Received: 3 May 2019 | Revised: 6 October 2020 | Accepted: 13 October 2020

DOI: 10.1002/nla.2349

RESEARCH ARTICLE

WILEY

Preconditioning Navier–Stokes control using multilevel sequentially semiseparable matrix computations

Yue Qiu[1] | Martin B. van Gijzen[2] | Jan-Willem van Wingerden[3] | Michel Verhaegen[3] | Cornelis Vuik[2]

[1]School of Information Science and Technology, ShanghaiTech University, Shanghai, China

[2]Delft Institute of Applied Mathematics, Delft University of Technology, Delft, The Netherlands

[3]Delft Center for Systems and Control, Delft University of Technology, Delft, The Netherlands

Correspondence
Cornelis Vuik, Numerical Analysis Group, Delft University of Technology, Van Mourik Broekmanweg 6, Delft, The Netherlands.
Email: c.vuik@tudelft.nl

Abstract

In this article, we study preconditioning techniques for the control of the Navier–Stokes equation, where the control only acts on a few parts of the domain. Optimization, discretization, and linearization of the control problem results in a generalized linear saddle-point system. The Schur complement for the generalized saddle-point system is very difficult or even impossible to approximate, which prohibits satisfactory performance of the standard block preconditioners. We apply the multilevel sequentially semiseparable (MSSS) preconditioner to the underlying system. Compared with standard block preconditioning techniques, the MSSS preconditioner computes an approximate factorization of the global generalized saddle-point matrix up to a prescribed accuracy in linear computational complexity. This in turn gives parameter independent convergence for MSSS preconditioned Krylov solvers. We use a simplified wind farm control example to illustrate the performance of the MSSS preconditioner. We also compare the performance of the MSSS preconditioner with the performance of the state-of-the-art preconditioning techniques. Our results show the superiority of the MSSS preconditioning techniques to standard block preconditioning techniques for the control of the Navier–Stokes equation.

KEYWORDS

generalized saddle-point systems, MSSS preconditioners, Navier–Stokes control

1 | INTRODUCTION

Nowadays, optimal control problems in practice are mostly solved with nonlinear programming (NLP) methods based on some discretization strategies of the original continuous problems in the functional space.[1] Once the optimization problem is discretized, the variables to be optimized are reduced to a finite-dimensional space. This results in a parameter optimization problem.[2] Simultaneous strategies, explicitly perform a discretization of the partial differential equations (PDEs) that prescribe the dynamics of the system as well as the cost function, that is, PDE simulation and the optimization procedure proceed simultaneously, cf. References 3-5. Sequential strategies, on the other hand, just parameterize the control input and employ numerical schemes in a black-box manner by utilizing the implicit function theorem (IFT).[2,6,7] The later approach turns out to be very efficient when the dimension of the control variable is much smaller than the system states which are described by the PDEs,[2,7] where the optimal shape design,[8] boundary control of fluid dynamics[9,10] are applications for this type of optimization approaches. For the simultaneous approach, its high potential in terms of efficiency and robustness turns out to be very difficult to be realized when the sizes of the states and inputs are both very large. This yields a very large-scale optimization problem, where the equality constraints by PDEs are appended to the augmented cost function. The Lagrangian multipliers, the number of which is the same as the state variables of PDEs, make the size of the optimization problem even bigger.

Just like many problems in science and engineering, the most time-consuming part for the optimal control of PDEs is to solve a series of linear systems arising from the PDE discretization.[3] With increasing improvement of computational resources and the advancement of numerical techniques, larger problems can be taken into consideration.[11] An important building block for the optimal control of PDEs is the preconditioning techniques to accelerate the simulation of PDEs. Many efforts have been dedicated to the development of efficient and robust preconditioning techniques for the distributed control problems where the control is distributed throughout the domain,[12-21] or the boundary control problems where only boundary conditions can be regulated.[22-24] For the in-domain control problems where the control only acts on a few parts of the domain, preconditioning techniques developed for the distributed control problems do not give satisfactory performance. This is because the developed preconditioning techniques highly depend on an efficient approximation of the Schur complement of the block linear system arising from the discretized optimality condition.[20,25] Research on in-domain control problems in engineering focuses on specific objective[1,22,26] without considering efficient preconditioning techniques.

In this article, we focus on efficient and robust preconditioning techniques for the in-domain control of the Navier–Stokes equation, which is to the best of our knowledge the first devoted effort. We use a simplified wind farm control example to illustrate the performance of our preconditioning technique, that is, the so-called multilevel sequentially semiseparable (MSSS) preconditioner. The MSSS preconditioner is proposed for PDE-constrained optimization problems[27] and later studied for some benchmark problems of computational fluid dynamics (CFD).[28] Applying the implicit function theorem (IFT) at the optimization step, we can reuse the preconditioners for the linearized Navier–Stokes equation to solve the forward sensitivity equations for the computations of the gradient and Hessian matrix. This reduces the computational cost significantly. We then study the MSSS preconditioner for the resulting generalized saddle-point system from optimization. In contrast to the standard block preconditioners that require to approximate the Schur complement of the block linear system, the MSSS preconditioner computes an approximate factorization of the global system matrix up to a prescribed accuracy in linear computational complexity. This is a big advantage over the standard block preconditioners. We evaluate the performance of the MSSS preconditioner using incompressible flow and fast iterative solver software (IFISS)[29,30][1] and compare with the state-of-the-art preconditioning techniques.

The advantage of MSSS matrix computations is their simplicity and low computational cost, which is $\mathcal{O}(r_k^3 N)$. Here, N is the matrix size, r_k is the rank of the off-diagonal blocks which is usually much smaller than N.[31,32] Related structured matrices include hierarchical matrices ($\mathcal{H}$-matrix),[33] $\mathcal{H}^2$-matrices,[34,35] hierarchically semiseparable (HSS) matrices,[36,37] with computational complexity $\mathcal{O}(N\log^{\alpha} N)$ for some moderate α. $\mathcal{H}$-matrices originate from approximating the kernel of integral equations and have been extended to elliptic PDEs.[38,39] $\mathcal{H}^2$-matrices and HSS matrices are specific subsets of $\mathcal{H}$-matrices and HSS matrices have been successfully applied in multifrontal solvers.[37] Efforts have been devoted to preconditioning symmetric positive definite systems by exploiting the HSS matrix structure[40] and unsymmetric systems by $\mathcal{H}$-matrix computations.[39,41] To have a comprehensive summary of different hierarchical structures and techniques, as well as their applicability and limitations, we refer to.[42] For interested readers to gain more insight on the latest development in hierarchical matrices by leveraging machine learning techniques, we recommend.[43] To keep a clear structure of this article, we only focus on the MSSS preconditioning techniques. For related work on exploiting the low-rank property of the subblocks in the hierarchical factorization of the system matrix for preconditioning or direct solution, we refer to.[44-48]

The structure of this article is organized as follows. In Section 2, we use a simple wind farm control example to formulate an in-domain Navier–Stokes control problem. Applying numerical approaches to solve this optimization problem, we obtain a generalized saddle-point system in Section 3. To preconditioning this generalized saddle-point system, we study the MSSS preconditioning technique in Section 4. In Section 5, we perform numerical experiments to illustrate

[1]IFISS is a computational laboratory for experimenting with state-of-the-art preconditioned iterative solvers for the discrete linear equation systems that arise in incompressible flow modeling

the performance of the MSSS preconditioning techniques for this type of problems. We also study the state-of-the-art preconditioning techniques in Section 5 as a comparison. Conclusions are drawn in Section 6.

2 | PROBLEM FORMULATION

In this section, we use wind farm control as an example to formulate the in-domain Navier–Stokes control problem. Many research activities illustrate that operating all the turbines in a wind farm at their own optimal state gives suboptimal performance of the overall wind farm.[49] Wind farm control aims to optimize the total power captured from the wind by taking coordinating control strategies to the turbines in the wind farm. By appropriately choosing the computational domain for the flow, the wind farm control can be formulated as an optimal flow control problem where the dynamics of the flow are described by the incompressible Navier–Stokes equation. For the wind farm control, the control only acts on a few parts of the domain where the turbines are located. This in turn gives an in-domain flow control problem. In the next part, we aim to build a simple wind farm model and use this model to formulate the in-domain Navier–Stokes control problem.

2.1 | Fluid dynamics

Some efforts have been devoted to develop a suitable model to simulate the wake effect in the wind farm, cf. References 50,51 for a general survey and[49] for recent developments. In general there exist two approaches for modeling of the wake. One is the heuristic approach that does not solve a flow equation but uses some rules of thumb,[49] a typical example is the Park model.[52] The second approach is solving an incompressible Navier–Stokes or Euler equation, cf. References 11. In this article, we use the second approach to model the flow in the wind farm. Moreover, some recent research try to take the boundary layer and some physical behavior into account. This in turn requires a more complicated turbulent flow model for the wind farm simulation study.[11,51,53] However, these research activities do not take efficient preconditioning techniques into account but focus on physical properties of the flow for the wind farm simulation. In this article, we focus on designing efficient and robust preconditioning techniques and we evaluate the performance of our preconditioning techniques by IFISS. We only consider flow problems that can be addressed within the framework of IFISS.

Consider the stationary incompressible Navier–Stokes equation in $\Omega \in \mathbb{R}^2$ that is given by

$$\begin{aligned} -\nu\Delta\mathbf{u} + (\mathbf{u}\cdot\nabla)\mathbf{u} + \nabla p &= \mathbf{f}, \\ \nabla\cdot\mathbf{u} &= 0, \end{aligned} \tag{1}$$

where ν is the kinematic viscosity, $\mathbf{u}$ is the velocity field, p denotes the pressure, and $\mathbf{f}$ is a source term. Here Ω is a bounded domain with its boundary given by $\Gamma = \partial\Omega = \partial\Omega_D \cup \partial\Omega_N$, where $\partial\Omega_D$ denotes the boundary where Dirichlet boundary conditions are prescribed and $\partial\Omega_N$ represents the boundary where Neumann boundary conditions are imposed. The Reynolds number $Re \in \mathbb{R}^+$ describes the ratio of the inertial and viscous forces within the fluid[54] and is defined by

$$Re \triangleq \frac{u_r L_r}{\nu},$$

where $u_r \in \mathbb{R}^+$ is the reference velocity, $L_r \in \mathbb{R}^+$ is the reference distance that the flow travels. The Reynolds number plays an important role in the flow equation that describes whether the flow under consideration is laminar or turbulent. In many engineering problems, turbulent flow happens when the Reynolds number $Re > 2000$.[54] In the case of flow through a straight pipe with a circular cross-section, at a Reynolds number below a critical value of approximately 2040, fluid motion will ultimately be laminar, whereas at larger Reynolds numbers, the flow can be turbulent.[55] Since we focus on efficient preconditioning techniques for the in-domain flow control using IFISS in this article and no turbulent flow model is included in IFISS, we consider a flow with Reynolds number $Re = 2000$, although this does not correspond to practical flow for wind farm control.

