

A LOW-RANK SOLVER FOR THE STOCHASTIC UNSTEADY NAVIER–STOKES PROBLEM*

HOWARD C. ELMAN[†] AND TENGFEI SU[‡]

Abstract. We study a low-rank iterative solver for the unsteady Navier–Stokes equations for incompressible flows with a stochastic viscosity. The equations are discretized using the stochastic Galerkin method, and we consider an all-at-once formulation where the algebraic systems at all the time steps are collected and solved simultaneously. The problem is linearized with Picard’s method. To efficiently solve the linear systems at each step, we use low-rank tensor representations within the Krylov subspace method, which leads to significant reductions in storage requirements and computational costs. Combined with effective mean-based preconditioners and the idea of inexact solve, we show that only a small number of linear iterations are needed at each Picard step. The proposed algorithm is tested with a model of flow in a two-dimensional symmetric step domain with different settings to demonstrate the computational efficiency.

Key words. time-dependent Navier–Stokes, stochastic Galerkin method, all-at-once system, low-rank tensor approximation

AMS subject classifications. 35R60, 60H35, 65F08, 65F10, 65N22

1. Introduction. Stochastic partial differential equations (PDEs) are widely used to model physical problems with uncertainty [16]. In this paper, we develop some new computational methods for solving the stochastic unsteady Navier–Stokes equations, using stochastic Galerkin methods [11] to address the stochastic nature of the problem and so-called all-at-once treatment of time integration.

For a time-dependent problem, the solutions at different time steps are usually computed in a sequential manner via time stepping. For example, a fully-implicit scheme with adaptive time step sizes was studied in [7, 14]. On the other hand, an all-at-once system can be formed by collecting the algebraic systems at all the discrete time steps into a single one, and the solutions are computed simultaneously. Such a formulation avoids the serial nature of time stepping, and allows parallelization in the time direction for accelerating the solution procedure [10, 18, 19]. A drawback, however, is that for large-size problems, the all-at-once system may require excessive storage. In this study, we address this issue by using a low-rank tensor representation of data within the solution methods.

We develop a low-rank iterative algorithm for solving the unsteady Navier–Stokes equations with an uncertain viscosity. The equations are linearized with Picard’s method. At each step of the nonlinear iteration, the stochastic Galerkin discretization gives rise to a large linear system, which is solved by a Krylov subspace method. Similar approaches have been used to study the steady-state problem [23, 27], where the authors also proposed effective preconditioners by taking advantage of the special structures of the linear systems. To reduce memory and computational costs, we compute low-rank approximations to the discrete solutions, which are represented as three-dimensional tensors in the all-at-once formulation. We refer to [12] for a

*This work was supported by the U.S. Department of Energy Office of Advanced Scientific Computing Research, Applied Mathematics program under award DE-SC0009301 and by the U.S. National Science Foundation under grant DMS1819115.

[†]Department of Computer Science and Institute for Advanced Computer Studies, University of Maryland, College Park, MD 20742 (elman@cs.umd.edu).

[‡]Applied Mathematics & Statistics, and Scientific Computation Program, University of Maryland, College Park, MD 20742 (tengfesu@math.umd.edu).

review of low-rank tensor approximation techniques, and we will use the tensor train decomposition [21] in this work. The tensor train decomposition allows efficient basic operations on tensors. A truncation procedure is also available to compress low-rank tensors in the tensor train format to ones with smaller ranks.

Our goal is to use the low-rank tensors within Krylov subspace methods, in order to efficiently solve the large linear systems arising in each nonlinear step. The basic idea is to represent all the vector quantities that arise during the course of a Krylov subspace computation as low-rank tensors. With this strategy, much less memory is needed to store the data produced during the iteration. Moreover, the associated computations, such as matrix-vector products and vector additions, become much cheaper. The tensors are compressed in each iteration to maintain low ranks. This idea has been used for the conjugate gradient (CG) method and the generalized minimal residual (GMRES) method, with different low-rank tensor formats [1, 2, 5, 15]. In addition, the convergence of Krylov subspace methods can be greatly improved by an effective preconditioner. In conjunction with the savings achieved through low-rank tensor computations, we will derive preconditioners for the stochastic all-at-once formulation based on some state-of-the-art techniques used for deterministic problems, and we will demonstrate their performances in numerical experiments. We also explore the idea of inexact Picard methods where the linear systems are solved inexactly at each Picard step to further save computational work, and we show that with this strategy very small numbers of iterations are needed for the Krylov subspace method.

We note that a different type of approach, the alternating iterative methods [6, 13, 25], including the density matrix renormalization group (DMRG) algorithm and its variants, can be used for solving linear systems in the tensor train format. In these methods, each component of the low-rank solution tensor is approached directly and optimized by projecting to a small local problem. This approach avoids the rank growth in intermediate iterates typically encountered in a low-rank Krylov subspace method. However, these methods are developed for solving symmetric positive definite systems and require nontrivial effort to be adapted for a nonsymmetric Navier–Stokes problem.

The rest of the paper is organized as follows. In section 2 we give a formal presentation of the problem. Discretization techniques that result in an all-at-once linear system at each Picard step are discussed in section 3. In section 4 we introduce the low-rank tensor approximation and propose a low-rank Krylov subspace iterative solver for the all-at-once systems. The preconditioners are derived in section 5 and numerical results are given in section 6.

2. Problem setting. Consider the unsteady Navier–Stokes equations for incompressible flows on a space-time domain $\mathcal{D} \times (0, t_f]$,

$$(2.1) \quad \begin{aligned} \frac{\partial \vec{u}}{\partial t} - \nabla \cdot (\nu \nabla \vec{u}) + \vec{u} \cdot \nabla \vec{u} + \nabla p &= \vec{0}, \\ \nabla \cdot \vec{u} &= 0, \end{aligned}$$

where \vec{u} and p stand for the velocity and pressure, respectively, ν is the viscosity, and \mathcal{D} is a two-dimensional spatial domain with boundary $\partial \mathcal{D} = \partial \mathcal{D}_D \cup \partial \mathcal{D}_N$. The Dirichlet boundary $\partial \mathcal{D}_D$ consists of an inflow boundary and fixed walls, and Neumann boundary conditions are set for the outflow,

$$(2.2) \quad \begin{aligned} \vec{u} &= \vec{u}_D && \text{on } \partial \mathcal{D}_D, \\ \nu \nabla \vec{u} \cdot \vec{n} - p \vec{n} &= \vec{0} && \text{on } \partial \mathcal{D}_N. \end{aligned}$$

We assume the Neumann boundary $\partial\mathcal{D}_N$ is not empty so that the pressure p is uniquely determined. The function $\bar{u}_D(x, t)$ denotes a time-dependent inflow, typically growing from zero to a steady state, and it is set to zero at fixed walls. The initial conditions are zero everywhere for both \bar{u} and p .

The uncertainty in the problem is introduced by a stochastic viscosity ν , which is modeled as a random field depending on a finite collection of random variables $\{\xi_l\}_{l=1}^m$ (or written as a vector ξ). Specifically, we consider a representation as a truncated Karhunen–Loève (KL, [17]) expansion,

$$(2.3) \quad \nu(x, \xi) = \nu_0(x) + \sum_{l=1}^m \nu_l(x) \xi_l,$$

where ν_0 is the mean viscosity, and $\{\nu_l\}_{l=1}^m$ are determined by the covariance function of ν . We assume that the random parameters $\{\xi_l\}_{l=1}^m$ are independent and that the viscosity satisfies $\nu(x, \xi) \geq \nu_{\min} > 0$ almost surely for any $x \in \mathcal{D}$. We refer to [23, 27] for different forms of the stochastic viscosity. The solutions \bar{u} and p in (2.1) will also be random fields which depend on the space parameter x , time t , and the random variables ξ .

3. Discrete problem. In this section, we derive a fully discrete problem for the stochastic unsteady Navier–Stokes equations (2.1). This involves a time discretization scheme and a stochastic Galerkin discretization for the physical and parameter spaces at each time step. The discretizations give rise to a nonlinear algebraic system. Instead of solving such a system at each time step, we collect the systems from all time steps to form an all-at-once system, where the discrete solutions at all the time steps are solved simultaneously. The discrete problem is then linearized with Picard’s method, and a large linear system is solved at each step of the nonlinear iteration.

3.1. Time discretization. For simplicity we use the backward Euler method for time discretization, which is first-order accurate but unconditionally stable and dissipative. The all-at-once formulation discussed later in section 3.3 requires predetermined time steps. Divide the interval $(0, t_f]$ into n_t uniform steps $\{t_k\}_{k=1}^{n_t}$ with step size $\tau = t_f/n_t$ and initial time $t_0 = 0$. Given the solution at time t_{k-1} , we need to solve the following equations for \bar{u}^k and p^k :

$$(3.1) \quad \begin{aligned} \frac{\bar{u}^k - \bar{u}^{k-1}}{\tau} - \nabla \cdot (\nu \nabla \bar{u}^k) + \bar{u}^k \cdot \nabla \bar{u}^k + \nabla p^k &= \vec{0}, \\ \nabla \cdot \bar{u}^k &= 0. \end{aligned}$$

After discretization (in physical space and parameter space) the implicit method requires solving an algebraic system at each time step. In the following we discuss how the system is assembled from the stochastic Galerkin discretization of (3.1).

