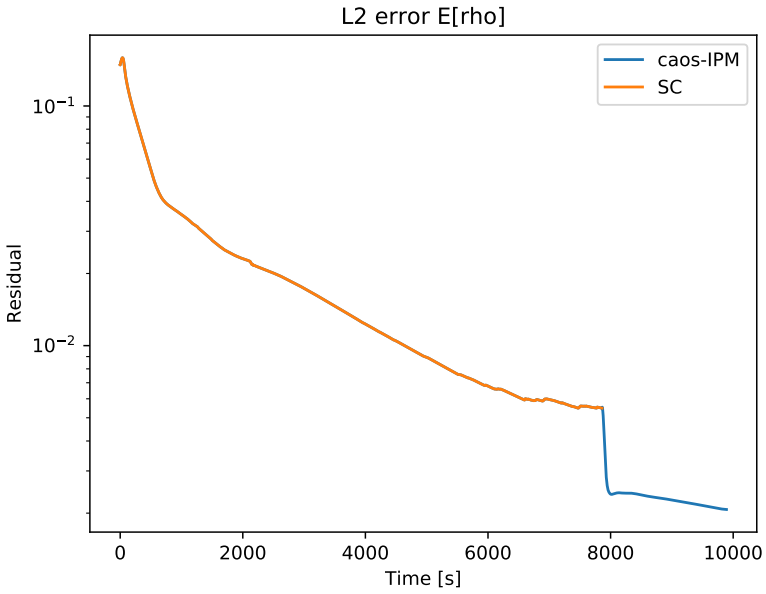

Figure 2: L2 error at airfoil with 5 MPI and 2 OMP threads.

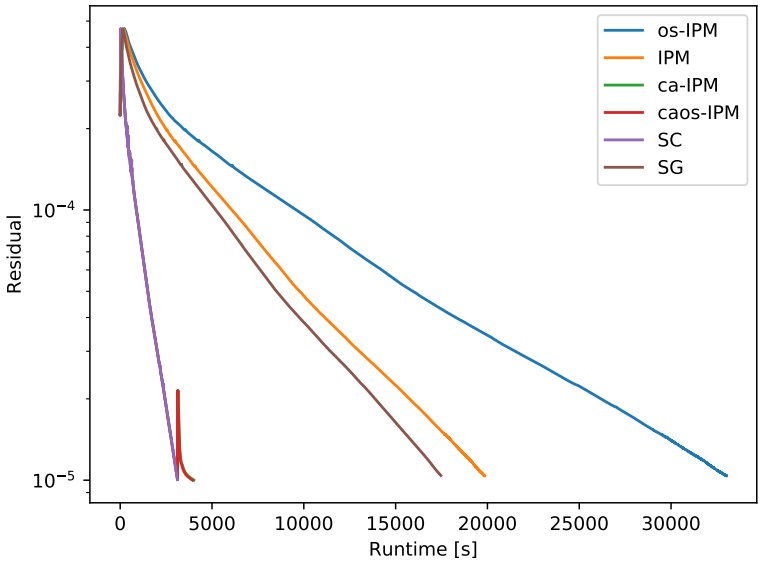

Figure 3: Convergence to steady state with 5 MPI and 2 OMP threads.