To study the aerodynamics of the wind farm, we cannot set an infinite dimensional computational domain. We can prescribe suitable boundary conditions for the flow that in turn gives a finite domain. We set a standard reference domain $\Omega = [-1, 1] \times [-1, 1]$ for the wind farm simulation study. The reference velocity u_r is set to 1.

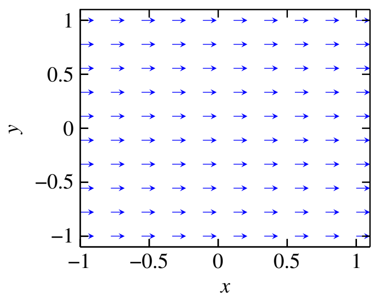


FIGURE 1 Resolved velocity without outer source

The turbines in the wind farm are located in a line that follows the stream direction, and the leftmost turbine is placed at the center of the domain. Such configurations are widely used in the wind farm simulation studies.[11,52] Here the diameter of the turbine is set to be $\frac{1}{64}$ of the reference computational domain. The distance between turbines is set to be five times of the diameter of the turbines.[56] Constant inflow is imposed on the left boundary and is given by

$$u_x = u_c, \quad u_y = 0,$$

or equivalently

$$\mathbf{u} = -u_c\mathbf{n}, \tag{2}$$

where $\mathbf{n}$ is the unit normal vector of the boundary that points outwards. For top, right, and bottom boundary that are far away from the turbines, the flow is considered to be free stream and the zero stress boundary condition for outflow given by (3) is prescribed,

$$\nu\frac{\partial \mathbf{u}}{\partial \mathbf{n}} - p\mathbf{n} = 0. \tag{3}$$

Here, $\frac{\partial \mathbf{u}}{\partial \mathbf{n}}$ is the Gâteaux derivative at $\partial\Omega_N$ in the direction of $\mathbf{n}$. This boundary condition states that the flow can move freely across the boundary by resolving the Navier–Stokes equation (1).

Associated with the prescribed boundary condition (2)–(3), the resolved velocity without outer source for the Navier–Stokes equation (1) is shown in Figure 1.

2.2 | Wind turbines

Wind turbines can be modeled as outer sources that interact with the flow. Two widely used methods for wind turbine modeling are the actuator disk method (ADM)[57] and the actuator line method.[58] In this article, we use the ADM method for wind turbines, where the thrust force is denoted by,

$$f = -C_T\rho A\hat{u}_d^2. \tag{4}$$

Here $\hat{u}_d$ is the average axial flow velocity at the turbine disk, C_T is the disk based thrust coefficient, ρ is the density of the flow, and A is the area that is swept by the turbine blades.

2.3 | Objective function

The wind farm control aims to maximize the total power captured by all the wind turbines in the wind farm, which can be represented as

$$P_t = \sum_{j=1}^{N_t} -f_j\hat{u}_j = \rho A\sum_{j=1}^{N_t} C_{T_j}\hat{u}_j^3,$$

where N_t is the number of turbines in the wind farm and $\hat{u}_j$ is the uniform disk averaged axial flow speed of the jth turbine.

To summarize, the in-domain Navier–Stokes control problem can be formulated as the following optimization problem,

$$
\begin{aligned}
\min_{C_T,\mathbf{u}} \quad & -\sum_{j=1}^{N_t} C_{T_j}\hat{u}_j^3 \\
\text{s.t.} \quad & -\nu\Delta\mathbf{u} + (\mathbf{u}\cdot\nabla)\mathbf{u} + \nabla p = \mathbf{f}(C_T,\mathbf{u}), \\
& \nabla\cdot\mathbf{u} = 0, \\
& 0 \le C_{T_j} \le 1, \quad (j = 1, \dots, N_t).
\end{aligned} \tag{5}
$$

Here C_{T_j} varies from 0 to 1 based on individual pitch control and yaw control of the turbine j, $\mathbf{f}(C_T,\mathbf{u})$ is a nonlinear function and it is of value (4) at the position where turbines are placed and 0 elsewhere, and $C_T = \begin{bmatrix} C_{T_1} & C_{T_2} & \dots & C_{T_{N_t}} \end{bmatrix}^T$.

3 | REDUCED NONLINEAR PROGRAMMING

In the previous section, we formulated an in-domain control problem (5) by using a wind farm control example. The size of the control variable N_t, which is the number of turbines, is much smaller than the size of the state variables (number of velocity and pressure grid unknowns). Therefore, we use the sequential approach that is based on the implicit function theorem to solve a reduced optimization problem.

3.1 | Reduced optimization problem

Denote the equality constraints in (5) for the flow equation by $h(C_T,\phi) = 0$ where $\phi = (\mathbf{u}, p)$, and the objective function by $g(C_T,\phi)$, then the optimization problem (5) can be written as

$$
\begin{aligned}
\min_{C_T,\phi} \quad & g(C_T,\phi) \\
\text{s.t.} \quad & h(C_T,\phi) = 0, \\
& 0 \le C_T \le 1.
\end{aligned} \tag{6}
$$

The equality constraint in (5) implies that the velocity and pressure ϕ is a function of $\mathbf{f}$. For the existence of this function, cf. References 7 for a proof. Since $\mathbf{f}$ is an explicit function of C_T, ϕ is a function of C_T and denote this function by $\phi = s(C_T)$. The function $s(C_T)$ is not explicitly computed but is obtained implicitly by solving the flow equation (1) using given C_T.

By using the implicit function theorem, we can rewrite the optimization problem in a reduced form,

$$
\begin{aligned}
\min_{C_T} \quad & g(C_T, s(C_T)) \\
\text{s.t.} \quad & 0 \le C_T \le 1.
\end{aligned} \tag{7}
$$

Newton-type methods, which are second-order methods, are well-suited to solve this type of nonlinear programming (NLP) problems. An alternative approach to solve this type of problem is the (reduced) sequentially quadratic programming ((R)SQP).[59] For the reduced optimization problem (7), these two types of methods are quite similar and we refer to[7] for a detailed discussion.

In this section, we apply Newton's method to solve the reduced NLP problem (7). The reason to choose the Newton's method is that the Hessian matrix for this problem is of small size and can be computed explicitly. Moreover, we can reuse the information from the last Newton step of solving the nonlinear flow equation to compute the gradient and the Hessian matrix, which makes Newton's method computationally competitive for this optimization problem. This will be explained in the following part. The Newton's method for this problem is described by Algorithm 1.

Algorithm 1. Reduced Newton's algorithm for (6)

```
1:
Input:    Initial guess $C_T^{(0)}$, maximal optimization steps $\mathtt{it}_{\max}$, stop tolerance $\varepsilon_0$, $k = 0$
2:  while $\|\nabla g_k\| > \varepsilon_0$ ςς $k \le \mathtt{it}_{\max}$ do                          ▷ optimization loop
3:      solve $h(C_T^{(k)}, \phi) = 0$ to compute $\phi^{(k)}$                               ▷ cf. Section 3.2
4:      compute the gradient $\nabla g_k$ at $(C_T^{(k)}, \phi^{(k)})$                        ▷ cf. Section 3.3
5:      compute the Hessian matrix $H_k$ at $(C_T^{(k)}, \phi^{(k)})$                         ▷ cf. Section 3.3
6:      compute the update $\Delta C_T^{(k)} = \arg\min \Delta C_T^T H_k \Delta C_T + \nabla g_k^T \Delta C_T$
7:      $C_T^{(k+1)} \leftarrow C_T^{(k)} + \Delta C_T^{(k)}$
8:      Check inequality constraints by projection
9:      if $C_{T_j} > 1$ then
10:         $C_{T_j} = 1$                                                              ▷ project on boundary
11:     else if $C_{T_j} < 0$ then
12:         $C_{T_j} = 0$                                                              ▷ project on boundary
13:     end if
14:     $k \leftarrow k + 1$
15: end while
16:
Output:    Optimal control variable $C_T$ and corresponding solution of $u$
```

In Algorithm 1, we denote the optimization step as the outer iteration, and at each outer iteration, we need to solve a Navier–Stokes equation with nonlinear right-hand side. This step is denoted by the inner iteration in line 3 of the algorithm. From the description of Algorithm 1, it is clear that the most time-consuming part for this optimization problem is the solution of the nonlinear flow equation and the computations of the gradient and Hessian matrix. Therefore, efficient numerical methods need to be developed.

We should emphasize that Algorithm 1 has the advantage when the dimension of the control is much smaller than the dimension of the system states. For such cases, the work to solve the forward sensitivity equation to compute the gradient and the Hessian matrix at lines 4 and 5 of the Algorithm 1 is not big since the system matrices for such linear systems in lines 4 and 5 are the same with the system matrix of the last nonlinear step to solve the Navier–Stokes equation in line 3. Once we have efficient preconditioning technique to solve the generalized saddle-point system resulting from the Navier–Stokes equation with nonlinear right-hand side in line 3, we can also compute the gradient and Hessian matrix easily in lines 4 and 5 of Algorithm 1. For the case when the dimension of the control variables is comparable with the dimension of the systems states, we refer to[7,60] for the adjoint equation approach.

We would like to point out here that for the time-dependent in-domain control problems, our preconditioning technique also fits for such cases when we apply the widely used adjoint equation approach to compute the gradient and the Hessian matrix. For the adjoint equation approach, we need to solve the Navier–Stokes equation forward in time and the adjoint equation backward in time. At each time step to compute the solution of the forward and the adjoint PDE, we need to solve a linear equation with generalized saddle-point system structure. The MSSS preconditioning technique fits well for such a framework.

3.2 | Computations of the flow equation

At each outer iteration of Algorithm 1, we need to solve a nonlinear flow equation at line 3 of Algorithm 1. We explicitly write this equation in decomposed form as

$$-\nu\left(\frac{\partial^2}{\partial x^2} + \frac{\partial^2}{\partial y^2}\right)u_x + \left(u_x\frac{\partial}{\partial x} + u_y\frac{\partial}{\partial y}\right)u_x + \frac{\partial}{\partial x}p = f_x(C_T, u_x, u_y),$$

$$-\nu\left(\frac{\partial^2}{\partial x^2}+\frac{\partial^2}{\partial y^2}\right)u_y+\left(u_x\frac{\partial}{\partial x}+u_y\frac{\partial}{\partial y}\right)u_y+\frac{\partial}{\partial y}p=f_y(C_T,u_x,u_y),$$
$$\frac{\partial}{\partial x}u_x+\frac{\partial}{\partial y}u_y=0. \tag{8}$$

Equation (8) is a nonlinear equation where the nonlinearity is caused by both the velocity convection operator and the nonlinear right-hand side. To solve such a nonlinear equation (8), we apply the Newton's method. At each Newton iteration of step 3 in Algorithm 1, we need to solve a linear system of the following type,

$$\begin{bmatrix}\nu K+N+J_{xx}^n+J_{xx}^f & J_{xy}^n+J_{xy}^f & B_x^T\\ J_{yx}^n+J_{yx}^f & \nu K+N+J_{yy}^n+J_{yy}^f & B_y^T\\ B_x & B_y & 0\end{bmatrix}\begin{bmatrix}\Delta u_x\\ \Delta u_y\\ \Delta p\end{bmatrix}=\begin{bmatrix}a\\ b\\ c\end{bmatrix}, \tag{9}$$

after discretizing the nonlinear partial differential equation (8) using mixed finite element method. Here N is the convection matrix, $J_{(\cdot)}^n$ denote the Jacobian matrices from the nonlinear velocity convection term, and $J_{(\cdot)}^f$ represent the Jacobian matrices from the nonlinear right-hand side. Since only a small part of the domain is controlled, f_x and f_y are zero almost everywhere in the domain except in the vicinity where the turbines are placed. This in turn gives singular Jacobian matrices $J_{(\cdot)}^f$.