3.2. Stochastic Galerkin method. At time step k , the stochastic Galerkin method finds parametrized approximate velocity solutions \bar{u}_h^k and pressure solutions p_h^k in finite-dimensional subspaces of $(H^1(\mathcal{D}))^2 \otimes L^2(\Gamma)$ and $L^2(\mathcal{D}) \otimes L^2(\Gamma)$, where Γ is the joint image of the random variables $\{\xi_l\}$. The functional spaces are defined as follows,

$$(3.2) \quad \begin{aligned} (H^1(\mathcal{D}))^2 \otimes L^2(\Gamma) &:= \left\{ \vec{v} : \mathcal{D} \times \Gamma \rightarrow \mathbb{R} \mid \mathbb{E}[\|\vec{v}\|_{(H^1(\mathcal{D}))^2}^2] < \infty \right\}, \\ L^2(\mathcal{D}) \otimes L^2(\Gamma) &:= \left\{ q : \mathcal{D} \times \Gamma \rightarrow \mathbb{R} \mid \mathbb{E}[\|q\|_{L^2(\mathcal{D})}^2] < \infty \right\}. \end{aligned}$$

The expectations are taken with respect to the joint distribution of the random variables $\{\xi_i\}$. In the following we use $\langle \cdot \rangle$ to denote the expected value. Let the finite-dimensional subspaces be $\mathcal{X} = \text{span}\{\vec{\phi}_i(x)\} \subset (H^1(\mathcal{D}))^2$, $\mathcal{Y} = \text{span}\{\varphi_i(x)\} \subset L^2(\mathcal{D})$, and $\mathcal{Z} = \text{span}\{\psi_r(\xi)\} \subset L^2(\Gamma)$. Let \mathcal{X}_D^k and \mathcal{X}_0 be the spaces of functions in \mathcal{X} with Dirichlet boundary conditions $\vec{u}_D(x, t_k)$ and $\vec{0}$ imposed for the velocity field, respectively. Then for (3.1) the stochastic Galerkin formulation entails the computation of $\vec{u}_h^k \in \mathcal{X}_D^k \otimes \mathcal{Z}$ and $p_h^k \in \mathcal{Y} \otimes \mathcal{Z}$, satisfying the weak form

$$(3.3) \quad \begin{aligned} \tau^{-1} \langle (\vec{u}_h^k, \vec{v}_h) \rangle - \tau^{-1} \langle (\vec{u}_h^{k-1}, \vec{v}_h) \rangle + \langle (\nu \nabla \vec{u}_h^k, \nabla \vec{v}_h) \rangle \\ + \langle (\vec{u}_h^k \cdot \nabla \vec{u}_h^k, \vec{v}_h) \rangle - \langle (p_h^k, \nabla \cdot \vec{v}_h) \rangle = 0, \\ \langle (\nabla \cdot \vec{u}_h^k, q_h) \rangle = 0, \end{aligned}$$

for any $\vec{v}_h \in \mathcal{X}_0 \otimes \mathcal{Z}$ and $q_h \in \mathcal{Y} \otimes \mathcal{Z}$. Here, (\cdot, \cdot) denotes the inner product in $L^2(\mathcal{D})$. For the physical spaces, we use a div-stable Taylor–Hood discretization [8] on quadrilateral elements, with biquadratic basis functions $\{\vec{\phi}_i\}_{i=1}^{n_u} = \left\{ \begin{pmatrix} \phi_i \\ 0 \end{pmatrix}, \begin{pmatrix} 0 \\ \phi_i \end{pmatrix} \right\}_{i=1}^{n_u/2}$ for velocity, and bilinear basis functions $\{\varphi_i\}_{i=1}^{n_p}$ for pressure. The stochastic basis functions $\{\psi_r\}_{r=1}^{n_\xi}$ are m -dimensional orthonormal polynomials constructed from generalized polynomial chaos (gPC, [28]) satisfying $\langle \psi_r, \psi_s \rangle = \delta_{rs}$. The stochastic Galerkin solutions are expressed as linear combinations of the basis functions,

$$(3.4) \quad \begin{aligned} \vec{u}_h^k(x, \xi) &= \sum_{s=1}^{n_\xi} \sum_{j=1}^{n_u} u_{js}^k \vec{\phi}_j(x) \psi_s(\xi), \\ p_h^k(x, \xi) &= \sum_{s=1}^{n_\xi} \sum_{j=1}^{n_p} p_{js}^k \varphi_j(x) \psi_s(\xi). \end{aligned}$$

The coefficient vectors $\mathbf{u}^k = [u_{11}^k, u_{21}^k, \dots, u_{n_u,1}^k, \dots, u_{1n_\xi}^k, u_{2n_\xi}^k, \dots, u_{n_u n_\xi}^k]$ and similarly defined \mathbf{p}^k are computed from the nonlinear algebraic system

$$(3.5) \quad \begin{pmatrix} \mathbb{F}^k(\mathbf{u}) & I_{n_\xi} \otimes B^T \\ I_{n_\xi} \otimes B & 0 \end{pmatrix} \begin{pmatrix} \mathbf{u}^k \\ \mathbf{p}^k \end{pmatrix} + \begin{pmatrix} -\tau^{-1}(I_{n_\xi} \otimes \mathbf{M}) & 0 \\ 0 & 0 \end{pmatrix} \begin{pmatrix} \mathbf{u}^{k-1} \\ \mathbf{p}^{k-1} \end{pmatrix} = \begin{pmatrix} \mathbf{f}^{u,k} \\ \mathbf{f}^{p,k} \end{pmatrix}$$

where

$$(3.6) \quad \mathbb{F}^k(\mathbf{u}) = \tau^{-1}(I_{n_\xi} \otimes \mathbf{M}) + \sum_{l=0}^m (G_l \otimes \mathbf{A}_l) + \sum_{l=1}^{n_\xi} (H_l \otimes \mathbf{N}(\vec{u}_{h,l}^k)).$$

Here I_{n_ξ} is the $n_\xi \times n_\xi$ identity matrix, and \otimes denotes the Kronecker product of two matrices. The boldface matrices \mathbf{M} , \mathbf{A}_l , and $\mathbf{N}(\vec{u}_{h,l}^k)$ are 2×2 block-diagonal, with the scalar mass matrix M , weighted stiffness matrix A_l , and discrete convection operator $N(\vec{u}_{h,l}^k)$ as diagonal components, where

$$(3.7) \quad [M]_{ij} = (\phi_j, \phi_i), [A_l]_{ij} = (\nu_l \nabla \phi_j, \nabla \phi_i), [N(\vec{u}_{h,l}^k)]_{ij} = (\vec{u}_{h,l}^k \cdot \nabla \phi_j, \phi_i),$$

for $i, j = 1, \dots, n_u/2$. Note the dependency on \mathbf{u}^k comes from the nonlinear convection term \mathbf{N} , with convection velocity $\vec{u}_{h,l}^k = \sum_j u_{jl}^k \vec{\phi}_j(x)$. Let $x = (x_1, x_2)$. The discrete divergence operator $B = [B_{x_1}, B_{x_2}]$, with

$$(3.8) \quad [B_{x_1}]_{ij} = -(\varphi_i, \frac{\partial \phi_j}{\partial x_1}), [B_{x_2}]_{ij} = -(\varphi_i, \frac{\partial \phi_j}{\partial x_2}),$$

for $i = 1, \dots, n_p$ and $j = 1, \dots, n_u/2$. The matrices $\{G_l\}_{l=0}^m$ and $\{H_l\}_{l=1}^{n_\xi}$ of (3.6) come from the stochastic basis functions and have entries

$$(3.9) \quad [G_l]_{rs} = \langle \xi_l \psi_r \psi_s \rangle, \quad [H_l]_{rs} = \langle \psi_l \psi_r \psi_s \rangle,$$

for $r, s = 1, \dots, n_\xi$, where $\xi_0 \equiv 1$. These matrices are also sparse due to orthogonality of the basis functions [9]. The Dirichlet boundary conditions are incorporated in the right-hand side of (3.5).

3.3. All-at-once system. As discussed in the beginning of the section, we consider an all-at-once system where the discrete solutions at all the time steps are computed together. Let

$$(3.10) \quad \mathbf{u} = \begin{pmatrix} \mathbf{u}^1 \\ \mathbf{u}^2 \\ \vdots \\ \mathbf{u}^{n_t} \end{pmatrix} \in \mathbb{R}^{n_t n_\xi n_u}$$

and let \mathbf{p} , \mathbf{f}^u , and \mathbf{f}^p be similarly defined. By collecting the algebraic systems (3.5) corresponding to all the time steps $\{t_k\}_{k=1}^{n_t}$, we get the single system

$$(3.11) \quad \begin{pmatrix} \mathbb{F}(\mathbf{u}) + \mathbb{C} & \mathbb{B}^T \\ \mathbb{B} & 0 \end{pmatrix} \begin{pmatrix} \mathbf{u} \\ \mathbf{p} \end{pmatrix} = \begin{pmatrix} \mathbf{f}^u \\ \mathbf{f}^p \end{pmatrix},$$

where $\mathbb{F}(\mathbf{u})$ is block diagonal with $\mathbb{F}^k(\mathbf{u})$ as the k th diagonal block, $\mathbb{B} = I_{n_t} \otimes I_{n_\xi} \otimes B$, and $\mathbb{C} = -\tau^{-1} C_{n_t} \otimes I_{n_\xi} \otimes \mathbf{M}$ with $C_{n_t} = \begin{pmatrix} 0 & & & \\ 1 & 0 & & \\ & \ddots & \ddots & \\ & & 1 & 0 \end{pmatrix} \in \mathbb{R}^{n_t \times n_t}$. Note that the zero initial conditions are incorporated in (3.5) for $k = 1$. The all-at-once system (3.11) is nonsymmetric and blockwise sparse. Each part of the system contains sums of Kronecker products of three matrices, i.e., in the form $\sum_l X_l^{(1)} \otimes X_l^{(2)} \otimes X_l^{(3)}$. In fact, from (3.6),

$$(3.12) \quad \mathbb{F}(\mathbf{u}) = \tau^{-1} I_{n_t} \otimes I_{n_\xi} \otimes \mathbf{M} + \sum_{l=0}^m (I_{n_t} \otimes G_l \otimes \mathbf{A}_l) + \mathbb{N}(\mathbf{u}).$$

We discuss later (see section 4.3) how the convection matrix \mathbb{N} can also be put in the Kronecker product form. It will be seen that this structure is useful for efficient matrix-vector product computations.