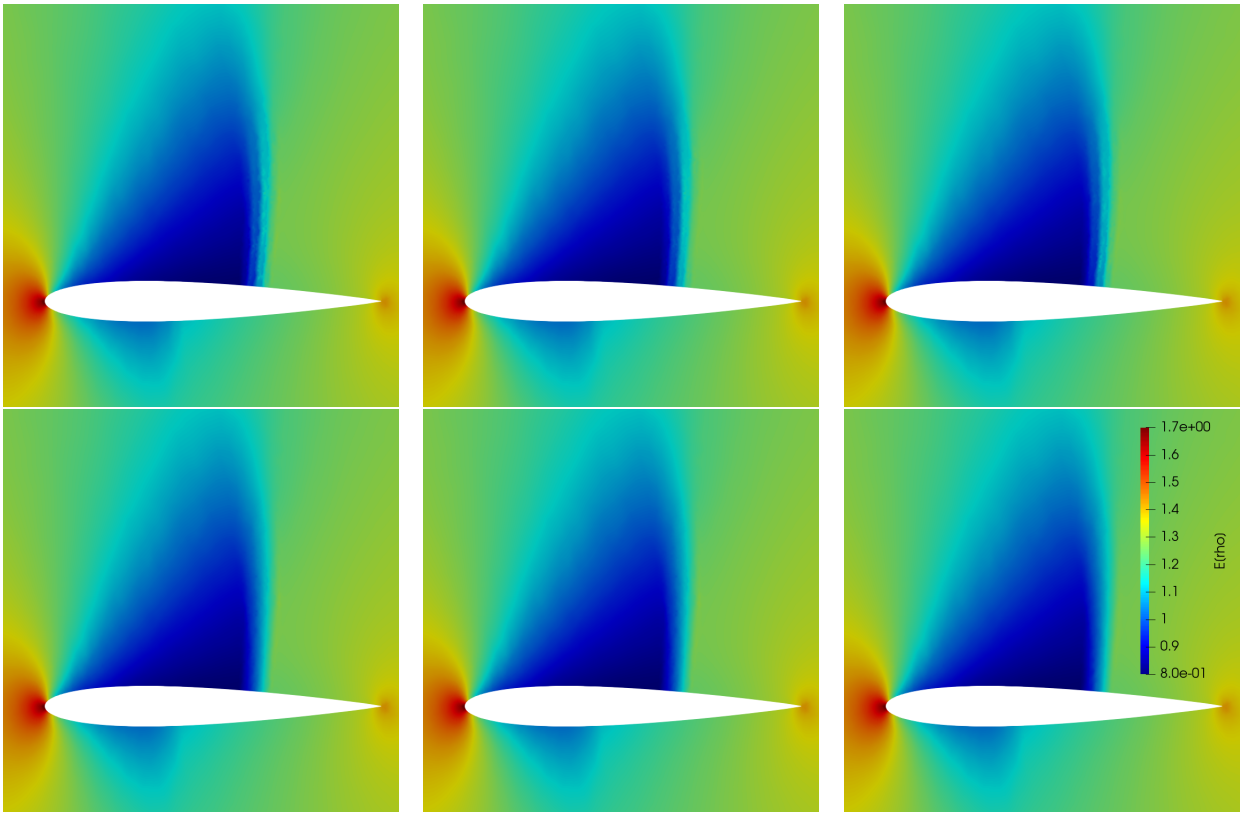

Figure 4: E[ρ] (from top left to bottom right) SC, SG, IPM, caIPM, caosIPM, reference solution.