Comparing system (9) with the standard linearized Navier–Stokes equation by the Newton's method given by

$$\begin{bmatrix}\nu K+N+J_{xx}^n & J_{xy}^n & B_x^T\\ J_{yx}^n & \nu K+N+J_{yy}^n & B_y^T\\ B_x & B_y & 0\end{bmatrix}\begin{bmatrix}\Delta u_x\\ \Delta u_y\\ \Delta p\end{bmatrix}=\begin{bmatrix}a\\ b\\ c\end{bmatrix}, \tag{10}$$

we see that the linearized flow equation (9) is a perturbed linearized Navier–Stokes equation with singular perturbation in the matrix blocks that correspond to the velocity.

Rewrite Equation (10) in a compact form as

$$\begin{bmatrix}A & B^T\\ B & 0\end{bmatrix}\begin{bmatrix}\Delta u\\ \Delta p\end{bmatrix}=\begin{bmatrix}f\\ g\end{bmatrix}, \tag{11}$$

and Equation (9) is given by a perturbed form

$$\begin{bmatrix}A+J^f & B^T\\ B & 0\end{bmatrix}\begin{bmatrix}\Delta u\\ \Delta p\end{bmatrix}=\begin{bmatrix}f\\ g\end{bmatrix}. \tag{12}$$

The linear system (12) is large, sparse, and highly indefinite. It belongs to the type of generalized saddle-point system.[61] Efficient preconditioning techniques are needed to solve such systems using Krylov methods.

The standard approach for preconditioning (12) are given by the following block preconditioners,

$$P_1=\begin{bmatrix}A+J^f & \\ & -S\end{bmatrix},\quad P_2=\begin{bmatrix}A+J^f & \\ B & S\end{bmatrix},$$

where $S=-B(A+J^f)^{-1}B^T$ is the Schur complement. It is shown in Reference 61 that the preconditioned spectrum by using P_1 has three distinct eigenvalues and one eigenvalue using P_2 with minimal polynomial of degree 2. Therefore, Krylov methods such as GMRES or IDR(s) compute the solution in at most three steps by using P_1 and two steps with P_2. The key in applying the block preconditioners P_1 or P_2 lies in an efficient approximation of the Schur complement $\tilde{S}\approx S$ and

$$|\lambda(\tilde{S}^{-1}S)|\leq\gamma,$$

where γ is a constant that is independent of the mesh sizes and Reynolds number.

In the past decades, efforts for efficient approximation of the Schur complement of the Navier–Stokes equation, that is, $\hat{S} \approx S_{ns} = -BA^{-1}B^T$ have been made, which have resulted in the SIMPLE method,[62] the augmented Lagrangian preconditioner,[63] the pressure convection-diffusion (PCD) preconditioner[64] among others. However, the aforementioned preconditioners do not perform well to solve the perturbed linear system (12) due to the presence of the perturbed term J^f. The matrix J^f comes from the discretization and linearization of the nonlinear source term f given by (4) and is singular since f only acts on part of the computational domain. In general, the singular matrix J^f does not have a low-rank factorized form, which makes the Schur complement even more difficult to approximate. Numerical experiments in Section 5 illustrate the performance of block preconditioner P_2 for (12) where we approximate the Schur complement $S = -B(A + J^f)^{-1}B^T$ using the PCD preconditioner, that is, $S_{PCD} \approx -BA^{-1}B^T \approx -B(A + J^f)^{-1}B^T$.

As we will see in Section 4, all the blocks of the matrix in (9) have an MSSS structure, it is therefore natural to permute the matrix with MSSS blocks into a global MSSS matrix. With this permutation, we can compute an approximate factorization. This factorization gives a global MSSS preconditioner for the system (9). This global preconditioner does not require to approximate the Schur complement of the generalized saddle-point system (12). Details of this preconditioning technique will be introduced in Section 4.

3.3 | Computations of partial derivatives

Denote the reduced gradient of the cost function in (7) by

$$\nabla g = \begin{bmatrix} \frac{\partial}{\partial C_{T_1}} g & \frac{\partial}{\partial C_{T_2}} g & \cdots & \frac{\partial}{\partial C_{T_{N_t}}} g \end{bmatrix}^T.$$

The gradient can be computed using

$$\frac{\partial}{\partial C_{T_j}} g = \frac{\partial g}{\partial C_{T_j}} + \frac{\partial g}{\partial u_x}\frac{\partial u_x}{\partial C_{T_j}} + \frac{\partial g}{\partial u_y}\frac{\partial u_y}{\partial C_{T_j}}, \quad (j = 1, 2, \dots, N_t).$$

Since the cost function g is an analytic function of C_T, u_x, and u_y, the partial derivatives $\frac{\partial g}{\partial C_{T_j}}$, $\frac{\partial g}{\partial u_x}$, and $\frac{\partial g}{\partial u_y}$ are trivial to compute. Next, we show how to compute the partial derivatives $\frac{\partial u_x}{\partial C_{T_j}}$ and $\frac{\partial u_y}{\partial C_{T_j}}$.

Assume that u_x, u_y, and p are twice differentiable with respect to $C_{T_j} (j = 1, 2, \dots, N_t)$, and that the first- and second-order derivatives have continuous second-order derivative in Ω, that is,

$$\frac{\partial u_x}{\partial C_{T_j}}, \quad \frac{\partial u_y}{\partial C_{T_j}}, \quad \frac{\partial p}{\partial C_{T_j}} \in C^2(\Omega),$$

and

$$\frac{\partial^2 u_x}{\partial C_{T_i} \partial C_{T_j}}, \quad \frac{\partial^2 u_y}{\partial C_{T_i} \partial C_{T_j}}, \quad \frac{\partial^2 p}{\partial C_{T_i} \partial C_{T_j}} \in C^2(\Omega),$$

for $(i, j = 1, 2, \dots, N_t)$. Then we can compute the first-order derivative by solving

$$-\nu \overbrace{\left(\frac{\partial^2}{\partial x^2} + \frac{\partial^2}{\partial y^2}\right)}^{\text{diffusion operator}} \left(\frac{\partial u_x}{\partial C_{T_j}}\right) + \overbrace{\left(u_x \frac{\partial}{\partial x} + u_y \frac{\partial}{\partial y}\right)}^{\text{convection operator}} \left(\frac{\partial u_x}{\partial C_{T_j}}\right)$$
$$+ \overbrace{\left(\frac{\partial u_x}{\partial x}\frac{\partial u_x}{\partial C_{T_j}} + \frac{\partial u_x}{\partial y}\frac{\partial u_y}{\partial C_{T_j}}\right)}^{\text{linear term}}$$

$$
\begin{aligned}
&+\frac{\partial}{\partial x}\left(\frac{\partial p}{\partial C_{T_j}}\right)=\frac{\partial}{\partial C_{T_j}}f_x(C_T,u_x,u_y),\\
&-\underbrace{\nu\left(\frac{\partial^2}{\partial x^2}+\frac{\partial^2}{\partial y^2}\right)}_{\text{diffusion operator}}\left(\frac{\partial u_y}{\partial C_{T_j}}\right)+\underbrace{\left(u_x\frac{\partial}{\partial x}+u_y\frac{\partial}{\partial y}\right)}_{\text{convection operator}}\left(\frac{\partial u_y}{\partial C_{T_j}}\right)+\underbrace{\left(\frac{\partial u_y}{\partial x}\frac{\partial u_x}{\partial C_{T_j}}+\frac{\partial u_y}{\partial y}\frac{\partial u_y}{\partial C_{T_j}}\right)}_{\text{linear term}}\\
&+\frac{\partial}{\partial y}\left(\frac{\partial p}{\partial C_{T_j}}\right)=\frac{\partial}{\partial C_{T_j}}f_y(C_T,u_x,u_y),\\
&\frac{\partial}{\partial x}\left(\frac{\partial u_x}{\partial C_{T_j}}\right)+\frac{\partial}{\partial y}\left(\frac{\partial u_y}{\partial C_{T_j}}\right)=0.
\end{aligned}
\tag{13}
$$

Here the partial differential equation (13) is obtained by computing the first-order derivative with respect to $C_{T_j}(j = 1, 2, \dots, N_t)$ for the flow equation (8) and making use of the linearity of the diffusion operator and the divergence operator.

Note that Equation (13) is a linear partial differential equation. It is often called the forward sensitivity equation of (8).[65] Homogeneous Dirichlet boundary conditions are imposed on the constant inflow boundary and natural boundary conditions are prescribed for the outflow boundary for the sensitivity equation (13). We use a mixed-finite element method to discretize (13) and use the following notations

$$\chi_j=\left[\left(\frac{\partial u_x}{\partial C_{T_j}}\right)^T \quad \left(\frac{\partial u_y}{\partial C_{T_j}}\right)^T \quad \left(\frac{\partial p}{\partial C_{T_j}}\right)^T\right]^T.$$

We reuse the same notation to denote its discrete analog to get a linear system given by

$$\underbrace{\begin{bmatrix} \nu K+N_l+J_{xx}^{n_l}+J_{xx}^{f} & J_{xy}^{n_l}+J_{xy}^{f} & B_x^T\\ J_{yx}^{n_l}+J_{yx}^{f} & \nu K+N_l+J_{yy}^{n_l}+J_{yy}^{f} & B_y^T\\ B_x & B_y & 0\end{bmatrix}}_{\mathcal{A}}\chi_j=\zeta_j, \tag{14}$$

where N_l is the convection matrix, $J_{xx}^{n_l}, J_{xy}^{n_l}, J_{yx}^{n_l}$, and $J_{yy}^{n_l}$ are the corresponding Jacobian matrices by linearizing the nonlinear velocity convection operator, respectively. J_{xx}^{f}, J_{xy}^{f}, J_{yx}^{f}, and J_{yy}^{f} are the associating Jacobian matrices that linearize the nonlinear right-hand side f with respect to C_{T_j}. It is easy to verify that the matrix $\mathcal{A}$ in (14) is just the same system matrix from the last Newton step of the linearized flow equation (9). The right-hand side vector ζ_j is obtained by discretizing known variables in (13). Therefore, the computed preconditioner for the last Newton step to solve the flow equation can be reused, and the computational work is much reduced.