3.4. Picard's method. We use Picard's method to solve the nonlinear equation (3.11). Picard's method is a fixed-point iteration. Let $\mathbf{u}^{(i)}$, $\mathbf{p}^{(i)}$ be the approximate solutions at the i th step. Each Picard step entails solving a large linear system

$$(3.13) \quad \begin{pmatrix} \mathbb{F}(\mathbf{u}^{(i-1)}) + \mathbb{C} & \mathbb{B}^T \\ \mathbb{B} & 0 \end{pmatrix} \begin{pmatrix} \mathbf{u}^{(i)} \\ \mathbf{p}^{(i)} \end{pmatrix} = \begin{pmatrix} \mathbf{f}^u \\ \mathbf{f}^p \end{pmatrix}.$$

Instead of (3.13), one can equivalently solve the corresponding residual equation for a correction of the solution. Let $\mathbf{u}^{(i)} = \mathbf{u}^{(i-1)} + \delta \mathbf{u}^{(i)}$, $\mathbf{p}^{(i)} = \mathbf{p}^{(i-1)} + \delta \mathbf{p}^{(i)}$. Then $\delta \mathbf{u}^{(i)}$ and $\delta \mathbf{p}^{(i)}$ satisfy

$$(3.14) \quad \begin{pmatrix} \mathbb{F}(\mathbf{u}^{(i-1)}) + \mathbb{C} & \mathbb{B}^T \\ \mathbb{B} & 0 \end{pmatrix} \begin{pmatrix} \delta \mathbf{u}^{(i)} \\ \delta \mathbf{p}^{(i)} \end{pmatrix} = \begin{pmatrix} \mathbf{r}^{u, (i-1)} \\ \mathbf{r}^{p, (i-1)} \end{pmatrix},$$

where the nonlinear residual is

$$(3.15) \quad \mathbf{r}^{(i)} = \begin{pmatrix} \mathbf{r}^{u,(i)} \\ \mathbf{r}^{p,(i)} \end{pmatrix} = \begin{pmatrix} \mathbf{f}^u \\ \mathbf{f}^p \end{pmatrix} - \begin{pmatrix} \mathbb{F}(\mathbf{u}^{(i)}) + \mathbb{C} & \mathbb{B}^T \\ \mathbb{B} & 0 \end{pmatrix} \begin{pmatrix} \mathbf{u}^{(i)} \\ \mathbf{p}^{(i)} \end{pmatrix}.$$

Let \mathbf{f} denote the right-hand side of (3.11). The complete algorithm is summarized in Algorithm 3.1. The initial iterates $\mathbf{u}^{(0)}$, $\mathbf{p}^{(0)}$ are obtained as the solution of a Stokes problem, for which in (3.13) the convection matrix \mathbb{N} is set to zero.

Algorithm 3.1 Picard's method

- 1: Solve Stokes problem for initial $\mathbf{u}^{(0)}$, $\mathbf{p}^{(0)}$, update convection matrix $\mathbb{N}(\mathbf{u}^{(0)})$, and compute nonlinear residual $\mathbf{r}^{(0)}$. $i = 0$.
 - 2: **while** $\|\mathbf{r}^{(i)}\|_2 > \text{tol}_{\text{picard}} * \|\mathbf{f}\|_2$ and $i < \text{maxit}$ **do**
 - 3: $i = i + 1$
 - 4: Solve linear system (3.14) for $\delta\mathbf{u}^{(i)}$, $\delta\mathbf{p}^{(i)}$
 - 5: Update solution $\mathbf{u}^{(i)}$, $\mathbf{p}^{(i)}$
 - 6: Update convection matrix $\mathbb{N}(\mathbf{u}^{(i)})$
 - 7: Compute nonlinear residual $\mathbf{r}^{(i)}$
 - 8: **end while**
 - 9: **return** $\mathbf{u}^{(i)}$, $\mathbf{p}^{(i)}$
-

4. Low-rank approximation. In this section we discuss low-rank approximation techniques and how they can be used with iterative solvers. The computational cost of solving (3.14) at each Picard step is high due to the large problem size $n_t n_\xi (n_u + n_p)$, especially when large numbers of spatial grid points or time steps are used to achieve high-resolution solution. We will address this using low-rank tensor approximations to the solution vectors \mathbf{u} and \mathbf{p} . We will develop efficient iterative solvers and preconditioners where the solution is approximated using a compressed data representation in order to greatly reduce memory requirements and computational effort. The idea is to represent the iterates in an approximate Krylov subspace method in a low-rank tensor format. The basic operations associated with the low-rank format are much cheaper, and as the Krylov subspace method converges it constructs a sequence of low-rank approximations to the solution of the system.

4.1. Tensor train decomposition. A tensor $\underline{z} \in \mathbb{R}^{n_1 \times \dots \times n_d}$ is a multidimensional array with entries $z(i_1, \dots, i_d)$, where $i_l = 1, \dots, n_l$, $l = 1, \dots, d$. The solution coefficients in (3.4) can be represented in the form of three-dimensional $n_t \times n_\xi \times n_x$ tensors \underline{u} (where $n_x = n_u$) and \underline{p} ($n_x = n_p$), such that $\underline{u}(k, s, j) = u_{j_s}^k$ and $\underline{p}(k, s, j) = p_{j_s}^k$. Equivalently, such tensors can be represented in vector format, where the vector version \mathbf{u} and \mathbf{p} are specified using the vectorization operation

$$(4.1) \quad \mathbf{u} = \text{vec}(\underline{u}) \Leftrightarrow \mathbf{u}(\overline{i_1 i_2 i_3}) = \underline{u}(i_1, i_2, i_3)$$

where $\overline{i_1 i_2 i_3} = i_3 + (i_2 - 1)n_x + (i_1 - 1)n_\xi n_x$, and $\mathbf{p} = \text{vec}(\underline{p})$ in a similar manner. In an iterative solver for the system (3.14), any iterate \mathbf{z} can be equivalently represented as a three-dimensional tensor $\underline{z} \in \mathbb{R}^{n_t \times n_\xi \times n_x}$. In the sequel we use vector \mathbf{z} and tensor \underline{z} interchangeably. The tensor train decomposition [21] is a compressed low-rank representation to approximate a given tensor and efficiently perform tensor operations. Specifically, the tensor train format of \underline{z} is defined as

$$(4.2) \quad \underline{z}(i_1, i_2, i_3) \approx \sum_{\alpha_1, \alpha_2} \underline{z}^{(1)}(i_1, \alpha_1) \underline{z}^{(2)}(\alpha_1, i_2, \alpha_2) \underline{z}^{(3)}(\alpha_2, i_3),$$

where $\underline{z}^{(1)} \in \mathbb{R}^{n_t \times \kappa_1}$, $\underline{z}^{(2)} \in \mathbb{R}^{\kappa_1 \times n_\xi \times \kappa_2}$, $\underline{z}^{(3)} \in \mathbb{R}^{\kappa_2 \times n_x}$ are the tensor train cores, and κ_1 and κ_2 are called the tensor train ranks. It is easy to see that if $\kappa_1, \kappa_2 \approx \kappa$ and κ is small, the memory cost to store \underline{z} is reduced from $O(n_t n_\xi n_x)$ to $O((n_t + n_\xi \kappa + n_x) \kappa)$.

The tensor train decomposition allows efficient basic operations on tensors. Most importantly, matrix-vector products can be computed much less expensively if the vector \mathbf{z} is in the tensor train format. For \underline{z} as in (4.2), the vector \mathbf{z} has an equivalent Kronecker product form [6]

$$(4.3) \quad \mathbf{z} = \text{vec}(\underline{z}) = \sum_{\alpha_1, \alpha_2} z_{\alpha_1}^{(1)} \otimes z_{\alpha_1, \alpha_2}^{(2)} \otimes z_{\alpha_2}^{(3)},$$

where in the right-hand side $z_{\alpha_1}^{(1)}$, $z_{\alpha_1, \alpha_2}^{(2)}$, and $z_{\alpha_2}^{(3)}$ are vectors of length n_t , n_ξ , and n_x , respectively, obtained by fixing the indices α_1 and α_2 in $\underline{z}^{(1)}$, $\underline{z}^{(2)}$, and $\underline{z}^{(3)}$. Then for any matrix $\mathbb{X} = X^{(1)} \otimes X^{(2)} \otimes X^{(3)}$, such as the blocks in (3.14),

$$(4.4) \quad \mathbb{X}\mathbf{z} = \sum_{\alpha_1, \alpha_2} (X^{(1)} z_{\alpha_1}^{(1)}) \otimes (X^{(2)} z_{\alpha_1, \alpha_2}^{(2)}) \otimes (X^{(3)} z_{\alpha_2}^{(3)}).$$

The product is also in tensor train format with the same ranks as in \mathbf{z} (of the right-hand side of (4.2)), and it only requires matrix-vector products for each component of \mathbb{X} . From left to right in the Kronecker products, the component matrices from (3.12) are sparse with numbers of nonzeros proportional to n_t , n_ξ , and n_x , respectively, and the computational cost is thus reduced from $O(n_t n_\xi n_x)$ to $O((n_t + n_\xi \kappa + n_x) \kappa)$.

Other vector computations, including additions and inner products, are also expensive with the tensor train format. One thing to note is that the additions of two vectors in tensor train format will tend to increase the ranks. This can be easily seen from (4.2), since the addition of two low-rank tensors end up with more terms for the summation on the right-hand side. An important operation for the tensor train format is a truncation (or rounding) operation, used to reduce the ranks for tensors that are already in the tensor train format but have suboptimal high ranks. For a given tensor \underline{z} as in (4.2), the truncation operation \mathcal{T} with tolerance ϵ computes

$$(4.5) \quad \tilde{\underline{z}} = \mathcal{T}_\epsilon(\underline{z}),$$

such that $\tilde{\underline{z}}$ has smaller ranks than \underline{z} and satisfies the relative error

$$(4.6) \quad \|\tilde{\underline{z}} - \underline{z}\|_F / \|\underline{z}\|_F \leq \epsilon.$$

(Note that $\|\underline{z}\|_F = \|\mathbf{z}\|_2$.) The truncation operator is based on the TT-SVD algorithm [21], given in Algorithm 4.1, which is used to compute a low-rank tensor train approximation for a full tensor $\underline{z} \in \mathbb{R}^{n_1 \times \dots \times n_d}$. In the algorithm, a sequence of singular value decompositions (SVDs) is computed for the so-called unfolding matrix Z , obtained by reshaping the entries of a tensor into a two-dimensional array. Terms corresponding to small singular values are dropped such that an error E in the truncated SVD satisfies $\|E\|_F \leq \delta_j$, $j = 1, \dots, d-1$ (see line 4 of Algorithm 4.1). It was shown in [21] that the algorithm produces a tensor train $\tilde{\underline{z}}$ that satisfies

$$(4.7) \quad \|\underline{z} - \tilde{\underline{z}}\|_F \leq \left(\sum_{k=1}^{d-1} \delta_k^2 \right)^{1/2}.$$

Thus, one can choose $\delta_1 = \dots = \delta_{d-1} = \epsilon \|\underline{z}\|_F / \sqrt{d-1}$ to make the relative error $\|\tilde{\underline{z}} - \underline{z}\|_F / \|\underline{z}\|_F \leq \epsilon$. Note the algorithm is costly since it requires SVDs on matrices

$Z \in \mathbb{R}^{\kappa_{j-1}n_j \times n_{j+1} \cdots n_d}$. However, when the tensor \underline{z} is already in the tensor train format, the computation can be greatly simplified, and only SVDs on the much smaller tensor train cores are needed. In this case, the cost of the truncation operation is $O(dn\kappa^3)$ if $n_1, \dots, n_d \approx n$ and $\kappa_1, \dots, \kappa_{d-1} \approx \kappa$. We refer to [21] for more details. In the numerical experiments, we use TT-Toolbox [22] for tensor train computations.