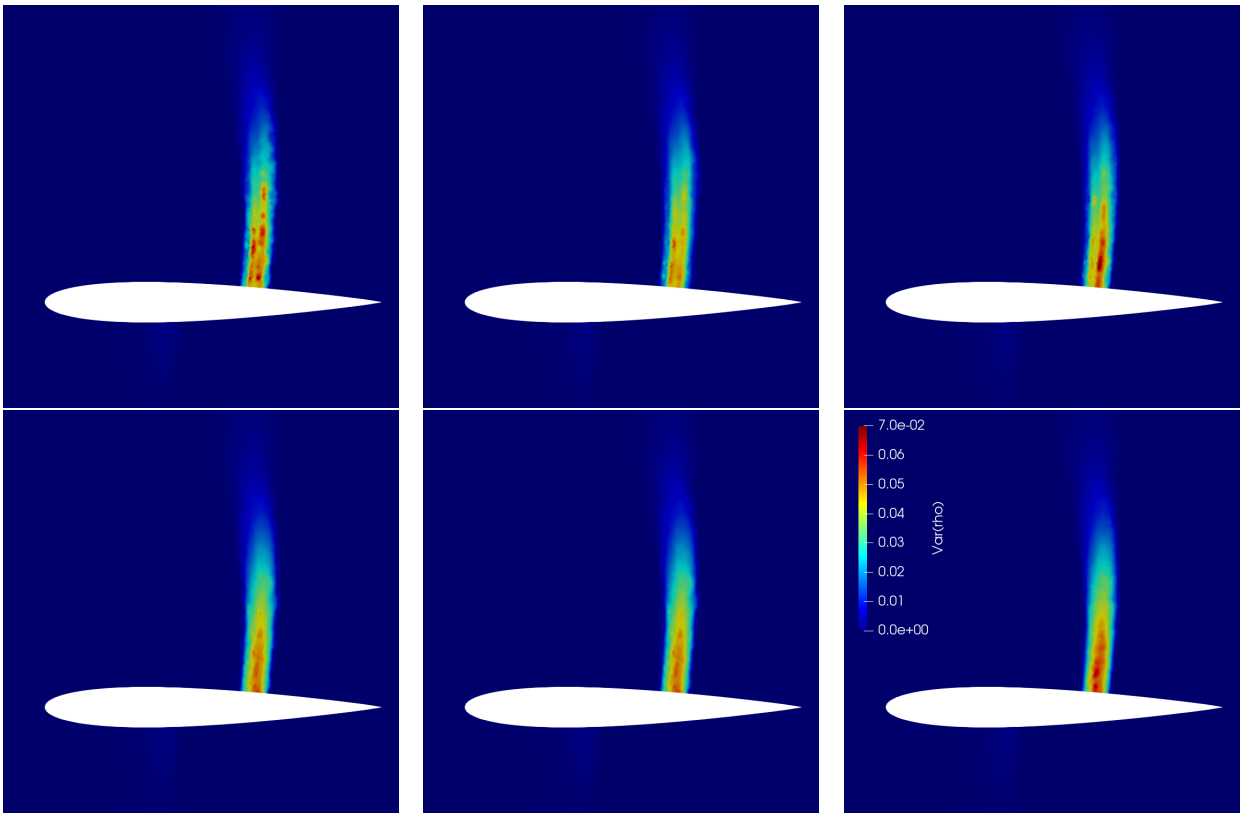

Figure 5: Var[ρ] (from top left to bottom right) SC, SG, IPM, caIPM, caosIPM, reference solution.