By stacking χ_j, and ζ_j $(j = 1, 2, \dots, N_t)$ as

$$\chi=\begin{bmatrix}\chi_1 & \chi_2 & \dots & \chi_{N_t}\end{bmatrix}, \quad \text{and} \quad \zeta=\begin{bmatrix}\zeta_1 & \zeta_2 & \dots & \zeta_{N_t}\end{bmatrix},$$

we obtain the following linear equations with multiple left-hand sides and multiple right-hand sides

$$\mathcal{A}\chi=\zeta. \tag{15}$$

Here the size of unknowns N_t, is much smaller than the problem size. Block Krylov subspace methods, such as block IDR(s),[66,67] are well-suited to solve all the unknowns simultaneously for this type of problems.

Next, we can use similar approaches as above for gradient computations to compute the Hessian matrix H, which is given by

$$H = \begin{bmatrix} \frac{\partial^2}{\partial C_{T_1}^2} g & \cdots & \cdots & \frac{\partial^2}{\partial C_{T_1} \partial C_{T_{N_t}}} g \\ \vdots & \ddots & & \vdots \\ \vdots & & \ddots & \vdots \\ \frac{\partial^2}{\partial C_{T_{N_t}} \partial C_{T_1}} g & \cdots & \cdots & \frac{\partial^2}{\partial C_{T_{N_t}}^2} g \end{bmatrix}.$$

According to second equation in section 3.3, we have

$$\begin{aligned} \frac{\partial^2}{\partial C_{T_j} \partial C_{T_k}} g = & \frac{\partial^2 g}{\partial C_{T_j} \partial C_{T_k}} + \frac{\partial^2 g}{\partial u_x^2} \frac{\partial u_x}{\partial C_{T_k}} \frac{\partial u_x}{\partial C_{T_j}} + \frac{\partial g}{\partial u_x} \frac{\partial^2 u_x}{\partial C_{T_j} \partial C_{T_k}} \\ & + \frac{\partial^2 g}{\partial u_y^2} \frac{\partial u_y}{\partial C_{T_k}} \frac{\partial u_y}{\partial C_{T_j}} + \frac{\partial g}{\partial u_y} \frac{\partial^2 u_y}{\partial C_{T_j} \partial C_{T_k}}. \end{aligned} \tag{16}$$

Since g is an analytic function of C_T, u_x, u_y and we have already computed the first-order derivative $\frac{\partial u_x}{\partial C_{T_i}}$, and $\frac{\partial u_y}{\partial C_{T_i}}$ by solving the sensitivity equation (13). Here we just need to compute the second-order derivatives $\frac{\partial^2 u_x}{\partial C_{T_j} \partial C_{T_k}}$, and $\frac{\partial^2 u_y}{\partial C_{T_j} \partial C_{T_k}}$ for the computations of the Hessian matrix. We can apply the same technique to the sensitivity equation (13), and we get

$$\begin{aligned} & -\overbrace{\nu\left(\frac{\partial^2}{\partial x^2} + \frac{\partial^2}{\partial y^2}\right)}^{\text{diffusion operator}} \left(\frac{\partial^2 u_x}{\partial C_{T_j} \partial C_{T_k}}\right) + \overbrace{\left(u_x \frac{\partial}{\partial x} + u_y \frac{\partial}{\partial y}\right)}^{\text{convection operator}} \left(\frac{\partial^2 u_x}{\partial C_{T_j} \partial C_{T_k}}\right) \\ & + \overbrace{\left(\frac{\partial u_x}{\partial x} \frac{\partial^2 u_x}{\partial C_{T_j} \partial C_{T_k}} + \frac{\partial u_x}{\partial y} \frac{\partial^2 u_y}{\partial C_{T_j} \partial C_{T_k}}\right)}^{\text{linear term}} + \frac{\partial}{\partial x}\left(\frac{\partial^2 p}{\partial C_{T_j} \partial C_{T_k}}\right) \\ & + \overbrace{\left(\frac{\partial u_x}{\partial C_{T_k}} \frac{\partial}{\partial x}\left(\frac{\partial u_x}{\partial C_{T_j}}\right) + \frac{\partial u_y}{\partial C_{T_k}} \frac{\partial}{\partial y}\left(\frac{\partial u_x}{\partial C_{T_j}}\right)\right)}^{\text{known}} \\ & + \overbrace{\left(\frac{\partial u_x}{\partial C_{T_j}} \frac{\partial}{\partial x}\left(\frac{\partial u_x}{\partial C_{T_k}}\right) + \frac{\partial u_y}{\partial C_{T_j}} \frac{\partial}{\partial y}\left(\frac{\partial u_x}{\partial C_{T_k}}\right)\right)}^{\text{known}} = \frac{\partial^2}{\partial C_{T_j} \partial C_{T_k}} f_x(C_T, u_x, u_y), \\ & -\underbrace{\nu\left(\frac{\partial^2}{\partial x^2} + \frac{\partial^2}{\partial y^2}\right)}_{\text{diffusion operator}} \left(\frac{\partial^2 u_y}{\partial C_{T_j} \partial C_{T_k}}\right) + \underbrace{\left(u_x \frac{\partial}{\partial x} + u_y \frac{\partial}{\partial y}\right)}_{\text{convection operator}} \left(\frac{\partial^2 u_y}{\partial C_{T_j} \partial C_{T_k}}\right) \\ & + \underbrace{\left(\frac{\partial u_y}{\partial x} \frac{\partial^2 u_x}{\partial C_{T_j} \partial C_{T_k}} + \frac{\partial u_y}{\partial y} \frac{\partial^2 u_y}{\partial C_{T_j} \partial C_{T_k}}\right)}_{\text{linear term}} + \frac{\partial}{\partial x}\left(\frac{\partial^2 p}{\partial C_{T_j} \partial C_{T_k}}\right) \\ & + \underbrace{\left(\frac{\partial u_x}{\partial C_{T_k}} \frac{\partial}{\partial x}\left(\frac{\partial u_y}{\partial C_{T_j}}\right) + \frac{\partial u_y}{\partial C_{T_k}} \frac{\partial}{\partial y}\left(\frac{\partial u_y}{\partial C_{T_j}}\right)\right)}_{\text{known}} \\ & + \underbrace{\left(\frac{\partial u_x}{\partial C_{T_j}} \frac{\partial}{\partial x}\left(\frac{\partial u_y}{\partial C_{T_k}}\right) + \frac{\partial u_y}{\partial C_{T_j}} \frac{\partial}{\partial y}\left(\frac{\partial u_y}{\partial C_{T_k}}\right)\right)}_{\text{known}} = \frac{\partial^2}{\partial C_{T_j} \partial C_{T_k}} f_y(C_T, u_x, u_y), \end{aligned} \tag{17}$$

$$\frac{\partial}{\partial x}\left(\frac{\partial^2 u_x}{\partial C_{T_j} \partial C_{T_k}}\right) + \frac{\partial}{\partial y}\left(\frac{\partial^2 u_y}{\partial C_{T_j} \partial C_{T_k}}\right) = 0. \tag{18}$$

Equations (17) and (18) are again linear partial differential equations and they are the same equation with the sensitivity equation (13) but with different right-hand sides. Boundary conditions for (17)–(18) are set by prescribing homogeneous Dirichlet boundary conditions on the constant inflow boundary and natural boundary conditions on the outflow boundary. By using mixed finite element method to discretize the equations (17)–(18), we have

$$\underbrace{\begin{bmatrix} \nu K + N_l + J_{xx}^{n_l} + J_{xx}^{f'} & J_{xy}^{n_l} + J_{xy}^{f'} & B_x^T \\ J_{yx}^{n_l} + J_{yx}^{f'} & \nu K + N_l + J_{yy}^{n_l} + J_{yy}^{f'} & B_y^T \\ B_x & B_y & 0 \end{bmatrix}}_{\mathcal{A}'} \gamma_{jk} = \xi_{jk}. \tag{19}$$

Here we reuse $\frac{\partial^2}{\partial C_{T_j}\partial C_{T_k}} u_x$, $\frac{\partial^2}{\partial C_{T_j}\partial C_{T_k}} u_y$, $\frac{\partial^2}{\partial C_{T_j}\partial C_{T_k}} p$ to represent their discrete analog, respectively, and $\gamma_{jk} = \begin{bmatrix} \frac{\partial^2}{\partial C_{T_j}\partial C_{T_k}} u_x^T & \frac{\partial^2}{\partial C_{T_j}\partial C_{T_k}} u_y^T & \frac{\partial^2}{\partial C_{T_j}\partial C_{T_k}} p^T \end{bmatrix}^T$.

For the linear system (19), $J_{xx}^{n_l}$, $J_{xy}^{n_l}$, $J_{yx}^{n_l}$, and $J_{yy}^{n_l}$ are the corresponding Jacobian matrices as introduced in (14). The Jacobian matrices $J_{(\cdot)}^{f'}$ is obtained by computing the second-order derivatives of f_x, f_y with respect to C_T and using partial differentiation rules. Since Equations (17)–(18) are the same with the sensitivity equation (13) but only with different right-hand sides, the linear system (19) has the same system matrix as the linear system (14), that is, $\mathcal{A}' = \mathcal{A}$. Moreover, $\mathcal{A}$ is just the system matrix from the last Newton step to solve the nonlinear flow equation (8). The preconditioners computed for the last Newton step to solve the Navier–Stokes equation can also be reused here.

The right-hand side vector ξ_{jk} is obtained by discretizing known variables in (17)–(18). By stacking the unknowns in the following way,

$$\gamma = \begin{bmatrix} \gamma_{11} & \gamma_{12} & \dots & \gamma_{N_tN_t} \end{bmatrix} \quad \text{and} \quad \xi = \begin{bmatrix} \xi_{11} & \xi_{12} & \dots & \xi_{N_tN_t} \end{bmatrix}.$$

We get the following linear system with multiple left-hand and right-hand sides,

$$\mathcal{A}\gamma = \xi. \tag{20}$$

Here the size of unknowns N_t^2 is also much more smaller than the problem size, block Krylov methods are still well-suited to this type of problems.

Note that to compute the Hessian matrix, we need to compute the second-order derivatives by solving the linear equation (20), while the unknowns are matrices of $N_x \times N_t^2$. Here N_x is the size of spatial dimension, and N_t is the total number of variables to be optimized. This can be efficiently computed by using block Krylov subspace methods when $N_t \ll N_x$. For the other cases, quasi-Newton methods such as the Broyden–Fletcher–Goldfarb–Shanno (BFGS) algorithm[59,60] can be applied without computing the real Hessian matrix but just to compute an approximation.

4 | MULTILEVEL SEQUENTIALLY SEMISEPARABLE PRECONDITIONERS

The multilevel sequentially semiseparable (MSSS) preconditioning technique was first studied for PDE-constrained optimization problems in Reference 27 and some benchmark problems of computational fluid dynamics problems,[28] and later extended to computational geoscience problems.[68] The global MSSS preconditioner computes an approximate factorization of the global (generalized) saddle-point matrix up to a prescribed accuracy in linear computational complexity using MSSS matrix computations. To start with, we first introduce the sequentially semiseparable matrices and the block matrix definition for such matrices is given by Definition 1.