Algorithm 4.1 TT-SVD

- 1: Let $Z = \underline{z}$. Set truncation parameters $\{\delta_j\}$. $\kappa_0 = 1$.
 - 2: **for** $j = 1, \dots, d - 1$ **do**
 - 3: $Z \leftarrow \text{reshape}(Z, [\kappa_{j-1}n_j, n_{j+1} \cdots n_d])$
 - 4: Compute truncated SVD $Z = U\Sigma V^T + E$, $\|E\|_F \leq \delta_j$, $\kappa_j = \text{rank}(\Sigma)$
 - 5: New core $\tilde{\underline{z}}^{(j)} \leftarrow \text{reshape}(U, [\kappa_{j-1}, n_j, \kappa_j])$
 - 6: Update $Z \leftarrow \Sigma V^T$
 - 7: **end for**
 - 8: New core $\tilde{\underline{z}}^{(d)} \leftarrow Z$
 - 9: **return** $\tilde{\underline{z}}$ in tensor train format with cores $\{\tilde{\underline{z}}^{(j)}\}$
-

4.2. Low-rank solver. The tensor train decomposition offers efficient tensor operations and we use it in iterative solvers to reduce the computational costs. The all-at-once system (3.14) to be solved at each step of Picard's method is nonsymmetric. We use a right-preconditioned GMRES method to solve the system. The complete algorithm for solving $\mathcal{L}\mathbf{z} = \mathbf{b}$ is summarized in Algorithm 4.2. The preconditioner \mathcal{P}^{-1} entails an inner iterative process and is not fixed for each GMRES iteration, and therefore a variant of the flexible GMRES method (see, e.g., [24]) is used. As discussed above, all the iterates in the algorithm are represented in the tensor train format for efficient computations, and a truncation operation with tolerance ϵ_{gmres} is used to compress the tensor train ranks so that they stay small relative to the problem size. It should be noted that since the quantities are truncated, the Arnoldi vectors $\{\mathbf{v}_i\}$ do not form orthogonal basis for the Krylov subspace, and thus this is not a true GMRES computation. When the algorithm is used for solving (3.14), the truncation operator is applied to quantities associated with the two tensor trains $\delta\mathbf{u}^{(i)}$ and $\delta\mathbf{p}^{(i)}$ separately. In section 5, we construct effective preconditioners for the system (3.14).

We also use the tensor train decomposition to construct a more efficient variant of Algorithm 3.1. In particular, the updated solutions $\mathbf{u}^{(i)}$ and $\mathbf{p}^{(i)}$ in line 5 are truncated, with a tolerance ϵ_{soln} , so that

$$(4.8) \quad \mathbf{u}^{(i)} = \mathcal{T}_{\epsilon_{\text{soln}}}(\mathbf{u}^{(i-1)} + \delta\mathbf{u}^{(i)}), \quad \mathbf{p}^{(i)} = \mathcal{T}_{\epsilon_{\text{soln}}}(\mathbf{p}^{(i-1)} + \delta\mathbf{p}^{(i)}).$$

Another truncation operation with ϵ_{gmres} is applied to compress the ranks of the nonlinear residual $\mathbf{r}^{(i)}$ in line 7. We will use this truncated version of Algorithm 3.1 in numerical experiments; choices of the truncation tolerances will be specified in section 6.

4.3. Convection matrix. We now show that in (3.12) if the velocity \mathbf{u} is in the tensor train format, the convection matrix $\mathbb{N}(\mathbf{u})$ can be represented as a sum of Kronecker products of matrices [3], which allows efficient matrix-vector product computations as in (4.4). Assume the coefficient tensor in (3.4) is approximated by a tensor train decomposition,

$$(4.9) \quad u_{jl}^k = \underline{u}(k, l, j) = \sum_{\alpha_1, \alpha_2} \underline{u}^{(1)}(k, \alpha_1) \underline{u}^{(2)}(\alpha_1, l, \alpha_2) \underline{u}^{(3)}(\alpha_2, j).$$

Algorithm 4.2 Low-rank GMRES method

- 1: Choose initial \mathbf{z}_0 , compute $\mathbf{s}_0 = \mathcal{T}_{\epsilon_{\text{gmres}}}(\mathbf{b} - \mathcal{L}\mathbf{z}_0)$, $\beta = \|\mathbf{s}_0\|_2$, and $\mathbf{v}_1 = \mathbf{s}_0/\beta$. Let $k = 0$.
 - 2: **while** $\|\mathbf{s}_k\|_2 > \text{tol}_{\text{gmres}} * \|\mathbf{b}\|_2$ and $k < \text{maxit}$ **do**
 - 3: $k = k + 1$
 - 4: Compute $\hat{\mathbf{v}}_k = \mathcal{P}^{-1}\mathbf{v}_k$
 - 5: Compute $\mathbf{x} = \mathcal{T}_{\epsilon_{\text{gmres}}}(\mathcal{L}\hat{\mathbf{v}}_k)$
 - 6: **for** $i = 1, \dots, k$ **do**
 - 7: $h_{ik} = \mathbf{x}^T \mathbf{v}_i$
 - 8: $\mathbf{x} = \mathbf{x} - h_{ik}\mathbf{v}_i$
 - 9: **end for**
 - 10: $h_{k+1,k} = \|\mathbf{x}\|_2$, $\mathbf{v}_{k+1} = \mathcal{T}_{\epsilon_{\text{gmres}}}(\mathbf{x}/h_{k+1,k})$
 - 11: Define $\hat{V}_k = [\hat{\mathbf{v}}_1, \dots, \hat{\mathbf{v}}_k]$ and $\bar{H} \in \mathbb{R}^{(k+1) \times k}$ with $\bar{H}_{ij} = h_{ij}$
 - 12: Compute $\mathbf{y}_k = \text{argmin}_y \|\beta e_1 - \bar{H}\mathbf{y}\|_2$, where $e_1 = [1, 0, \dots, 0]^T$
 - 13: Compute $\mathbf{z}_k = \mathcal{T}_{\epsilon_{\text{gmres}}}(\mathbf{z}_0 + \hat{V}_k \mathbf{y}_k)$
 - 14: Compute $\mathbf{s}_k = \mathcal{T}_{\epsilon_{\text{gmres}}}(\mathbf{b} - \mathcal{L}\mathbf{z}_k)$
 - 15: **end while**
 - 16: **return** \mathbf{z}_k
-

Note that the entries of $\mathbf{N}(\bar{u}_{h,l}^k)$ are linear in $\bar{u}_{h,l}^k$ and

$$(4.10) \quad \bar{u}_{h,l}^k = \sum_j u_{jl}^k \bar{\phi}_j(x) = \sum_{\alpha_1, \alpha_2} \underline{u}^{(1)}(k, \alpha_1) \underline{u}^{(2)}(\alpha_1, l, \alpha_2) \left(\sum_j \underline{u}^{(3)}(\alpha_2, j) \bar{\phi}_j(x) \right).$$

Let $\bar{u}_{\alpha_2}^{(3)} = \sum_j \underline{u}^{(3)}(\alpha_2, j) \bar{\phi}_j(x)$. Then the k th diagonal block of $\mathbb{N}(\mathbf{u})$ is

$$(4.11) \quad \sum_{l=1}^{n_\xi} (H_l \otimes \mathbf{N}(\bar{u}_{h,l}^k)) = \sum_{\alpha_1, \alpha_2} \underline{u}^{(1)}(k, \alpha_1) \sum_{l=1}^{n_\xi} (\underline{u}^{(2)}(\alpha_1, l, \alpha_2) H_l) \otimes \mathbf{N}(\bar{u}_{\alpha_2}^{(3)}).$$

The convection matrix $\mathbb{N}(\mathbf{u})$ can be expressed as

$$(4.12) \quad \mathbb{N}(\mathbf{u}) = \sum_{\alpha_1, \alpha_2} \text{diag}(u_{\alpha_1}^{(1)}) \otimes \sum_{l=1}^{n_\xi} (\underline{u}^{(2)}(\alpha_1, l, \alpha_2) H_l) \otimes \mathbf{N}(\bar{u}_{\alpha_2}^{(3)}).$$

Here $u_{\alpha_1}^{(1)}$ is a vector obtained by fixing the index α_1 in $\underline{u}^{(1)}$, and $\text{diag}(u_{\alpha_1}^{(1)})$ is a diagonal matrix with $u_{\alpha_1}^{(1)}$ on the diagonal. The result is a sum of Kronecker products of three smaller matrices. Such a representation can be constructed for any iterate $\mathbf{u}^{(i)}$ in the tensor train format.

Given the number of terms in the summation in the right-hand side of (4.12), the matrix-vector product with \mathbb{N} will result in a dramatic tensor train rank increase, from κ to κ^2 . Unless κ is very small, a tensor train with rank κ^2 will require too much memory and also be expensive to work with. To overcome this difficulty, when solving the all-at-once system (3.14), we use a low-rank approximation of $\mathbf{u}^{(i)}$ to construct $\mathbb{N}(\tilde{\mathbf{u}}^{(i)})$. Specifically, let

$$(4.13) \quad \tilde{\mathbf{u}}^{(i)} = \mathcal{T}_{\epsilon_{\text{conv}}}(\mathbf{u}^{(i)})$$

with some truncation tolerance ϵ_{conv} . Since $\tilde{\mathbf{u}}^{(i)}$ has smaller ranks than $\mathbf{u}^{(i)}$, the approximate convection matrix $\mathbb{N}(\tilde{\mathbf{u}}^{(i)})$ contains a smaller number of terms in (4.12),

and thus the rank increase becomes less significant when computing matrix-vector products with it. In other words, the linear system solved at each Picard step becomes

$$(4.14) \quad \begin{pmatrix} \mathbb{F}(\tilde{\mathbf{u}}^{(i-1)}) + \mathbb{C} & \mathbb{B}^T \\ \mathbb{B} & \mathbf{0} \end{pmatrix} \begin{pmatrix} \delta \mathbf{u}^{(i)} \\ \delta \mathbf{p}^{(i)} \end{pmatrix} = \begin{pmatrix} \mathbf{r}^{u,(i-1)} \\ \mathbf{r}^{p,(i-1)} \end{pmatrix}.$$

Note that the original $\mathbf{u}^{(i)}$ is still used for computing the nonlinear residual $\mathbf{r}^{(i)}$ in Picard's method.