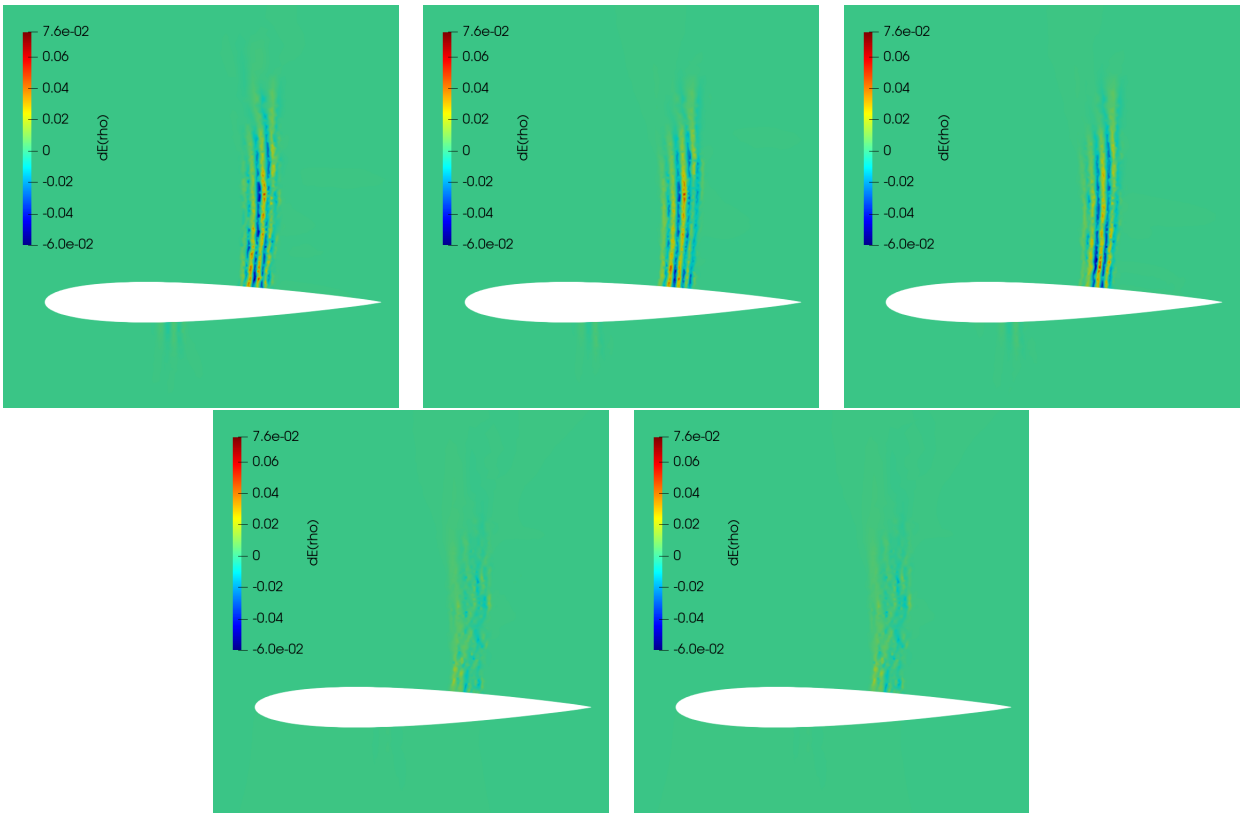

Figure 6: E[ρ] distance to reference solution (from top left to bottom right) SC, SG, IPM, caIPM, caosIPM.

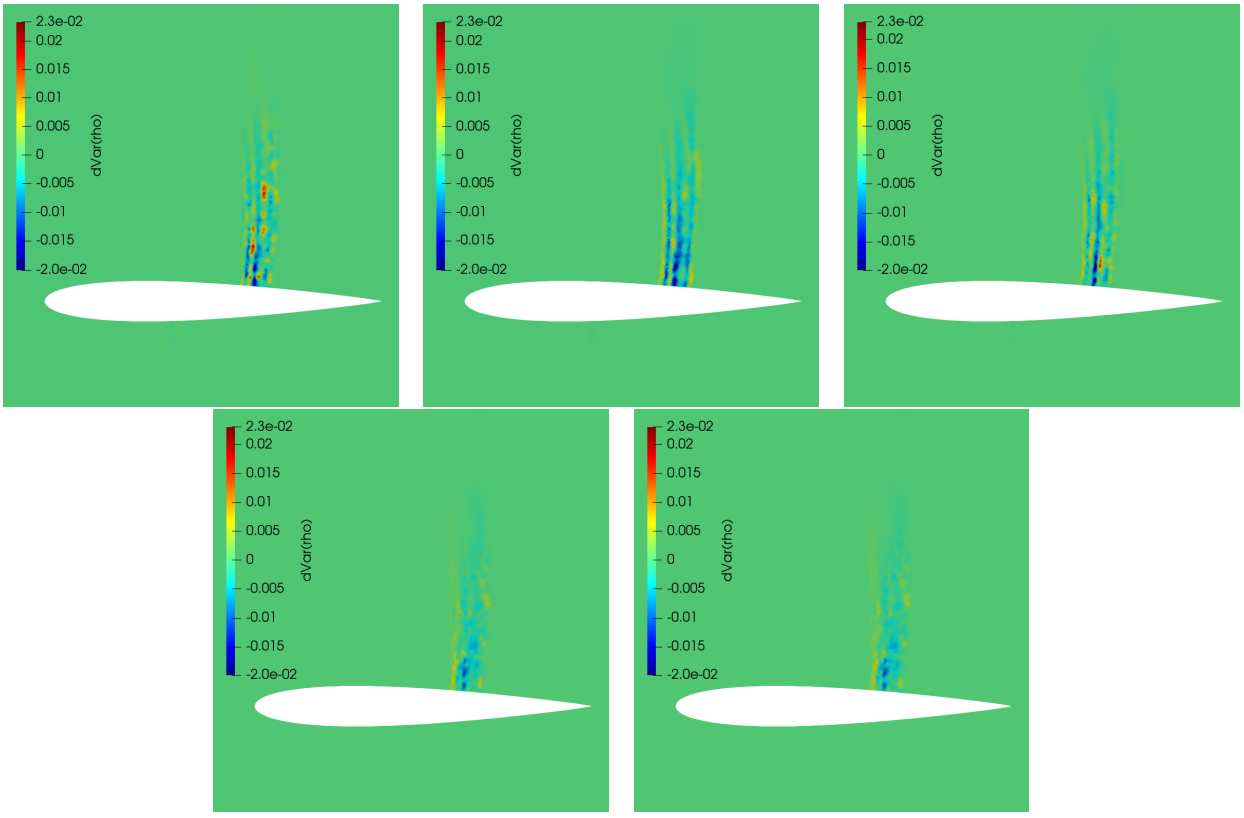

Figure 7: Var[ρ] distance to reference solution (from top left to bottom right) SC, SG, IPM, caIPM, caosIPM.

5. Summary and outlook

References

[1] Dongbin Xiu and George Em Karniadakis. The wiener–askey polynomial chaos for stochastic differential equations. *SIAM journal on scientific computing*, 24(2):619–644, 2002.