Definition 1 (31). Let A be a $p \times p$ block matrix with SSS structure such that A can be written in the following block-partitioned form

$$A_{ij} = \begin{cases} U_i W_{i+1} \dots W_{j-1} V_j, & \text{if } i < j; \\ D_i, & \text{if } i = j; \\ P_i R_{i-1} \dots R_{j+1} Q_j, & \text{if } i > j. \end{cases} \tag{21}$$

The matrices U_i, W_i, V_i, D_i, P_i, R_i, Q_i are matrices whose sizes are compatible for matrix–matrix product when their sizes are not mentioned. They are called generators for an SSS matrix.

Take a 4×4 SSS matrix for example, its generators representation is given by

$$\begin{bmatrix} D_1 & U_1V_2 & U_1W_2V_3 & U_1W_2W_3V_4 \\ P_2Q_1 & D_2 & U_2V_3 & U_2W_3V_4 \\ P_3R_2Q_1 & P_3Q_2 & D_3 & U_3V_4 \\ P_4R_3R_2Q_1 & P_4R_3Q_2 & P_4Q_3 & D_4 \end{bmatrix}.$$

Basic operations such as addition, multiplication, and inversion are closed under the SSS matrix structure and can be performed in linear computational complexity as long as the off-diagonal rank which is defined by the maximum sizes of $\{W_i\}$ and $\{R_i\}$ is much smaller compared with the matrix size. The SSS matrix structure can be derived from 1D interconnected systems.[69] While considering the grid points as interconnected systems, an SSS structured can be inferred from the discretization of 1D PDEs. Multilevel sequentially semiseparable matrices generalize the SSS matrices to the multi-dimensional case. Similar to Definition 1 for SSS matrices, the generators representation for MSSS matrices, specifically for k-level SSS matrices, is defined in Definition 2.

Definition 2 (27). The matrix A is said to be a k-level SSS matrix if it has a form like (21) and all its generators are $(k-1)$-level SSS matrices. The one-level SSS matrix is the SSS matrix that satisfies Definition 1.

Just like SSS structure for 1D discretized PDEs, we obtain an MSSS structure for high-dimensional PDEs when we lump the grid points that are one dimension lower. Take 2D PDEs for example, first we lump the grid points in one dimension and consider the lumped grid points as a single grid point, we obtain a 1D interconnected system again. This in turn gives us a two-level MSSS matrix for the discretized 2D PDEs. Similarly, we obtain a three-level MSSS matrix for 3D PDEs with the number of blocks equals the number of the grid points in the corresponding dimension.

Take the 2D Navier–Stokes equation on a square domain for example. We apply mixed finite element methods such as the $Q_1 - P_0$ method with uniform grid given by Figure 2 to discretize the Navier–Stokes equation. Here, the circles represent the grid points for velocity unknowns, while the stars are the grid points for pressure unknowns. We use the matrix K in (9) as an example to show the two-level MSSS structure of the matrix blocks in (9). The matrix K has a block tridiagonal form which is a specific MSSS structure ($W_i = 0$, $R_i = 0$), therefore, can be represented by a two-level MSSS matrix. This is shown by Figure 3. For higher order FEM discretization, the MSSS structure can also be obtained, we refer to Reference 68 for more details.

Following the relations between interconnected systems and MSSS matrices, it can be verified that all matrix blocks in (9) have a block tridiagonal structure for a uniform mesh discretization. Therefore, we can apply the MSSS permutation operations introduced in Reference 27 to permute the block system (9) with MSSS blocks into a global MSSS matrix (block tridiagonal) form given by

$$\bar{A} = \begin{bmatrix} A_{1,1} & A_{1,2} & & & \\ A_{2,1} & A_{2,2} & A_{2,3} & & \\ & A_{3,2} & A_{3,3} & \ddots & \\ & & \ddots & \ddots & \ddots \\ & & & \ddots & A_{N,N} \end{bmatrix}.$$

Here all the matrix blocks A_{ij} are MSSS matrices again and are one- level lower than the matrix $\bar{A}$. For 2D problems, $\bar{A}$ is a two-level MSSS matrix and all $A_{i,j}$ are one-level MSSS, that is, SSS matrices while $\bar{A}$ is a three-level MSSS matrix and all $A_{i,j}$ are two-level MSSS matrices for 3D problems. For details of the MSSS permutations, we refer to lemma 3.5 in Reference 27.

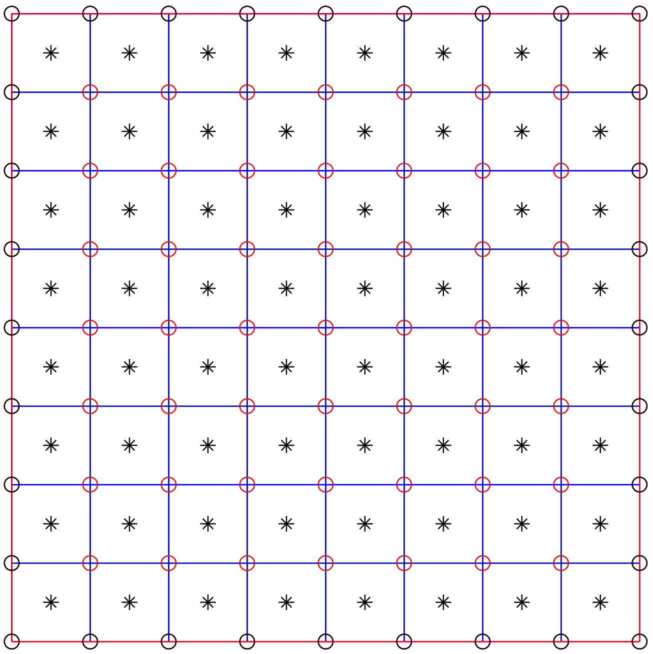
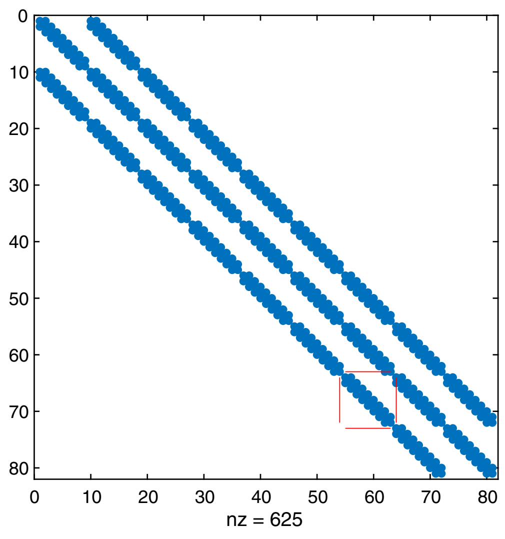
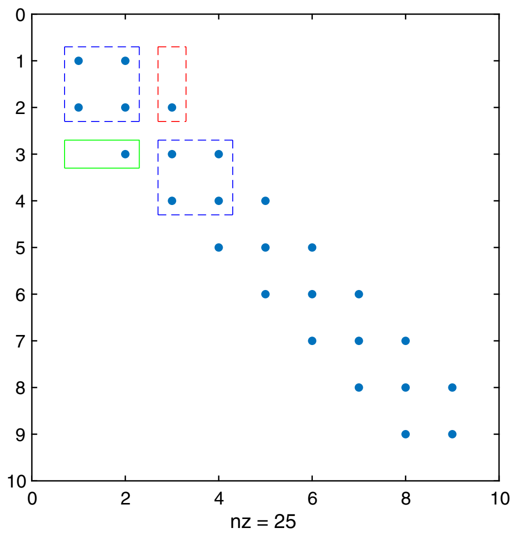


FIGURE 2 The grid for the 2D Navier–Stokes discretization



(A) Structure pattern of the upper level



(B) Structure pattern of the lower level (the labeled block on the left)

FIGURE 3 Two-level MSSS structure of K in (9)

After MSSS permutation, we can compute an block LDU factorization $\bar{A} = LDU$ where

$$L = \begin{bmatrix} I & & & & \\ A_{2,1}S_1^{-1} & I & & & \\ & A_{3,2}S_2^{-1} & I & & \\ & & \ddots & \ddots & \\ & & & A_{N,N-1}S_{N-1}^{-1} & I \end{bmatrix}, \quad D = \begin{bmatrix} S_1 & & & \\ & S_2 & & \\ & & \ddots & \\ & & & S_N \end{bmatrix}, \quad U = \begin{bmatrix} I & S_1^{-1}A_{1,2} & & \\ & I & S_2^{-1}A_{2,3} & \\ & & \ddots & \ddots \\ & & & S_{N-1}^{-1}A_{N-1,N} \\ & & & I \end{bmatrix}. \tag{22}$$

Here S_i is the Schur complement at the ith factorization step and satisfies the recursion given by

$$S_i = \begin{cases} A_{i,i} & \text{if } i = 1 \\ A_{i,i} - A_{i,i-1} S_{i-1}^{-1} A_{i-1,i} & \text{if } 2 \le i \le N \end{cases}.$$

Note that the Schur complement S_i is a dense matrix and one dimension lower than the original problem. Under the MSSS framework, we apply MSSS matrix computations to approximate the Schur complement S_i, which yields an MSSS preconditioner. Related work in applying rank structured matrix computations for the Schur complement approximation in the block triangular factorization are studied in References 38,70.

Compared with block preconditioners for the generalized saddle-point system (9) that need an efficient approximation of the Schur complement $S = -B(A + J)^{-1}B^T$, the permutation operation is a highlight for MSSS preconditioners, which enables us to compute a global factorization of the system matrix without approximation of S.

The block LDU factorization can be computed in linear computational complexity, that is, $\mathcal{O}(r^3 N_s)$, where r is the off-diagonal rank of the Schur complement S_i and bounded by a small constant, N_s is the problem/matrix size. This is achieved by performing the model order reduction operations for the MSSS matrices. In this article, we focus on 2D problems where the Schur complement is 1D and can be approximated by one-level MSSS matrix computations. For the application of MSSS preconditioners for 3D problems, we refer to References 27,68. Moreover, we have the following theorem that gives the bound of the MSSS preconditioned spectrum for 2D problems.

Theorem 1. *For a nonsingular two-level MSSS matrix $\bar{A}$, we can compute an approximate block factorization given by $\tilde{\bar{A}} = \tilde{L}\tilde{D}\tilde{U}$ with $\tilde{L}$, $\tilde{D}$, $\tilde{U}$ of the form in (22) that satisfies*

$$\|\bar{A} - \tilde{\bar{A}}\|_2 \le \varepsilon, \quad \text{and} \quad \|I - \tilde{\bar{A}}^{-1}\bar{A}\|_2 \le \frac{\varepsilon}{\varepsilon_0 - \varepsilon},$$

where ε_0 is the smallest singular value of $\bar{A}$, I denotes the identity matrix, and ε satisfies

$$\varepsilon \le 2\sqrt{p}(p-1)\tau.$$

Here p is the number of blocks in the one-level SSS blocks of $\tilde{\bar{A}}$, which is smaller than the number of the Schur complements, that is, N, and τ is a freely chosen parameter to compute the MSSS preconditioner.