5. Preconditioning. In this section we discuss preconditioning techniques for the all-at-once system (3.14) so that the Krylov subspace methods converge in a small number of iterations. To simplify the notation, we use \mathbf{w} instead of $\mathbf{u}^{(i-1)}$, and the associated approximate solution at the k th time step is

$$(5.1) \quad \bar{w}_h^k(x, \xi) = \sum_{l=1}^{n_\xi} \sum_{j=1}^{n_u} w_{jl}^k \vec{\phi}_j(x) \psi_l(\xi) = \sum_{l=1}^{n_\xi} \bar{w}_{h,l}^k(x) \psi_l(\xi)$$

with $\bar{w}_{h,l}^k(x) = \sum_j w_{jl}^k \vec{\phi}_j(x)$. In the following the dependence on \mathbf{w} in $\mathbb{F}(\mathbf{w})$ is omitted in most cases. We derive a preconditioner by extending ideas for more standard problems [8], starting with an ‘‘idealized’’ block triangular preconditioner

$$(5.2) \quad \mathcal{P} = \begin{pmatrix} \mathbb{F} + \mathbb{C} & \mathbb{B}^T \\ \mathbf{0} & -\mathbb{S} \end{pmatrix}.$$

With this choice of preconditioner, the Schur complement is $\mathbb{S} = \mathbb{B}(\mathbb{F} + \mathbb{C})^{-1}\mathbb{B}^T$, and the idealized preconditioned system derived from a block factorization

$$(5.3) \quad \begin{pmatrix} \mathbb{F} + \mathbb{C} & \mathbb{B}^T \\ \mathbb{B} & \mathbf{0} \end{pmatrix} \mathcal{P}^{-1} = \begin{pmatrix} \mathbb{I} & \mathbf{0} \\ \mathbb{B}\mathbb{F}^{-1} & \mathbb{I} \end{pmatrix}$$

has eigenvalues equal to 1 and Jordan blocks of order 2. (Here \mathbb{I} is an identity block.) Thus a right-preconditioned true GMRES method will converge in two iterations. However, the application of \mathcal{P}^{-1} involves solving linear systems associated with \mathbb{S} and $\mathbb{F} + \mathbb{C}$. These are too expensive for practical computation and to develop preconditioners we will construct inexpensive approximations to the linear solves. Specifically, we derive mean-based preconditioners that use results from the mean deterministic problem. Such preconditioners for the stochastic steady-state Navier–Stokes equations have been studied in [23]. We generalize the techniques for the all-at-once formulation of the unsteady equations.

5.1. Deterministic operator. We review the techniques used for approximating the Schur complement in the deterministic case [8]. The approximations are based on the fact that a commutator of the convection-diffusion operator with the divergence operator

$$(5.4) \quad \mathcal{E} = \nabla \cdot (-\nu \nabla^2 + \bar{w}_{h,1}^k \cdot \nabla) - (-\nu \nabla^2 + \bar{w}_{h,1}^k \cdot \nabla)_p \nabla \cdot$$

is small under certain assumptions about smoothness and boundary conditions. The subscript p means the operators are defined on the pressure space. For a discrete convection-diffusion operator $\mathbf{F} = \mathbf{A}_0 + \mathbf{N}(\bar{w}_{h,1}^k)$ (which is part of the mean problem we discuss later), as defined in (3.7), an approximation to the Schur complement $S = B\mathbf{F}^{-1}B^T$ is identified from a discrete analogue of (5.4),

$$(5.5) \quad E = (M_p^{-1}B)(M^{-1}\mathbf{F}) - (M_p^{-1}F_p)(M_p^{-1}B) \approx 0,$$

where the subscript p means the corresponding matrices constructed on the discrete pressure space. Equation (5.5) leads to an approximation to the Schur complement matrix,

$$(5.6) \quad S = B\mathbf{F}^{-1}B^T \approx M_p F_p^{-1} B M^{-1} B^T.$$

The pressure convection-diffusion (PCD) preconditioner is constructed by replacing the mass matrices with approximations containing only their diagonal entries (denoted by a subscript $*$) in (5.6),

$$(5.7) \quad S_{\text{PCD}}^{-1} = (B M_*^{-1} B^T)^{-1} F_p M_{p*}^{-1}.$$

The least-squares commutator (LSC) preconditioner avoids the construction of matrices on the pressure space, with the approximation to F_p ,

$$(5.8) \quad F_p \approx (B M^{-1} \mathbf{F} M^{-1} B^T) (B M^{-1} B^T)^{-1} M_p$$

(see [8, section 9.2] for a derivation). The LSC preconditioner is obtained by substituting F_p in (5.6) and replacing the mass matrices with their diagonals,

$$(5.9) \quad S_{\text{LSC}}^{-1} = (B M_*^{-1} B^T)^{-1} (B M_*^{-1} \mathbf{F} M_*^{-1} B^T) (B M_*^{-1} B^T)^{-1}.$$

For both preconditioners, the only use of the matrices \mathbf{F} and F_p is through matrix-vector products with them.

5.2. Approximations to S^{-1} . The Schur complement \mathbb{S} involves $(\mathbb{F} + \mathbb{C})^{-1}$ and is impractical to work with. For our stochastic unsteady problem, we consider mean-based preconditioners that use approximations to the Schur complement matrix

$$(5.10) \quad \mathbb{S}_0 = \mathbb{B}(\mathbb{F}_0 + \mathbb{C})^{-1} \mathbb{B}^T,$$

where the “mean” matrix \mathbb{F}_0 is block-diagonal with \mathbb{F}_0^k as the k th diagonal block, and

$$(5.11) \quad \mathbb{F}_0^k = \tau^{-1}(I_{n_\xi} \otimes \mathbf{M}) + I_{n_\xi} \otimes \mathbf{A}_0 + I_{n_\xi} \otimes \mathbf{N}(\vec{w}_{h,1}^k).$$

This corresponds to taking only the first term in the two summations on the right-hand side of (3.6). Since the gPC basis functions are orthonormal with $\langle \psi_r \psi_s \rangle = \delta_{rs}$ and $\psi_1 \equiv 1$, it follows $\langle \psi_s \rangle = \delta_{1s}$, and $G_0 = H_1 = I_{n_\xi}$. The matrices \mathbf{A}_0 and $\mathbf{N}(\vec{w}_{h,1}^k)$ are constructed from the mean of ν and \vec{w}_h^k ,

$$(5.12) \quad \langle \nu \rangle = \nu_0, \quad \langle \vec{w}_h^k \rangle = \sum_j w_{j1}^k \vec{\phi}_j(x) = \vec{w}_{h,1}^k.$$

The matrix \mathbb{F}_0^k can be expressed as $I_{n_\xi} \otimes (\tau^{-1} \mathbf{M} + \mathbf{A}_0 + \mathbf{N}(\vec{w}_{h,1}^k))$ and this enables use of approximations associated with a deterministic problem. Now, similarly define $\mathbb{F}_{p,0}$ on the pressure space, with

$$(5.13) \quad \mathbb{F}_{p,0}^k = \tau^{-1}(I_{n_\xi} \otimes M_p) + I_{n_\xi} \otimes A_{p,0} + I_{n_\xi} \otimes N_p(\vec{w}_{h,1}^k).$$

Let $\mathbb{M} = I_{n_t} \otimes I_{n_\xi} \otimes \mathbf{M}$ and $\mathbb{M}_p = I_{n_t} \otimes I_{n_\xi} \otimes M_p$. Assuming the validity of (5.5) it is easy to check that

$$(5.14) \quad \mathbb{M}_p^{-1} \mathbb{B} \mathbb{M}^{-1} \mathbb{F}_0 - \mathbb{M}_p^{-1} \mathbb{F}_{p,0} \mathbb{M}_p^{-1} \mathbb{B} \approx 0.$$

On the other hand, let $\mathbb{C}_p = -\tau^{-1}C_{n_t} \otimes I_{n_\xi} \otimes M_p$, so that \mathbb{C} satisfies

$$(5.15) \quad \mathbb{M}_p^{-1} \mathbb{B} \mathbb{M}^{-1} \mathbb{C} - \mathbb{M}_p^{-1} \mathbb{C}_p \mathbb{M}_p^{-1} \mathbb{B} = 0.$$

Combining (5.14) and (5.15) gives an approximation to \mathbb{S}_0 ,

$$(5.16) \quad \mathbb{S}_0 = \mathbb{B}(\mathbb{F}_0 + \mathbb{C})^{-1} \mathbb{B}^T \approx \mathbb{M}_p(\mathbb{F}_{p,0} + \mathbb{C}_p)^{-1} \mathbb{B} \mathbb{M}^{-1} \mathbb{B}^T.$$

Then the mean-based PCD preconditioner is given as

$$(5.17) \quad \mathbb{S}_{\text{PCD},0}^{-1} = (\mathbb{B} \mathbb{M}_*^{-1} \mathbb{B}^T)^{-1} (\mathbb{F}_{p,0} + \mathbb{C}_p) \mathbb{M}_{p*}^{-1},$$

where $\mathbb{M}_* = I_{n_t} \otimes I_{n_\xi} \otimes \mathbf{M}_*$ and $\mathbb{M}_{p*} = I_{n_t} \otimes I_{n_\xi} \otimes M_{p*}$. Similarly from (5.8), it holds that

$$(5.18) \quad \mathbb{F}_{p,0} + \mathbb{C}_p \approx (\mathbb{B} \mathbb{M}^{-1} (\mathbb{F}_0 + \mathbb{C}) \mathbb{M}^{-1} \mathbb{B}^T) (\mathbb{B} \mathbb{M}^{-1} \mathbb{B})^{-1} \mathbb{M}_p.$$

Substituting $\mathbb{F}_{p,0} + \mathbb{C}_p$ in (5.17) and replacement of the mass matrices with their diagonals gives the mean-based LSC preconditioner

$$(5.19) \quad \mathbb{S}_{\text{LSC},0}^{-1} = (\mathbb{B} \mathbb{M}_*^{-1} \mathbb{B}^T)^{-1} (\mathbb{B} \mathbb{M}_*^{-1} (\mathbb{F}_0 + \mathbb{C}) \mathbb{M}_*^{-1} \mathbb{B}^T) (\mathbb{B} \mathbb{M}_*^{-1} \mathbb{B}^T)^{-1}.$$

