AIAA AVIATION Forum
August 2-6, 2021, VIRTUAL EVENT
AIAA AVIATION 2021 FORUM

10.2514/6.2021-2896

A randomized time-domain algorithm for efficiently computing resolvent modes

Ali Farghadan* and Aaron Towne†
Department of Mechanical Engineering, University of Michigan, Ann Arbor, MI, 48109, USA

Eduardo Martini‡
Institut Pprime CNRS, Université de Poitiers ENSMA, France

André V. G. Cavalieri§
Divisão de Engenharia Aeronáutica, Instituto Tecnológico de Aeronáutica, São José dos Campos, Brazil

We introduce a new algorithm for computing resolvent modes of large systems based on randomized singular value decomposition (RSVD) combined with a time-marching method. The most expensive steps of the RSVD algorithm in the context of resolvent analysis, which constitute a bottleneck in its application to large systems, are replaced by leveraging the time-domain equations that have given rise to the resolvent operator. Specifically, the actions of the resolvent operator and its adjoint on a vector are obtained by equivalent direct and adjoint marching operations in the time domain. Our algorithm exploits streaming calculations to alleviate memory issues emerging for large systems, and we develop strategies to minimize the time-stepping cost while maintaining a desired level of accuracy. We validated our proposed algorithm by comparing the resolvent modes and gains of a Ginzburg-Landau model problem to those obtained from RSVD. Then, we use an axisymmetric jet and a three-dimensional extension thereof to assess and demonstrate the accuracy, cost, and memory efficiency of our new algorithm when applied to a high-dimensional system. In the three-dimensional case, we achieve orders-of-magnitude reduction in both CPU and memory usage compared to a direct application of RSVD.

I. Introduction

In the era of big data, the need for efficient dimension reduction is critically important. One important object for analysis and model reduction of dynamical systems is the resolvent operator.[1,2] The resolvent operator acts as a transfer function between inputs and the response of the state. Within fluid mechanics, it provides a simplified framework to study flow perturbations[3,4] and has arguably become one of the most important operator-theoretic methods in the field.[5,6]

The pseudo-spectral analysis of instability growth in shear flows by Trefethen *et al.*[7] shaped the early formulation of resolvent analysis. The input-output viewpoint was then used to study energy amplification mechanisms within a channel flow by Jovanovic & Bamieh,[8] who found that the most amplified structures obtained from resolvent analysis correspond to structures commonly observed in experiments. Mckeon & Shrama[9] suggested that the input (forcing) be used to represent the nonlinear terms that would otherwise be discarded, clarifying its interpretation, and that of resolvent analysis more broadly, for turbulent flows. When the forcing has no preferred direction and the resolvent operator is low-rank, the flow can be compactly approximated using only the leading response modes. Thereafter, various applications of resolvent analysis surged in the fluid mechanics community. Moarref *et al.*[10] found a low-rank model for streamwise energy intensity via resolvent analysis. Using the same framework, Luhar *et al.*[11] developed some guidelines for compliant walls in a turbulent flow. Towne *et al.*[12] showed that resolvent modes are closely related to coherent structures educed from data using spectral proper orthogonal decomposition (SPOD). In addition, Symon *et al.*[13] compared the energy transfer between the resolvent model and DNS across linear and nonlinear mechanisms.

*Ph.D. student, AIAA student member
†Assistant Professor, AIAA Member
‡Postdoc fellow, AIAA member
§Assistant Professor, AIAA Member

More recently, several resolvent-based models have been developed for design optimization,[14] flow estimation,[15,16] and control.[17,18]

The optimal forcing and response modes are obtained from the singular value decomposition (SVD) of the resolvent operator. However, both obtaining the resolvent operator, and computing its SVD are expensive operations using traditional methods. Despite the potential of resolvent-based modeling, the computational cost has seriously hindered the application of resolvent analysis to large systems and three-dimensional flows in particular. To address this, novel methods and algorithms have been proposed in recent years prominently based on matrix-free time-stepping methods,[16,19] randomized linear algebra techniques,[20] one-way Navier-Stokes equations,[21,22] and equation-free data-driven models.[23,24]