Proof. For the proof of this theorem, we refer to lemma 3.2, theorem 3.6, and theorem 5.3 of Reference 71. ■

Theorem 1 states that the MSSS preconditioner can be computed up to arbitrary accuracy, this in turn makes the MSSS preconditioned spectrum cluster around $(1, 0)$ and the radius of the circle that contains the preconditioned spectrum can be made arbitrarily small. This results in a robust preconditioner and makes the convergence of the MSSS preconditioned Krylov solver independent of the mesh size and Reynolds number for the generalized saddle-point system (12).

In practice, ε is usually of the same order as τ, which shows that the bound given by Theorem 1 is pessimistic in practice. Choosing τ between 10^{-2} and 10^{-4} usually gives satisfactory performance for MSSS preconditioned Krylov solvers. This in turn yields a small r that is bounded by a constant independent of N_s and $r \ll N_s$, which means that the MSSS preconditioner can be computed in linear computational complexity. Numerical results in Section 5 verify this statement.

To analyze the storage costs of MSSS preconditioners, we take an $N \times N$ grid for example. Under the MSSS framework, we approximate the off-diagonal blocks of the dense Schur complements with low-rank patterns. To simplify the analysis, we assume that all p diagonal blocks of the SSS matrix are of uniform size and the off-diagonal rank is bounded by $r \ll N$. The storage costs for each Schur complements is

$$p\left(\frac{N}{p}\right)^2 + 4r\frac{N}{p}(p-1) + 2r^2(p-2).$$

Theoretically, we set the number of blocks $p = N$ for smaller computational complexity and storage costs while in practice the sizes of the diagonal blocks are often set to a small number, say 8 for making use of level 3 BLAS implementations, that is, $p = \frac{N}{8} \sim \mathcal{O}(N)$. This in turn gives $\mathcal{O}(N)$ storage costs for each Schur complement and $\mathcal{O}(N^2)$ storage costs for all the Schur complements for MSSS preconditioners, which is linear with respect to the problem size N^2. This is advantageous over storing all full dense S_i that results in $\mathcal{O}(N^3)$ storage costs and $\mathcal{O}(N^4)$ computational complexity which is square with respect to the problem size compared with linear computational complexity for MSSS preconditioners. For the storage costs of MSSS preconditioners for 3D problems, we refer to section 4.4 of Reference 68.

5 | NUMERICAL EXPERIMENTS

In the previous sections, we formulate an in-domain control problem. It is shown that this in-domain control problem can be solved by the reduced Newton method described by Algorithm 1. The biggest computational issue is to solve a nonlinear flow equation (8) by using the Newton's method and two linear equations (14)–(19) with generalized saddle-point structure to compute the gradient and the Hessian matrix. At each optimization step, we solve the nonlinear flow equation (8) with inner iterations. A preconditioned Krylov solver is performed at each inner iteration.

In Section 3, we showed that the linearized flow equation (9) is a perturbed linearized Navier–Stokes equation (10) with singular perturbation on the $(1, 1)$ block of the linearized Navier–Stokes system matrix. Standard preconditioning techniques for the Navier–Stokes equation, which highly depend on efficient approximation of the Schur complement, fail to give satisfactory performance for this problem due to this perturbation. This will be illustrated by numerical results later. In this section, we evaluate the performance of the MSSS preconditioning techniques for the in-domain control problem and compare with the pressure convection diffusion (PCD) preconditioner[72] that is implemented in IFISS. All the numerical experiments are implemented in MATLAB 2015a on a desktop of Intel Core i5 CPU of 3.10 GHz and 16 GB memory with the Debian GNU/Linux 8.0 system.

5.1 | Preconditioning techniques

In this part, we report the performance of the MSSS preconditioner and the PCD preconditioner for the second Newton iteration in solving the Navier–Stokes equation with external source terms at the first optimization step of Algorithm 1. We use the `IDR(s)` method[73] to solve the preconditioned system. The preconditioned `IDR(s)` solver is stopped if the 2-norm of the residual at step k, which is denoted by $\|r_k\|_2$, satisfies $\|r_k\|_2 \leq 10^{-4}\|r_0\|_2$.

To compute an MSSS preconditioner for a linear system of the generalized saddle-point-type (9), we first convert all the matrix blocks to the MSSS format, which is illustrated by Figure 3. The cost for this part is little since no computations are performed and only memory allocations are needed. Second, we permute the matrix with MSSS blocks into a global MSSS matrix. In this step, all the operations are performed to concatenate generators of MSSS matrices and no computations are needed. The final step is to compute an approximate LDU factorization of the global matrix using MSSS matrix computations. This is the most computational intensive part for the MSSS preconditioners and the computational complexity is linear with respect to the problem size as we have analyzed in Section 4, which is also illustrated by numerical results in this section.

We also compare the performance of MSSS preconditioners with that of the PCD preconditioner $\mathcal{P}_p$ for the linear system (12) given by,

$$\mathcal{P}_p = \begin{bmatrix} A + J^f & B^T \\ & -S_p \end{bmatrix}, \tag{23}$$

where $S_p = L_p A_p^{-1} M_p$ is the approximation of the Schur complement $B(A + J^f)^{-1}B^T$. Here, A_p and L_p are the convection–diffusion operator and Laplace operator in the finite dimensional solution space of the pressure with prescribed boundary conditions, M_p is the pressure mass matrix. For this PCD preconditioner, both $A + J^f$ and S_p are approximated by the algebraic multigrid (AMG) method that is implemented in IFISS.

We set the Reynolds number $Re = 2000$ as discussed in the previous section. We report the performance of both preconditioners in Tables 1 and 2. Here the column "precon." represents the time for MSSS permutations and the block LDU factorization using MSSS matrix computations or the time for the setup of the AMG method, and the column "IDR(4)" denotes the time of the preconditioned `IDR(4)` solver to compute the solution up to prescribed accuracy. Both time are measured in seconds. Since we focus on 2D problems, the MSSS matrices are two-level MSSS matrices and for making use of level 3 BLAS implementations, we set the size of the SSS blocks to 8 for all test cases.

Grid	Problem size	#Iter.	Precon. (s)	IDR(4) (s)	Total (sec.)
64×64	$1.25e+04$	2	4.36	0.64	5.00
128×128	$4.97e+04$	3	18.43	2.53	20.96
256×256	$1.98e+05$	5	65.09	9.25	74.34
512×512	$7.88e+05$	3	272.63	24.62	297.25

TABLE 1 Computational results of the MSSS preconditioner for $Re = 2000$

Grid	Problem size	#Iter.	Precon. (s)	IDR(4) (s)	Total (s)
64×64	$1.25e+04$	400	8.56	no convergence	–
128×128	$4.97e+04$	400	70.74	no convergence	–
256×256	$1.98e+05$	266	237.68	42.93	280.61
512×512	$7.88e+05$	203	1386.98	101.72	1488.70

TABLE 2 Computational results of the PCD preconditioner for $Re = 2000$

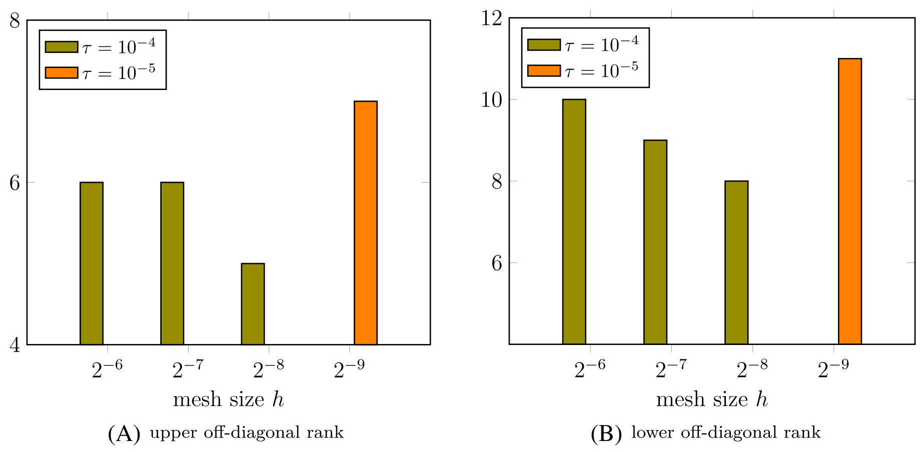


FIGURE 4 Maximum off-diagonal rank of MSSS preconditioner

Table 1 shows the time to compute the MSSS preconditioner scales linearly with the problem size. The number of iterations remains virtually constant with the refinement of the mesh. The time of the preconditioned `IDR(4)` solver also scales linearly with the problem size. For PCD preconditioner, the preconditioned `IDR(4)` solver fails to converge to the solution of prescribed accuracy within 400 iterations for relatively bigger mesh size. For a very fine mesh, the preconditioned `IDR(4)` solver computes the solution up to the prescribed accuracy within 300 iterations. This is because the entries of perturbation by the matrix J^f in (23), which is introduced by the nonlinear right-hand side, is of $\mathcal{O}(h^2)$. As $h \rightarrow 0$, this perturbation by the Jacobian matrix becomes smaller compared with A in the $(1, 1)$ block of (23). Therefore, S_p approximates the perturbed Schur complement that corresponds to smaller mesh sizes better than that for bigger mesh sizes.

The computational results by the MSSS preconditioner in Table 2 show that the time for the setup of AMG does not scale linearly with the problem size. The reason may be that the AMG implemented in IFISS is not optimized, or the AMG algorithm in IFISS does not have linear computational complexity. To compute the MSSS preconditioner, we set $\tau = 10^{-5}$ for the 512×512 grid and 10^{-4} for the rest grids. Here τ is the upper bound of the discarded singular values for the model order reduction that is performed to compute the approximate factorization. The maximum off-diagonal rank for the lower and upper part for this set up are plotted in Figure 4. For details of this approximate factorization, we refer to Reference 71. The maximum off-diagonal rank in Figure 4 is bounded by a small constant around 10 for all the computations of MSSS preconditioners, which is much smaller than the size of the systems to be solved. This in turn gives the linear computational complexity of the MSSS preconditioning techniques, which is illustrated by Table 1.