The two mean-based preconditioners in (5.17) and (5.19) have the same form as for the deterministic problem, except that there is an extra term \mathbb{C} or \mathbb{C}_p from the all-at-once formulation. Computations associated with the two approximations to the Schur complement are also inexpensive. For example, $(\mathbb{B} \mathbb{M}_*^{-1} \mathbb{B}^T)^{-1} = I_{n_t} \otimes I_{n_\xi} \otimes (\mathbf{B} \mathbf{M}_*^{-1} \mathbf{B}^T)^{-1}$, and this only requires solving a system with $\mathbf{B} \mathbf{M}_*^{-1} \mathbf{B}^T$ a discrete Laplacian. Multiplications with the mean matrix $\mathbb{F}_0 + \mathbb{C}$ are reduced to its components (see (4.4)),

$$(5.20) \quad \mathbb{F}_0 + \mathbb{C} = \tau^{-1} (I_{n_t} \otimes I_{n_\xi} \otimes \mathbf{M}) + I_{n_t} \otimes I_{n_\xi} \otimes \mathbf{A}_0 + \mathbb{N}_0 - \tau^{-1} (C_{n_t} \otimes I_{n_\xi} \otimes \mathbf{M}).$$

The matrix \mathbb{N}_0 is block-diagonal with $\mathbb{N}_0^k = I_{n_\xi} \otimes \mathbf{N}(\vec{w}_{h,1}^k)$ and can be expressed as a sum of Kronecker products of matrices as discussed in section 4.3,

$$(5.21) \quad \mathbb{N}_0(\mathbf{w}) = \sum_{\alpha_1, \alpha_2} \text{diag}(w_{\alpha_1}^{(1)}) \otimes (\underline{w}^{(2)}(\alpha_1, 1, \alpha_2) I_{n_\xi}) \otimes \mathbf{N}(\vec{w}_{\alpha_2}^{(3)}).$$

5.3. System solve with $\mathbb{F} + \mathbb{C}$. The application of the preconditioner \mathcal{P}^{-1} in (5.2) also involves solving a linear system associated with the (1,1) block $\mathbb{F} + \mathbb{C}$. For approximation, we replace it with the mean matrix $\mathbb{F}_0 + \mathbb{C}$, and solve a system of the form

$$(5.22) \quad (\mathbb{F}_0 + \mathbb{C}) \mathbf{v} = \mathbf{y}.$$

For such a system it is easy to compute matrix-vector products and we again use a low-rank GMRES method for solving the system. This inner GMRES solver is preconditioned with

$$(5.23) \quad \mathcal{M} = (I_{n_t} - C_{n_t}) \otimes I_{n_\xi} \otimes (\tau^{-1} \mathbf{M} + \mathbf{A}_0 + \mathbf{N}(\vec{w}_{h,1}^{\text{avg}})),$$

where $\vec{w}_{h,1}^{\text{avg}}$ is the average of $\vec{w}_{h,1}^k$ over all time steps. For small time step τ , the contribution from the mass matrix, $\tau^{-1} \mathbf{M}$, becomes dominant and \mathcal{M} forms a good approximation to the coefficient matrix $\mathbb{F}_0 + \mathbb{C}$. The application of \mathcal{M}^{-1} is also conveniently reduced to computations associated with smaller matrices. We note that (5.22) need not be solved accurately. In particular, with a stopping criterion $\|\mathbf{y} - (\mathbb{F}_0 + \mathbb{C}) \mathbf{v}\|_2 \leq \text{tol} \|\mathbf{y}\|_2$, a relatively large stopping tolerance, e.g., $\text{tol} = 10^{-1}$, will suffice for the mean-based preconditioner \mathcal{P} to be effective.

Remark 5.1. For systems like (5.22), a block diagonal preconditioner ($\mathcal{M} = \mathbb{F}_0$) was studied in [20], where it was shown that preconditioned GMRES converges very slowly before a sharp drop in the residual occurs when the number of iterations reaches n_t , which is equal to the number of diagonal blocks. In numerical experiments, we found that the preconditioner in (5.23) is more effective than a block diagonal one, for which performance deteriorates as τ becomes smaller.

6. Numerical experiments.

6.1. Benchmark problem. Consider a flow around a symmetric step where the spatial domain \mathcal{D} is a two-dimensional rectangular duct with a symmetric expansion (see Figure 6.1). The Dirichlet inflow boundary conditions at $(-1, x_2)$, $|x_2| \leq 0.5$ are deterministic and time-dependent, growing from zero to a steady parabolic profile,

$$(6.1) \quad \vec{u}_D((-1, x_2), t) = \begin{pmatrix} 1 - 4x_2^2 \\ 0 \end{pmatrix} (1 - e^{-10t}).$$

Neumann boundary conditions $\nu \partial u_{x_1} / \partial x_1 = p$, $\partial u_{x_2} / \partial x_1 = 0$ are imposed at the outflow boundary $(12, x_2)$, $|x_2| \leq 1$, and no-flow conditions $\vec{u} = \vec{0}$ at the fixed walls $(x_1, \pm 1)$, $0 \leq x_1 \leq 12$; $(x_1, \pm 0.5)$, $-1 \leq x_1 \leq 0$; $(0, x_2)$, $0.5 \leq |x_2| \leq 1$. The initial conditions are zero everywhere for both \vec{u} and p . The Taylor-Hood spatial discretization with biquadratic basis functions for the velocity space and bilinear basis functions for the pressure space is defined on a uniform grid of square elements with mesh size h , and it is constructed using the IFISS software package [26].

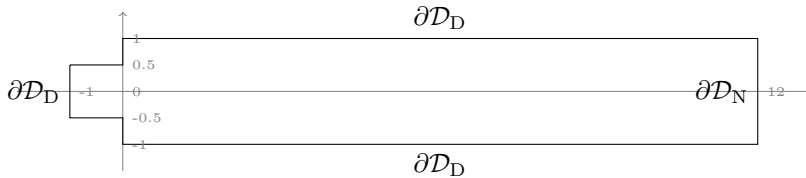


FIG. 6.1. Symmetric step domain with boundary conditions.

The stochastic viscosity $\nu(x, \xi)$ is represented as a truncated KL expansion

$$(6.2) \quad \nu(x, \xi) = \nu_0 \left(1.0 + \sigma \sum_{l=1}^m \sqrt{\beta_l} a_l(x) \xi_l \right).$$

The constants ν_0 and $\nu_0 \sigma$ represent the mean and the standard deviation of the stochastic field. We use an exponential covariance function $c(x, y) = \exp(-\|x - y\|_1 / b)$, where b is the correlation length. The pair $(\beta_l, a_l(x))$ is the l th largest eigenvalue and the corresponding eigenfunction of $c(x, y)$, satisfying

$$(6.3) \quad \int_{\mathcal{D}} c(x, y) a_l(y) dy = \beta_l a_l(x).$$

This can be computed with a standard finite element method. The random variables $\{\xi_l\}_{l=1}^m$ are assumed to be independent and each of them uniformly distributed on the interval $[-\sqrt{3}, \sqrt{3}]$, so they have zero means and unit variances. For the stochastic Galerkin method, the basis functions $\{\psi_r\}_{r=1}^{n_\xi}$ are m -dimensional Legendre polynomials, with total degrees bounded by d_ψ . Then the number of stochastic basis functions

is $n_\xi = (m + d_\psi)! / (m!d_\psi!)$. In the numerical experiments, unless otherwise stated, the parameter values associated with the discrete problem are chosen as in [Table 6.1](#). This gives a problem with dimensions $n_t = 64$, $n_\xi = 20$, $n_u = 2992$, $n_p = 461$, and $n_t n_\xi (n_u + n_p) = 4419840$. All computations are done in MATLAB 9.4.0 (R2018a) on a desktop with 64 GB memory.

TABLE 6.1
Parameter values for numerical experiments.

ν_0	σ	b	m	d_ψ	t_f	τ	h
1/50	0.01	4.0	3	3	1.0	2^{-6}	2^{-2}

6.2. Inexact Picard method. The main computational cost associated with Picard’s method is to solve an all-at-once system [\(3.14\)](#) at each step. In [section 4](#) we discussed how to construct low-rank approximate solutions in tensor train format with much cheaper computations. To further reduce the cost, we adopt the idea of inexact Picard method [\[4\]](#), where the linear systems are solved inexactly to save unnecessary computational work. Let [\(3.14\)](#) be denoted as $\mathcal{L}\mathbf{z}^{(i)} = \mathbf{r}^{(i-1)}$, and define the residual norm $\|\mathbf{s}_k\|_2 = \|\mathbf{r}^{(i-1)} - \mathcal{L}\mathbf{z}_k^{(i)}\|_2$ for an approximate solution $\mathbf{z}_k^{(i)}$. It was shown in [\[4\]](#) that if the stopping criterion for the linear solve (line 2 of [Algorithm 4.2](#)) is given as

$$(6.4) \quad \|\mathbf{s}_k\|_2 \leq \text{tol}_{\text{gmres}} \|\mathbf{r}^{(i-1)}\|_2,$$

then Picard’s method converges as long as $\text{tol}_{\text{gmres}} < 1$. This is especially helpful for our low-rank GMRES method. The best accuracy that the low-rank GMRES method can achieve is related to the truncation tolerance ϵ_{gmres} used in the algorithm (see [Figure 6.2a](#)). A relaxed stopping tolerance not only reduces the number of GMRES iterations, but it also allows use of larger truncation tolerances for tensor rank compressions, resulting in smaller ranks for the iterates and more efficient computations in the iterative solver. In the numerical tests, we set $\text{tol}_{\text{gmres}} = 10^{-1}$ and $\epsilon_{\text{gmres}} = 10^{-2} * \text{tol}_{\text{gmres}} = 10^{-3}$. The same tolerances are used for solving the linear system [\(5.22\)](#) required for the preconditioning operation. For the initial $\mathbf{u}^{(0)}$, $\mathbf{p}^{(0)}$, the Stokes problem is solved to satisfy $\|\mathbf{s}_k\|_2 \leq \text{tol}_{\text{gmres}} \|\mathbf{f}\|_2$ where \mathbf{f} is the right-hand side of [\(3.13\)](#).