Randomized SVD (RSVD) provides large computational savings for approximating the leading modes of large systems.[25] This algorithm was initially used in the context of resolvent analysis by Moarref *et al.*[10] in a turbulent channel flow. A more detailed study was then published by Ribeiro *et al.*[20] showing promising results in higher dimensional systems while significantly reducing the computational cost. The key step is to create a low-dimensional subspace onto which the resolvent operator is projected. Finding the SVD of the low-dimensional approximation of the resolvent operator is less expensive than computing it for the full resolvent operator, while still yielding accurate estimates of the leading singular values and vector. Sampling the resolvent operator and projecting it onto the low-dimensional subspace both require computing the action of the resolvent operator on a vector, and this step constitutes the dominant cost of the algorithm and limits its CPU cost and memory benefits of the RSVD algorithm when applied to resolvent analysis. Our new algorithm replaces these bottlenecks with cheaper operations using direct and adjoint time marching of the linear equations in the time domain.

Monokrousos *et al.*[19] leveraged time marching of direct and adjoint equations in a power iteration setup to obtain the optimal modes of a boundary layer flow on a flat plate. Each frequency for which resolvent modes are desired is computed separately, requiring intensive effort when a range of frequencies is of interest. Brynjell-Rahkola *et al.*[26] proposed usage of a Laplace preconditioner to accelerate the power iteration convergence limited to low Reynolds flows. More recently, an efficient time-stepping technique was proposed by Martini *et al.*[16] Two versions of their approach, termed transient and steady-state response methods, were demonstrated to be cost-effective as they simultaneously capture optimal modes for several frequencies. In this paper, we combine the steady-state response method and RSVD to obtain an efficient and flexible algorithm for computing resolvent modes of large systems. In particular, the proposed algorithm replaces two aforementioned expensive steps within the RSVD algorithm with direct and adjoint marching schemes, significantly reducing its overall CPU cost and memory consumption when applied to large systems. Our approach constitutes a generalization of the power iteration approach in Martini *et al.*[16]

To demonstrate and analyze our method, we apply it, along with RSVD and SVD when possible, to three test problems. A Ginzburg-Landau equation is used to validate our method relative to a standard application of RSVD and to explore the impact of parameters within RSVD and our method on solution errors. An axisymmetric jet and a three-dimensional (3D) jet are then used to assess the method and empirically determine its CPU cost and memory scaling with problem size. Overall, these test problems demonstrate computational cost savings of our algorithm when applied to large systems, especially 3D ones.

II. Background

A. Resolvent analysis

In this section, the resolvent operator is defined for the linearized Navier–Stokes (NS) equations. The nonlinear governing equations in general coordinates are expressed as

$$\frac{\partial \mathbf{q}}{\partial t} = \boldsymbol{\mathcal{N}}(\mathbf{q}), \tag{1}$$

where $\boldsymbol{\mathcal{N}}$ is the compressible NS operator. Equation (1) can be linearized by applying a Reynolds decomposition to the flow state vector $\mathbf{q}(\mathbf{x}, t) = \bar{\mathbf{q}}(\mathbf{x}) + \mathbf{q}'(\mathbf{x}, t)$, yielding

$$\frac{\partial \mathbf{q}'}{\partial t} = \mathbf{A}(\bar{\mathbf{q}})\mathbf{q}' + \mathbf{f}'(\bar{\mathbf{q}}, \mathbf{q}'), \tag{2}$$

where $\bar{\mathbf{q}}$ denotes the steady mean or base flow, and $\mathbf{q}'$ represents the fluctuation around the base flow, $\mathbf{f}'$ is the forcing that contains the nonlinear terms and external body forces, $\mathbf{A} \in \mathbb{C}^{N\times N}$ is the linearized NS operator, and N is the state dimension of the discretized system.

Taking the Fourier transform of equation (2) and solving for the transformed state, we obtain

$$\hat{\mathbf{q}}(\omega) = \mathbf{R}(\omega)\hat{\mathbf{f}}(\omega), \tag{3}$$

where

$$\mathcal{F}(\mathbf{x}(t)) = \hat{\mathbf{x}}(\omega) = \int_{-\infty}^{+\infty} \mathbf{x}(t)e^{-i\omega t}\, dt \tag{4}$$

is the Fourier transform, $\mathbf{x}(t)$ represents any arbitrary vector, and $\hat{(\cdot)}$, denotes the frequency counterpart of the time-domain vector. The transfer function between the input $\hat{\mathbf{f}}$ and output $\hat{\mathbf{q}}$ in the frequency domain is the resolvent operator

$$\mathbf{R}(\omega) = (i\omega\mathbf{I} - \mathbf{A})^{-1} \in \mathbb{C}^{N\times N}. \tag{5}$$