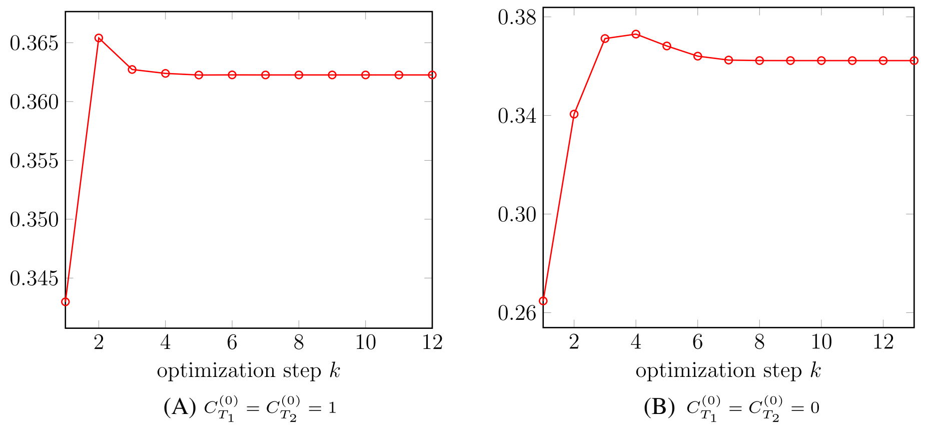


FIGURE 5 Total output power (scaled)

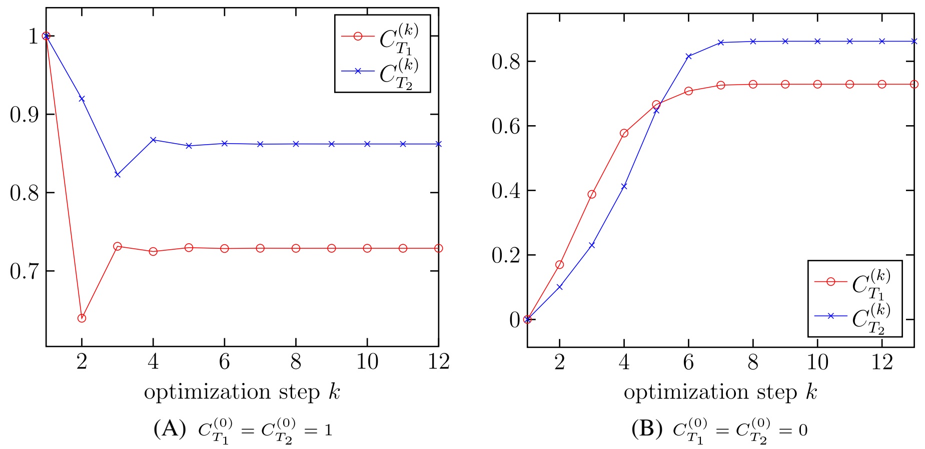


FIGURE 6 Optimized C_{T_1}, C_{T_2} with different initial guesses

5.2 | Optimization algorithm

We test Algorithm 1 for the optimization problem (6) by using a 256×256 grid. At each outer iteration, we need to solve a series of linear equations of size $197,634 \times 197,634$ to compute the solution of the nonlinear flow equation (8) by the Newton's method. We also need to solve two linear equations (14) and (19) of the same size to compute the gradient and the Hessian matrix.

We use the wind farm configuration as introduced in Section 2.1. With this problem settings, the rightmost turbine just operates in the maximum power tracking mode, that is, $C_{T_3} = 1$. Therefore, we just need to optimize C_{T_1} and C_{T_2} for the first two turbines. Without optimization, the turbines are operated in the mode that corresponds to the maximal power extraction for each single turbine, that is, $C_{T_1} = C_{T_2} = 1$. We start with $C_{T_1}^{(0)} = C_{T_2}^{(0)} = 1$ as an initial start for this optimization problem. Then the (scaled) total extracted power by the wind farm at each optimization step is given in Figure 5(a).

Results in Figure 5(a) show that the total power is increased by around 5.5% when applying optimal control scheme. To show the performance of Algorithm 1, we also solve the optimization problem with an initial start $C_{T_1}^{(0)} = C_{T_2}^{(0)} = 0$, although this corresponds to an impractical operation status. The scaled total power is given in Figure 5(b). For those two cases with different initial guesses, the corresponding optimized variables C_{T_1} and C_{T_2} at each optimization step are given in Figure 6. Figure 6(a,b) show that with different initial guesses, the optimized variables (C_{T_1}, C_{T_2}) converge to the same point (0.729, 0.862), which corresponds to a local optimum of the optimization problem.

Note that there is an overshoot after a few optimization steps for both cases as illustrated by Figure 5, which makes the optimization problem to converge to a local optimum. This may be primarily because of the nonconvexity and highly nonlinearity of this optimization problem. The convexity of this optimization problem is still an open problem. Another reason that contributes to this behavior is the sensitivity of this optimization problem. Here we measure the sensitivity of the change of the velocity in the flow direction with respect to C_{T_j} by $\frac{\partial u_x}{\partial C_{T_j}}$. We plot the magnitude of $\frac{\partial u_x}{\partial C_{T_1}}$ and $\frac{\partial u_x}{\partial C_{T_2}}$ at the local optimum of C_{T_1} and C_{T_2} in Figure 7.

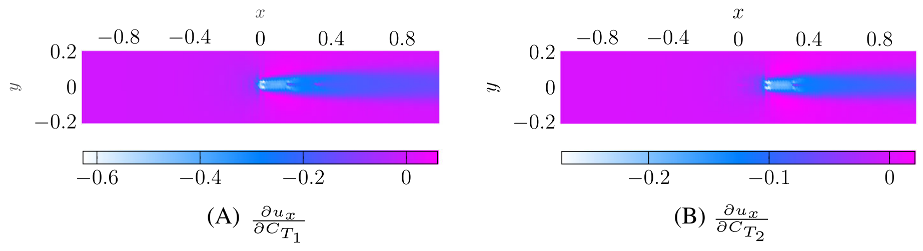


FIGURE 7 $\frac{\partial u_x}{\partial C_{T_1}}$ and $\frac{\partial u_x}{\partial C_{T_2}}$ in magnitude

Figure 7 show that there is a big gradient in the vicinity of the places where the turbines are located. This in turn tells us that the velocity in the vicinity of the turbines is very sensitive to the changes of C_{T_j}. This makes this optimization problem very sensitive. Another reason may be the robustness and the efficiency of the Newton's method given by Algorithm 1. To overcome such an overshot, instead of the fixed step size of the Newton's method, some line search methods such as the Armijo linear search algorithm[60] could be applied to adaptively choose the step size.

6 | CONCLUSIONS AND REMARKS

In this article, we studied preconditioning in-domain Navier–Stokes control problem using multilevel sequentially semiseparable (MSSS) matrix computations. For the in-domain Navier–Stokes control problem, the control input only acts on part of the domain, which results in a problem even more difficult to solve. The optimization problem was solved by a reduced Newton's method while the most time consuming part for this problem was solving a nonlinear flow equation and computations of the gradient and the Hessian matrix. We showed that the gradient and the Hessian matrix can be computed by solving a linear equation that is exactly the same as the last nonlinear iteration of the Navier–Stokes equation. Therefore, the preconditioner for the solution of the flow equation can be reused. This in turn reduces the computational complexity dramatically. We evaluated the performance of the MSSS preconditioning technique by using IFISS. Numerical results show the superiority of the MSSS preconditioning technique to the standard preconditioning technique.

Since we focused on preconditioning in-domain Navier–Stokes control problem, we used a laminar flow model in IFISS to study the performance of the MSSS preconditioning technique. The next step to extend this research shall focus on applying the turbulent flow model for the real-world wind farm control applications, while some recent development in optimal control of unsteady Reynolds-averaged Navier–Stokes equations (RANS)[74] is a good starting point to extend our preconditioning technique.

The MSSS structure can be inferred from a topologically uniform mesh discretization of a regular domain. When general domains can be parameterized using topologically uniform mesh, such as in the isogeometric analysis, applying MSSS is direct. Extending MSSS to unstructured meshes is also possible, one can first compute the solution of the original problem on a structured mesh and then interpolate on the unstructured mesh since MSSS is often used as a preconditioner but not a direct solver. By following this procedure, the convergence of the Krylov solver for problems with unstructured mesh discretization can be improved.

ACKNOWLEDGMENTS

The authors would like to thank the anonymous referees that greatly help to improve the quality of the work presented in this paper. This work does not have any conflicts of interest.

ORCID

Yue Qiu https://orcid.org/0000-0003-0360-0442
Cornelis Vuik https://orcid.org/0000-0002-5988-9153

REFERENCES

1. Leineweber DB, Bauer I, Bock HG, Schlöder JP. An efficient multiple shooting based reduced SQP strategy for large-scale dynamic process optimization. Part I: theoretical aspects. Comput Chem Eng. 2003;27(2):157–166.
2. Borzi A, Schulz V. Multigrid methods for PDE optimization. SIAM Rev. 2009;51(2):361–395.