[2] Norbert Wiener. The homogeneous chaos. *American Journal of Mathematics*, 60(4):897–936, 1938.

[3] Dongbin Xiu and Jan S Hesthaven. High-order collocation methods for differential equations with random inputs. *SIAM Journal on Scientific Computing*, 27(3):1118–1139, 2005.

[4] Ivo Babuška, Fabio Nobile, and Raul Tempone. A stochastic collocation method for elliptic partial differential equations with random input data. *SIAM Journal on Numerical Analysis*, 45(3):1005–1034, 2007.

[5] GJA Loeven and H Bijl. Probabilistic collocation used in a two-step approach for efficient uncertainty quantification in computational fluid dynamics. *Computer Modeling in Engineering & Sciences*, 36(3):193–212, 2008.

[6] Dongbin Xiu. Fast numerical methods for stochastic computations: a review. *Communications in computational physics*, 5(2-4):242–272, 2009.

[7] AK Alekseev, IM Navon, and ME Zelentsov. The estimation of functional uncertainty using polynomial chaos and adjoint equations. *International Journal for numerical methods in fluids*, 67(3):328–341, 2011.

[8] Roger G Ghanem and Pol D Spanos. *Stochastic finite elements: a spectral approach*. Courier Corporation, 2003.

[9] Gaël Poëtte, Bruno Després, and Didier Lucor. Uncertainty quantification for systems of conservation laws. *Journal of Computational Physics*, 228(7):2443–2467, 2009.

[10] Louisa Schlachter and Florian Schneider. A hyperbolicity-preserving stochastic galerkin approximation for uncertain hyperbolic systems of equations. *Journal of Computational Physics*, 375:80–98, 2018.

[11] Jonas Kusch, Graham W Alldredge, and Martin Frank. Maximum-principle-satisfying second-order intrusive polynomial moment scheme. *arXiv preprint arXiv:1712.06966*, 2017.

[12] C. Kristopher Garrett, Cory Hauck, and Judith Hill. Optimization and large scale computation of an entropy-based moment closure. *Journal of Computational Physics*, 302:573–590, 2015.

[13] OP Le Maıtre, OM Knio, HN Najm, and RG Ghanem. Uncertainty propagation using wiener–haar expansions. *Journal of computational Physics*, 197(1):28–57, 2004.

[14] Jonas Kusch, Ryan G McClarren, and Martin Frank. Filtered stochastic galerkin methods for hyperbolic equations. *arXiv preprint arXiv:1808.00819*, 2018.

[15] Timothy Barth. Non-intrusive uncertainty propagation with error bounds for conservation laws containing discontinuities. In *Uncertainty quantification in computational fluid dynamics*, pages 1–57. Springer, 2013.

[16] Richard P Dwight, Jeroen AS Witteveen, and Hester Bijl. Adaptive uncertainty quantification for computational fluid dynamics. In *Uncertainty Quantification in Computational Fluid Dynamics*, pages 151–191. Springer, 2013.

[17] Per Pettersson, Gianluca Iaccarino, and Jan Nordström. Numerical analysis of the burgers' equation in the presence of uncertainty. *Journal of Computational Physics*, 228(22):8394–8412, 2009.

[18] Philipp Öffner, Jan Glaubitz, and Hendrik Ranocha. Stability of correction procedure via reconstruction with summation-by-parts operators for burgers' equation using a polynomial chaos approach. *arXiv preprint arXiv:1703.03561*, 2017.

[19] Julie Tryoen, O Le Maıtre, and Alexandre Ern. Adaptive anisotropic spectral stochastic methods for uncertain scalar conservation laws. *SIAM Journal on Scientific Computing*, 34(5):A2459–A2481, 2012.

[20] Ilja Kröker and Christian Rohde. Finite volume schemes for hyperbolic balance laws with multiplicative noise. *Applied Numerical Mathematics*, 62(4):441–456, 2012.

[21] Jan Giesselmann, Fabian Meyer, and Christian Rohde. A posteriori error analysis for random scalar conservation laws using the stochastic galerkin method. *arXiv preprint arXiv:1709.04351*, 2017.

[22] SB Hazra, V Schulz, J Brezillon, and NR Gauger. Aerodynamic shape optimization using simultaneous pseudo-timestepping. *Journal of Computational Physics*, 204(1):46–64, 2005.