The largest energy amplification of the mapping from inputs to outputs is commonly of interest. One can obtain these optimal solutions from the SVD of the resolvent operator,[27]

$$\mathbf{R} = \mathbf{U}\mathbf{\Sigma}\mathbf{V}^*, \tag{6}$$

where $(\cdot)^*$ denotes the conjugate transpose, $\mathbf{\Sigma}$ contains the singular values associated with the gain, and $\mathbf{U}$ and $\mathbf{V}$ are left and right singular vectors corresponding to output and input modes, respectively.

The procedure of explicitly obtaining the resolvent operator in equation (5) and the subsequent decomposition in equation (6) are prohibitively expensive for large systems, hindering the application of resolvent analysis to realistic three-dimensional turbulent flows. Randomized SVD (RSVD) provides a means to efficiently approximate the leading singular values and vectors of a matrix, and this capability provides one of the key ingredients of our algorithm.

B. Randomized singular value decomposition

First, we show the steps of RSVD assuming that the matrix $\mathbf{R}$ is directly accessible. This is typically not the case for resolvent analysis, and we address necessary modifications in the following section.

RSVD leverages random sampling in which a reduced matrix is obtained by sampling of $\mathbf{R}$, *i.e.*, multiplying a set of random test vectors. The *sketch* of $\mathbf{R}$[28] is thus obtained by

$$\mathbf{Y} = \mathbf{R}\mathbf{\Theta}, \tag{7}$$

where $\mathbf{\Theta} \in \mathbb{C}^{N\times k}$ is the random test matrix and the number of columns, k, determines the maximum number of leading modes that could be approximated. $\mathbf{Y}$ is the response to a non-preferential direction forcing $\mathbf{\Theta}$ and preserves the dominant left singular vectors of $\mathbf{R}$.

At this stage, an optional step is performing power iteration, which is especially useful when the singular values of $\mathbf{R}$ decay slowly. This step updates the sketch in a loop,

$$\begin{aligned} &\text{for } i = 1:q \\ &\quad \mathbf{Y}_{old} = \mathbf{Y} \\ &\quad \mathbf{Y} \;= \mathbf{R}\mathbf{R}^*\mathbf{Y}_{old} \end{aligned} \tag{8}$$

and improves the approximation error exponentially.[25] The number of desired power iterations is denoted by q. Power iterations are especially helpful for improving the forcing modes, which tend to converge more slowly than the response modes.

An orthogonal subspace $\mathbf{Q}$ is obtained from the sketch of $\mathbf{R}$ via the QR factorization

$$\mathbf{Y} = \mathbf{Q}\mathbf{T}, \tag{9}$$

where $\mathbf{T}$ is an upper triangular matrix. The approximation of $\mathbf{R}$ can then be written as

$$\mathbf{R} \approx \mathbf{Q}\mathbf{Q}^*\mathbf{R} = \mathbf{Q}\mathbf{S}, \tag{10}$$

where

$$\mathbf{S} = \mathbf{Q}^*\mathbf{R}. \tag{11}$$

Finding the SVD of $\mathbf{S}$ is inexpensive due to it reduced dimension and can be written as

$$\mathbf{S} = \tilde{\mathbf{U}}\mathbf{\Sigma}\mathbf{V}^*, \tag{12}$$

where $\tilde{\mathbf{U}}$ and $\mathbf{V}$ are the left and right singular vectors of $\mathbf{S}$, respectively, and $\mathbf{\Sigma}$ is the corresponding diagonal matrix of singular values. $\mathbf{V}$ and $\mathbf{\Sigma}$ estimate the k leading right singular vector and corresponding singular values of $\mathbf{R}$. From equations (10) and (12), the left singular vector of $\mathbf{R}$ can be recovered as

$$\mathbf{R} \approx \mathbf{QS} = (\mathbf{Q}\tilde{\mathbf{U}})\mathbf{\Sigma}\mathbf{V}^* = \mathbf{U}\mathbf{\Sigma}\mathbf{V}^*, \tag{13}$$

where $\mathbf{U} = \mathbf{Q}\tilde{\mathbf{U}}$ contains the estimates of the k leading left singular vectors of $\mathbf{R}$.