6.3. Numerical results. In the following, we examine the performance of the proposed low-rank algorithm in different settings. The choices of stopping and truncation tolerances are summarized in [Table 6.2](#). In [Algorithm 3.1](#), the stopping criterion for Picard’s method is

$$(6.5) \quad \|\mathbf{r}^{(i)}\|_2 \leq \text{tol}_{\text{picard}} * \|\mathbf{f}\|_2.$$

We set $\text{tol}_{\text{picard}} = 10^{-5}$. A small truncation tolerance $\epsilon_{\text{soln}} = 10^{-7}$ is used to produce low-rank approximate solutions $\mathbf{u}^{(i)}$ and $\mathbf{p}^{(i)}$ in [\(4.8\)](#). It is shown in [Figure 6.2b](#) that, like the exact method, the inexact Picard method still exhibits a linear convergence rate. It takes 5 Picard steps to reach the required accuracy. [Figure 6.3](#) shows the tensor train ranks κ_1 and κ_2 of the iterates at each Picard step. As the Picard iteration converges, the right-hand side of [\(6.4\)](#) becomes smaller, and the corrections $\delta\mathbf{u}^{(i)}$ and $\delta\mathbf{p}^{(i)}$ computed from the low-rank GMRES method have increasing ranks. On the other hand, for the approximate solutions $\mathbf{u}^{(i)}$ and $\mathbf{p}^{(i)}$, their ranks drop to smaller values in the latter steps of the iteration. With a more stringent $\text{tol}_{\text{picard}}$, a smaller

ϵ_{soln} is required and the approximate solutions have slightly higher ranks than those shown in Figure 6.3b. Also shown in Figure 6.3b are the tensor train ranks of $\tilde{\mathbf{u}}^{(i)}$ for constructing the approximate convection matrices using (4.13). They have much smaller values than the ranks of $\mathbf{u}^{(i)}$.

TABLE 6.2
Stopping and truncation tolerances.

GMRES stopping tolerance	$tol_{\text{gmres}} = 10^{-1}$
Picard stopping tolerance	$tol_{\text{picard}} = 10^{-5}$
GMRES truncation tolerance	$\epsilon_{\text{gmres}} = 10^{-3}$
Truncation tolerance for solutions	$\epsilon_{\text{soln}} = 10^{-7}$
Truncation tolerance for convection matrix	$\epsilon_{\text{conv}} = 10^{-3}$

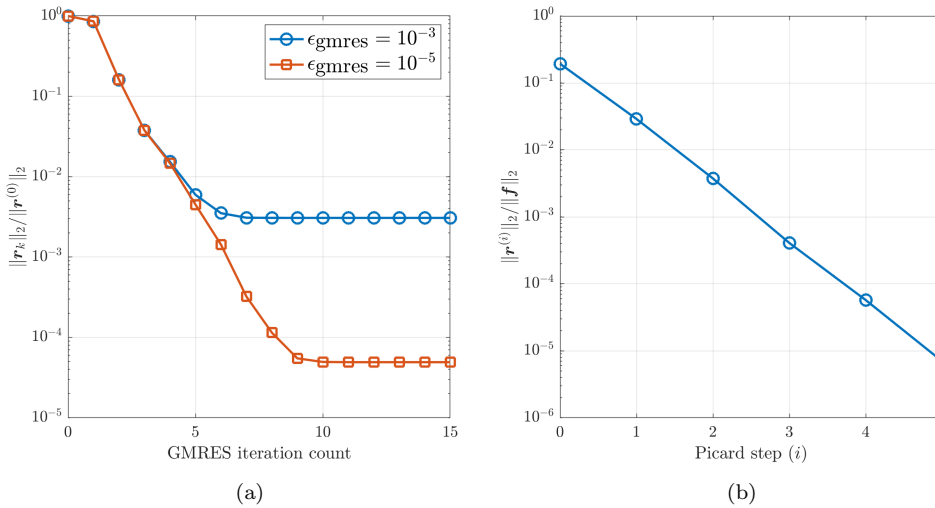


FIG. 6.2. (a) Convergence of the low-rank GMRES method (at the first Picard step) with different truncation tolerances. (b) Convergence of the inexact Picard method with tolerances chosen as in Table 6.2. LSC preconditioner is used.

We demonstrate the savings obtained from the inexact solves. Table 6.3 shows the performance of Picard’s method if different stopping tolerances are used in (6.4). With a larger tol_{gmres} , the number of Picard steps does not increase, while the total number of GMRES iterations and the associated computational costs are greatly reduced.

TABLE 6.3
Performance of Picard’s method with different values of GMRES stopping tolerance tol_{gmres} . Truncation tolerance $\epsilon_{\text{gmres}} = 10^{-2} * tol_{\text{gmres}}$. $tol_{\text{picard}} = 10^{-5}$. LSC preconditioner is used.

tol_{gmres}	10^{-1}	10^{-3}	10^{-5}
ϵ_{gmres}	10^{-3}	10^{-5}	10^{-7}
Number of Picard steps	5	5	5
Total number of GMRES iterations	18	39	58
Computational time (s)	205.9	555.7	1168.2

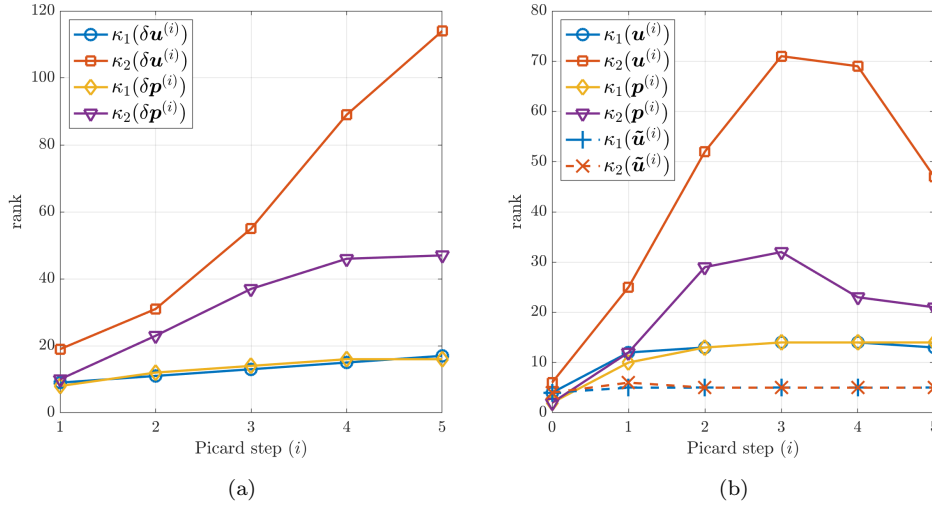


FIG. 6.3. (a) Tensor train ranks of corrections $\delta \mathbf{u}^{(i)}$ and $\delta \mathbf{p}^{(i)}$. (b) Tensor train ranks of approximate solutions $\mathbf{u}^{(i)}$ and $\mathbf{p}^{(i)}$, and tensor train ranks of $\tilde{\mathbf{u}}^{(i)}$ for convection matrix. LSC preconditioner is used.

We compare the two mean-based preconditioners discussed in section 5. Figure 6.4 shows the number of GMRES iterations required at each Picard step, and the associated computational costs when the two preconditioners are used. For two different mesh sizes, the PCD preconditioner results in larger numbers of GMRES iterations, and thus higher computational times, than the LSC preconditioner. It should also be noted that for both preconditioners, only a small number of GMRES iterations is needed for solving the linear system at each Picard step. This is partially due to the large stopping tolerance used in (6.4). The LSC preconditioner will be used for the numerical tests below.

In the following, we test the algorithm with several variants of the benchmark problem determined by various values of parameters associated with it. Figure 6.5a shows the solution ranks and computational times for three different values of σ . When σ is smaller, the standard deviation is smaller, the discrete solution can be approximated by a tensor train with smaller ranks, and it is also less expensive to solve the nonlinear problem. On the other hand, even for $\sigma = 0.1$, the low-rank solution takes much less storage than a full tensor. For example, the ranks of the approximate solution $\mathbf{u}^{(i)}$ are $\kappa_1 = 13$, $\kappa_2 = 83$. The ratio of storage requirements between such a tensor train and a full tensor is

$$(6.6) \quad \frac{n_t \kappa_1 + n_\xi \kappa_1 \kappa_2 + n_u \kappa_2}{n_t n_\xi n_u} = \frac{270748}{3829760} \approx 7.1\%.$$

The same quantities are plotted in Figure 6.5b for different values of the mean viscosity ν_0 . The ranks and computational times are not significantly affected by ν_0 .

Finally, the algorithm is applied to solve discrete problems with various mesh sizes h or time step sizes τ . It can be seen from Figure 6.6a that there is only a slight increase in the solution ranks as the spatial mesh is refined. It is also shown in Figure 6.6a that the computational time increases with an asymptotic rate $O(h^{-2})$ (note that a logarithmic scale is used in the figure). In other words, as the spatial

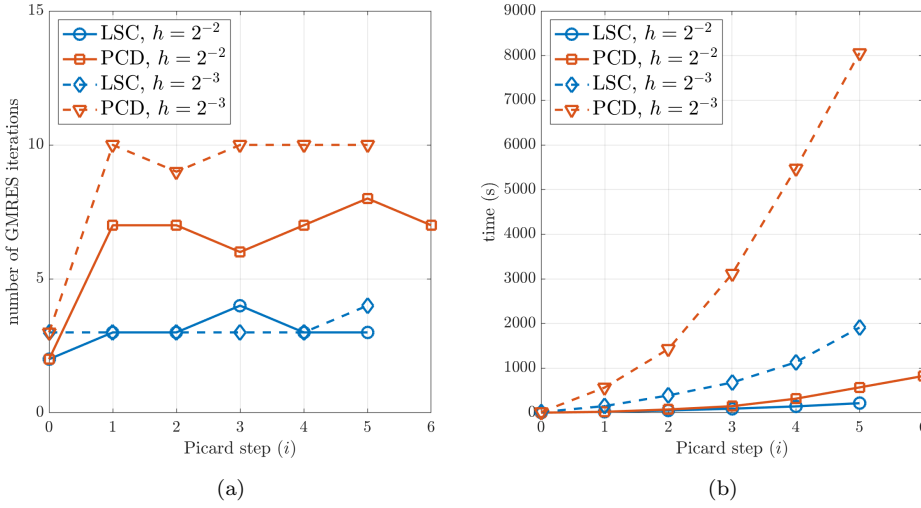


FIG. 6.4. (a) Number of GMRES iterations at each Picard step. (b) Cumulative computational time after each Picard step. For $h = 2^{-2}$ with PCD preconditioner, it takes 6 Picard steps for convergence.