3. Biegler LT, Ghattas O, Heinkenschloss M, van Bloemen WB. Large-scale PDE-constrained optimization: an Introduction. In: Biegler LT, Ghattas O, Heinkenschloss M, van Bloemen WB, editors. Large-scale PDE-constrained optimization. New York, NY: Springer, 2003; p. 3–13.
4. Gill PE, Murray W, Ponceleón DB, Saunders MA. Preconditioners for indefinite systems arising in optimization. SIAM J Matrix Anal Appl. 1992;13(1):292–311.
5. Schöberl J, Zulehner W. Symmetric indefinite preconditioners for saddle point problems with applications to PDE-constrained optimization problems. SIAM J Matrix Anal Appl. 2007;29(3):752–773.
6. Badreddine H, Vandewalle S, Meyers J. Sequential Quadratic Programming (SQP) for optimal control in direct numerical simulation of turbulent flow. J Comput Phys. 2014;256:1–16.
7. Hinze M, Kunisch K. Second order methods for optimal control of time-dependent fluid flow. SIAM J Control Optim. 2001;40(3):925–946.
8. Laumen M. Newton's method for a class of optimal shape design problems. SIAM J Optim. 2000;10(2):503–533.
9. Berggren M. Numerical solution of a flow-control problem: vorticity reduction by dynamic boundary action. SIAM J Sci Comput. 1998;19(3):829–860.
10. Fursikov AV, Gunzburger MD, Hou LS. Boundary value problems and optimal boundary control for the Navier-Stokes system: the two-dimensional case. SIAM J Control Optim. 1998;36(3):852–894.
11. Goit JP, Meyers J. Optimal control of energy extraction in wind-farm boundary layers. J Fluid Mech. 2015;768:5–50.
12. Adavani SS, Biros G. Multigrid algorithms for inverse problems with linear parabolic PDE constraints. SIAM J Sci Comput. 2008;31(1):369–397.
13. Axelsson O, Neytcheva M. Preconditioning methods for linear systems arising in constrained optimization problems. Numer Linear Algebra Appl. 2003;10(1-2):3–31.
14. Axelsson O, Farouq S, Neytcheva M. Comparison of preconditioned Krylov subspace iteration methods for PDE-constrained optimization problems: Poisson and convection-diffusion control. Numer Algorithms. 2016;73(3):631–663.
15. Axelsson O, Farouq S, Neytcheva M. Comparison of preconditioned Krylov subspace iteration methods for PDE-constrained optimization problems: Stokes control. Numer Algorithms. 2017;74(3):19–37.
16. Bai ZZ, Wang ZQ. On parameterized inexact Uzawa methods for generalized saddle point problems. Linear Algebra Appl. 2008;428(11):2900–2932.
17. Pearson JW, Gondzio J. Fast interior point solution of quadratic programming problems arising from PDE-constrained optimization. Numerische Mathematik. 2017;137(4):959–999.
18. Pearson JW, Pestana J, Silvester DJ. Refined saddle-point preconditioners for discretized Stokes problems. Numerische Mathematik. 2018;138(2):331–363.
19. Potschka A, Mommer MS, Schlöder JP, Bock HG. Newton-picard-based preconditioning for linear-quadratic optimization problems with time-periodic parabolic PDE constraints. SIAM J Sci Comput. 2012;34(2):A1214–A1239.
20. Rees T, Wathen AJ. Preconditioning iterative methods for the optimal control of the Stokes equations. SIAM J Sci Comput. 2011;33(5):2903–2926.
21. Stoll M, Breiten T. A low-rank in time approach to PDE-constrained optimization. SIAM J Sci Comput. 2015;37(1):B1–B29.
22. Bánsch E, Benner P, Saak J, Weichelt H. Riccati-based boundary feedback stabilization of incompressible Navier–Stokes flows. SIAM J Sci Comput. 2015;37(2):A832–A858.
23. Mardal KA, Nielsen BF, Nordaas M. Robust preconditioners for PDE-constrained optimization with limited observations. BIT Numer Math. 2017;57(2):405–431.
24. Heidel G, Wathen A. Preconditioning for boundary control problems in incompressible fluid dynamics. Numer Linear Algebra Appl. 2019;26(1):e2218.
25. Pearson JW. Preconditioned iterative methods for Navier-Stokes control problems. J Comput Phys. 2015;292(0):194–207.
26. Meyers J, Meneveau C. Optimal turbine spacing in fully developed wind farm boundary layers. Wind Energy. 2012;15(2):305–317.
27. Qiu Y, van Gijzen MB, van Wingerden JW, Verhaegen M, Vuik C. Efficient preconditioners for PDE-constrained optimization problems with a multilevel sequentially semiseparable matrix structure. Electr Trans Numer Anal. 2015;44:367–400.
28. Qiu Y, van Gijzen MB, van Wingerden JW, Verhaegen M, Vuik C. Evaluation of multilevel sequentially semiseparable preconditioners on CFD benchmark problems using incompressible flow and iterative solver software. Math Methods Appl Sci. 2018;41(3):888–903.
29. Elman H, Ramage A, Silvester D. IFISS: a computational laboratory for investigating incompressible flow problems. SIAM Rev. 2014;56(2):261–273.
30. Silvester DJ, Elman HC, Ramage A. Incompressible flow and iterative solver software (IFISS) version 3.2; 2012. Http://www.manchester.ac.uk/ifiss/.
31. Chandrasekaran S, Dewilde P, Gu M, et al. Some fast algorithms for sequentially semiseparable representations. SIAM J Matrix Anal Appl. 2005;27(2):341–364.
32. Vandebril R, Van Barel M, Mastronardi N. Matrix computations and semiseparable matrices: linear systems. Baltimore: Johns Hopkins University Press, 2007.
33. Hackbusch WA. Sparse matrix arithmetic based on $\mathscr{H}$-matrices. Part I: introduction to $\mathscr{H}$-Matrices. Computing. 1999;62(2):89–108.
34. Börm S. $\mathcal{H}_2$-matrices–Multilevel methods for the approximation of integral operators. Comput Vis Sci C. 2004;7(3-4):173–181.
35. Hackbusch W, Börm S. Data-sparse approximation by adaptive $\mathscr{H}^2$-matrices. Computing. 2002;69(1):1–35.
36. Chandrasekaran S, Dewilde P, Gu M, Lyons W, Pals T. A fast solver for HSS representations via sparse matrices. SIAM J Matrix Anal Appl. 2006;29(1):67–81.

37. Xia J, Chandrasekaran S, Gu M, Li XS. Superfast multifrontal method for large structured linear systems of equations. SIAM J Matrix Anal Appl. 2010;31(3):1382–1411.
38. Bebendorf M. Why finite element discretizations can be factored by triangular hierarchical matrices. SIAM J Numer Anal. 2007;45(4):1472–1494.
39. Le Borne S, Grasedyck L. $\mathcal{H}$ matrix preconditioners in convection-dominated problems. SIAM J Matrix Anal Appl. 2006;27(4):1172–1183.
40. Xia J. A robust inner-outer hierarchically semi-separable preconditioner. Numer Linear Algebra Appl. 2012;19(6):992–1016.
41. Börm S, and Le Borne S. $\mathcal{H}$ LU factorization in pre-conditioners for augmented Lagrangian and grad-div stabilized saddle point systems. Int J Numer Methods Fluids 2012;68(1):83–98.
42. Cai D, Chow E, Erlandson L, Saad Y, Xi Y. SMASH: structured matrix approximation by separation and hierarchy. Numer Linear Algebra Appl. 2018;25(6):e2204. https://doi.org/10.1002/nla.2204.
43. Erlandson L, Cai D, Xi Y, Chow E. Accelerating parallel hierarchical matrix-vector products via data-driven sampling. Proceedings of the 2020 IEEE International Parallel and Distributed Processing Symposium (IPDPS). New Orleans: IEEE; 2020. p. 749–758.
44. Amestoy P, Ashcraft C, Boiteau O, Buttari A, L'Excellent JY, Weisbecker C. Improving multifrontal methods by means of block low-rank representations. SIAM J Sci Compu. 2015;37(3):A1451–A1474.
45. Aminfar A, Darve E. A fast, memory efficient and robust sparse preconditioner based on a multifrontal approach with applications to finite-element matrices. Int J Numer Meth Eng. 2016;107(6):520–540.
46. Gillman A, Martinsson PG. An $\mathcal{O}(N)$ algorithm for constructing the solution operator to 2D elliptic boundary value problems in the absence of body loads. Adv Comput Math. 2014;40(4):773–796.
47. Grasedyck L, Kriemann R, Le Borne S. Domain decomposition based $\mathscr{H}$-LU preconditioning. Numer Math. 2009;112(4):565–600.
48. Schmitz PG, Ying L. A fast nested dissection solver for Cartesian 3D elliptic problems using hierarchical matrices. J Comput Phys. 2014;258:227–245.
49. Annoni J, Gebraad PMO, Scholbrock AK, Fleming PA, van Wingerden JW. Analysis of axial-induction-based wind plant control using an engineering and a high-order wind plant model. Wind Energy. 2015;19(6):1135–1150.
50. Crespo A, Hernández FJR, Frandsen S. Survey of modelling methods for wind turbine wakes and wind farms. Wind Energy. 1999;2(1):1–24.
51. Sanderse B, van der Pijl SP, Koren B. Review of computational fluid dynamics for wind turbine wake aerodynamics. Wind Energy. 2011;14(7):799–819.
52. Annoni J, Seiler P, Johnson K, Fleming P, Gebraad P. Evaluating wake models for wind farm control. Portland: Proceedings of American Control Conference; 2014. p. 2517–2523.
53. Abkar M, Porté-Agel F. The effect of free-atmosphere stratification on boundary-layer flow and power output from very large wind farms. Energies. 2013;6(5):2338–2361.
54. White FM, Corfield I. Viscous fluid flow. New York, NY: McGraw-Hill, 2006.
55. Avila K, Moxey D, de Lozar A, Avila M, Barkley D, Hof B. The onset of turbulence in pipe flow. Science. 2011;333(6039):192–196.
56. Torres P, Van Wingerden JW, Verhaegen M. Modeling of the flow in wind farms for total power optimization. Santiego Chile: Proceedings of the 9th IEEE International Conference on Control and Automation; 2011. p. 963–968.
57. Mikkelsen R. *Actuator Disc Methods Applied to Wind Turbines* [PhD Thesis]. Technical University of Denmark; 2003.
58. Troldborg N. *Actuator Line Modeling of Wind Turbine Wakes* [PhD Thesis]. Technical University of Denmark; 2008.
59. Nocedal J, Wright S. Numerical optimization. New York, NY: Springer Science & Business Media, 2006.
60. Heinkenschloss M. Numerical solution of implicitly constrained optimization problems. CAAM Technical Report TR08-05. Houston: Department of Computational and Applied Mathematics, Rice University, 2008.
61. Benzi M, Golub GH, Liesen J. Numerical solution of saddle point problems. Acta Numer. 2005;14:1–137.
62. Li C, Vuik C. Eigenvalue analysis of the SIMPLE preconditioning for incompressible flow. Numer Linear Algebra Appl. 2004;11(5-6):511–523.
63. Benzi M, Olshanskii MA, Wang Z. Modified augmented Lagrangian preconditioners for the incompressible Navier-Stokes equations. Int J Numer Methods Fluids. 2011;66(4):486–508.
64. Kay D, Loghin D, Wathen A. A preconditioner for the steady-state Navier-Stokes equations. SIAM J Sci Comput. 2002;24(1):237–256.
65. Borzí A, Schulz V. Computational optimization of systems governed by partial differential equations. Philadelphia: SIAM, 2011.
66. Abe K, Sleijpen GLG. Hybrid Bi-CG methods with a Bi-CG formulation closer to the IDR approach. Appl Math Comput. 2012;218(22):10889–10899.
67. Du L, Sogabe T, Yu B, Yamamoto Y, Zhang S. A block IDR(s) method for nonsymmetric linear systems with multiple right-hand sides. J Comput Appl Math. 2011;235(14):4095–4106.
68. Baumann M, Astudillo R, Qiu Y, Ang EY, van Gijzen MB, Plessix RÉ. An MSSS-preconditioned matrix equation approach for the time-harmonic elastic wave equation at multiple frequencies. Comput Geosci. 2018;22(1):43–61.
69. Rice JK, Verhaegen M. Efficient system identification of heterogeneous distributed systems via a structure exploiting extended Kalman filter. IEEE Trans Automat Control. 2011;56(7):1713–1718.
70. Bağcl H, Pasciak JE, Sirenko KY. A convergence analysis for a sweeping preconditioner for block tridiagonal systems of linear equations. Numer Linear Algebra Appl. 2014;22(2):371–392.
71. Qiu Y, van Gijzen MB, van Wingerden JW, Verhaegen M, Vuik C. Convergence analysis of the multilevel sequentially semiseparable preconditioners. Technical Report 15-01. Delft, The Netherlands: Delft Institution of Applied Mathematics, 2015.

72. Elman HC, Silvester DJ, WAJ. Finite elements fast iterative solvers: with applications in incompressible fluid dynamics. New York, NY: Oxford University Press, 2005.
73. van Gijzen MB, Sonneveld P. Algorithm 913: an elegant IDR(s) variant that efficiently exploits biorthogonality properties. ACM Trans Math Softw. 2011;38(1):5:1–5:19.
74. Günther S, Gauger NR, Wang Q. Simultaneous single-step one-shot optimization with unsteady PDEs. J Comput Appl Math. 2016;294:12–22.

How to cite this article: Qiu Y, van Gijzen MB, van Wingerden J-W, Verhaegen M, Vuik C. Preconditioning Navier–Stokes control using multilevel sequentially semiseparable matrix computations. *Numer Linear Algebra Appl*. 2021;28:e2349. https://doi.org/10.1002/nla.2349