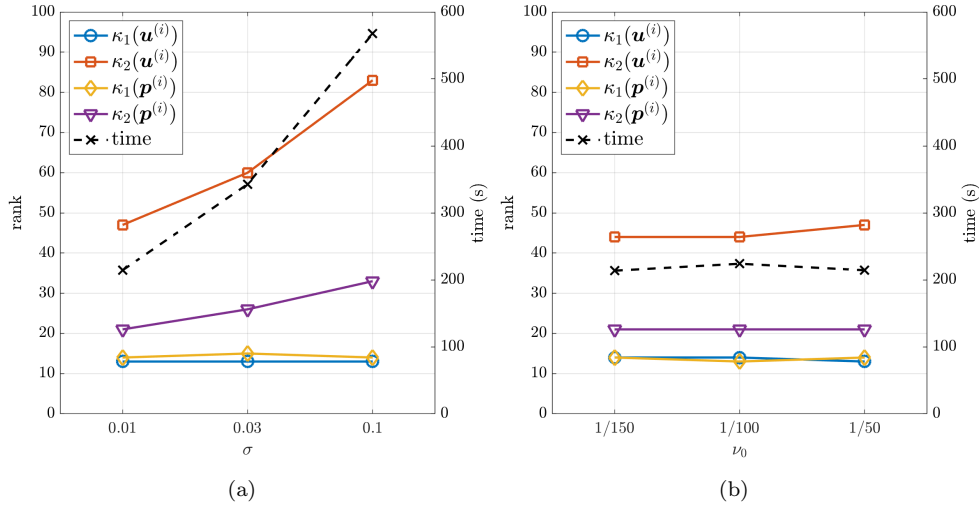


FIG. 6.5. Tensor train ranks of solutions $\mathbf{u}^{(i)}$ and $\mathbf{p}^{(i)}$ at final Picard step and computational times to compute solutions, for different values of σ and ν_0 .

mesh is refined, no extra computational burden is introduced except for the increased problem size. For different time step sizes τ , the computational time increases much more slowly than $O(\tau^{-1})$ (see Figure 6.6b). This is due to the fact that in $\mathbb{F} + \mathbb{C}$, the matrices obtained from time discretization are very simple (e.g., I_{n_t} and C_{n_t}), and thus an increase in n_t does not make a significant impact on the computational costs.

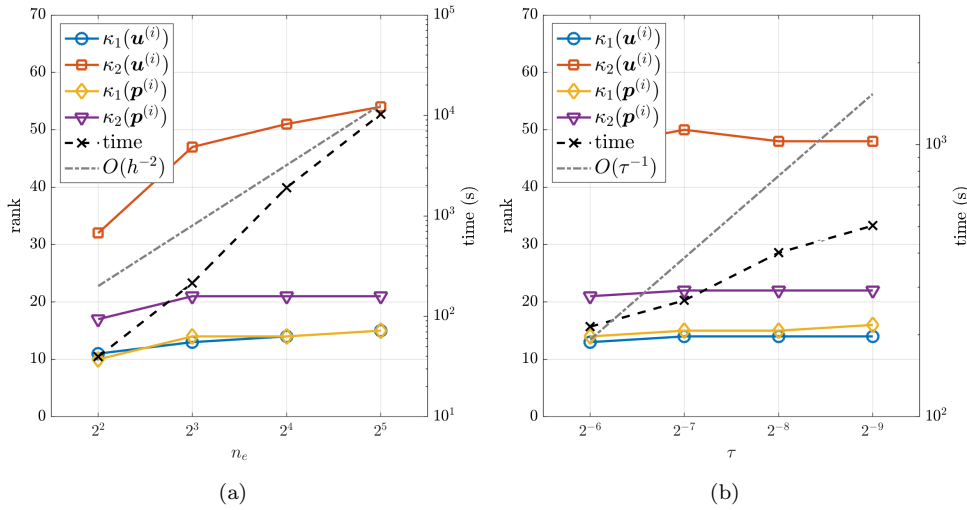


FIG. 6.6. Tensor train ranks of solutions $\mathbf{u}^{(i)}$ and $\mathbf{p}^{(i)}$ at final Picard step and computational times to compute solutions, for different values of h and τ . In (a), $n_e = 2/h$ is the number of elements in the vertical interval $[-1, 1]$ of the domain \mathcal{D} .

7. Conclusions. In this paper, we developed and studied efficient low-rank iterative methods for solving the time-dependent Navier–Stokes equations with a random viscosity. We considered an all-at-once formulation where the discrete solutions at all the time steps are solved together in a single system. To address the high storage and computational costs of this strategy, we used low-rank tensor approximations in a Newton–Krylov type algorithm. For the all-at-once system, we proposed two mean-based preconditioners using results from the deterministic problem. The computational costs were further reduced with inexact Picard method and approximate convection matrices. It was shown in the numerical experiments that the low-rank method is able to solve the nonlinear problem efficiently and the discrete solutions have small tensor ranks.

REFERENCES

- [1] R. ANDREEV AND C. TOBLER, *Multilevel preconditioning and low-rank tensor iteration for space-time simultaneous discretizations of parabolic PDEs*, Numerical Linear Algebra with Applications, 22 (2015), pp. 317–337.
- [2] J. BALLANI AND L. GRASEDYCK, *A projection method to solve linear systems in tensor format*, Numerical Linear Algebra with Applications, 20 (2013), pp. 27–43.
- [3] P. BENNER, S. DOLGOV, A. ONWUNTA, AND M. STOLL, *Solving optimal control problems governed by random Navier–Stokes equations using low-rank methods*, Mar. 2017, <https://arxiv.org/abs/1703.06097>.
- [4] P. BIRKEN, *Termination criteria for inexact fixed-point schemes*, Numerical Linear Algebra with Applications, 22 (2015), pp. 702–716.
- [5] S. V. DOLGOV, *TT-GMRES: solution to a linear system in the structured tensor format*, Russian Journal of Numerical Analysis and Mathematical Modelling, 28 (2013), pp. 149–172.
- [6] S. V. DOLGOV AND D. V. SAVOSTYANOV, *Alternating minimal energy methods for linear systems in higher dimensions*, SIAM Journal on Scientific Computing, 36 (2014), pp. A2248–A2271.
- [7] H. ELMAN, M. MIHAJLOVIĆ, AND D. SILVESTER, *Fast iterative solvers for buoyancy driven flow problems*, Journal of Computational Physics, 230 (2011), pp. 3900–3914.
- [8] H. C. ELMAN, D. J. SILVESTER, AND A. J. WATHEN, *Finite Elements and Fast Iterative Solvers*:

- With Applications in Incompressible Fluid Dynamics*, Oxford University Press, UK, second ed., 2014.
- [9] O. G. ERNST AND E. ULLMANN, *Stochastic Galerkin matrices*, SIAM Journal on Matrix Analysis and Applications, 31 (2010), pp. 1848–1872.
 - [10] M. J. GANDER AND M. NEUMÜLLER, *Analysis of a new space-time parallel multigrid algorithm for parabolic problems*, SIAM Journal on Scientific Computing, 38 (2016), pp. A2173–A2208.
 - [11] R. G. GHANEM AND P. D. SPANOS, *Stochastic Finite Elements: A Spectral Approach*, Dover Publications, New York, 2003.
 - [12] L. GRASEDYCK, D. KRESSNER, AND C. TOBLER, *A literature survey of low-rank tensor approximation techniques*, GAMM-Mitteilungen, 36 (2013), pp. 53–78.
 - [13] S. HOLTZ, T. ROHWEDDER, AND R. SCHNEIDER, *The alternating linear scheme for tensor optimization in the tensor train format*, SIAM Journal on Scientific Computing, 34 (2012), pp. A683–A713.
 - [14] D. A. KAY, P. M. GRESHO, D. F. GRIFFITHS, AND D. J. SILVESTER, *Adaptive time-stepping for incompressible flow part II: Navier–Stokes equations*, SIAM Journal on Scientific Computing, 32 (2010), pp. 111–128.
 - [15] D. KRESSNER AND C. TOBLER, *Low-rank tensor Krylov subspace methods for parametrized linear systems*, SIAM Journal on Matrix Analysis and Applications, 32 (2011), pp. 1288–1316.
 - [16] O. LE MAÎTRE AND O. M. KNIO, *Spectral Methods for Uncertainty Quantification: With Applications to Computational Fluid Dynamics*, Springer, Netherlands, 2010.
 - [17] M. LOËVE, *Probability Theory*, Van Nostrand, New York, 1960.
 - [18] Y. MADAY AND E. M. RÖNQUIST, *Parallelization in time through tensor-product space–time solvers*, Comptes Rendus Mathématique, 346 (2008), pp. 113–118.
 - [19] E. McDONALD, J. PESTANA, AND A. WATHEN, *Preconditioning and iterative solution of all-at-once systems for evolutionary partial differential equations*, SIAM Journal on Scientific Computing, 40 (2018), pp. A1012–A1033.
 - [20] E. McDONALD AND A. WATHEN, *A simple proposal for parallel computation over time of an evolutionary process with implicit time stepping*, in Numerical Mathematics and Advanced Applications ENUMATH 2015, Springer, 2016, pp. 285–293.
 - [21] I. V. OSELEDETS, *Tensor-train decomposition*, SIAM Journal on Scientific Computing, 33 (2011), pp. 2295–2317.
 - [22] I. V. OSELEDETS, S. DOLGOV, V. KAZEEV, D. SAVOSTYANOV, O. LEBEDEVA, P. ZHLOBICH, T. MACH, AND L. SONG, *TT-Toolbox*, <https://github.com/oseledets/TT-Toolbox>. Version 2.2.
 - [23] C. E. POWELL AND D. J. SILVESTER, *Preconditioning steady-state Navier–Stokes equations with random data*, SIAM Journal on Scientific Computing, 34 (2012), pp. A2482–A2506.
 - [24] Y. SAAD, *Iterative Methods for Sparse Linear Systems*, SIAM, Philadelphia, second ed., 2003.
 - [25] U. SCHOLLWÖCK, *The density-matrix renormalization group*, Reviews of Modern Physics, 77 (2005), pp. 259–315.
 - [26] D. SILVESTER, H. ELMAN, AND A. RAMAGE, *Incompressible Flow and Iterative Solver Software (IFISS)*, Sept. 2016, <https://www.manchester.ac.uk/ifiss>. Version 3.5.
 - [27] B. SOUSEDÍK AND H. C. ELMAN, *Stochastic Galerkin methods for the steady-state Navier–Stokes equations*, Journal of Computational Physics, 316 (2016), pp. 435–452.
 - [28] D. XIU AND G. E. KARNIADAKIS, *The Wiener–Askey polynomial chaos for stochastic differential equations*, SIAM Journal on Scientific Computing, 24 (2002), pp. 619–644.