C. Computing resolvent modes using RSVD

Direct application of the RSVD algorithm outlined above to the resolvent operator of high-dimensional systems is impractical since $\mathbf{R}$ is the inverse of a large matrix and computing it quickly becomes intractable. In other words, $\mathbf{R}$ is not directly accessible. A modified version of RSVD is provided by Ribeiro *et al.*[20] to address this issue. In what follows, the steps of RSVD that need modification are identified and a remedy to these bottlenecks is proposed.

To obtain the sketch of the resolvent operator from equation (7), first, $\mathbf{R}$ has to be computed from equation (5) which is an expensive operation for large systems. So instead, the sketch of the resolvent operator can be found by solving the linear system[29,30]

$$\mathbf{LY} = \mathbf{\Theta}, \tag{14}$$

where $\mathbf{L} = \mathbf{R}^{-1} = i\omega\mathbf{I} - \mathbf{A}$. Similarly, $\mathbf{S}$ in (11) is also replaced by solving the linear system

$$\mathbf{SL} = \mathbf{Q}^*. \tag{15}$$

These two modifications avoid the need to explicitly form $\mathbf{R}$, but solving the linear systems in equations (14) and (15) remain the most expensive steps in the RSVD algorithm. The cost of solving these linear systems can typically be reduced by first computing the LU decomposition of $\mathbf{L}$, which can then be used to solve by equations (14) and (15).[20] The dominant cost within the RSVD algorithm then becomes finding the LU decomposition of $\mathbf{L}$, which constitutes a major bottleneck in the application of RSVD to resolvent analysis for large systems, especially 3D ones. Using this approach, Ribeiro *et al.*[20] achieved an order of magnitude reduction in CPU time compared to an iteratively restarted Arnoldi method for computing resolvent modes of a flow over an airfoil, but the memory savings were more modest (less than a factor of three). The computational time could be further reduced via parallelization, but this does not reduce the memory usage, which becomes the primary bottleneck for large problems.

In the remainder of this paper, we will refer to this modified procedure compatible with resolvent analysis simply as RSVD. Our proposed algorithm replaces the expensive steps of computing LU decomposition of $\mathbf{L}$ and solving the linear systems with alternative steps leveraging time-stepping techniques.

D. Steady-state response method (SSRM)

As discussed above, solving linear systems whose solutions give the action of the resolvent operator on a vector are the most expensive steps in the RSVD algorithm. Following Martini *et al.*,[16] we leverage the fact that the resolvent operator, unlike a more general matrix to which RSVD could be applied, is intimately connected to a linear differential equation in the time domain. Specifically, using time-stepping of direct and adjoint linear systems, we eliminate the expensive steps within the RSVD algorithm, as described below.

Consider the linear system with harmonic forcing

$$\frac{d\mathbf{q}}{dt} = \mathbf{Aq} + \mathbf{v}e^{i\omega t}, \tag{16}$$

where $\omega \in \mathbb{R}$ is an arbitrary frequency, and $\mathbf{v}$ is an arbitrary forcing vector. The frequency domain counterpart of the steady-state solution of equation (16) is equivalent to finding $\hat{\mathbf{q}}(\omega)$ directly in Fourier space, $\hat{\mathbf{q}}(\omega) = (i\omega\mathbf{I} - \mathbf{A})^{-1}\hat{\mathbf{v}}(\omega)$. Hence, solving the direct linear system (16) in the time domain is a surrogate for finding action of $\mathbf{R}$ on an arbitrary forcing vector $\mathbf{v}$.

Direct marching, however, is insufficient to find $\mathbf{S}$. Re-writing $\mathbf{S} = (\mathbf{R}^*\mathbf{Q})^*$ suggests *adjoint marching* as the equivalent time-domain approach. Consider the adjoint linear system

$$-\frac{d\mathbf{q}}{dt} = \mathbf{A}^*\mathbf{q} + \mathbf{v}e^{i\omega t}. \tag{17}$$

Similar to direct marching, the frequency domain counterpart of the steady-state solution in equation (17) is equivalent to finding $\hat{\mathbf{q}}(\omega) = \mathbf{R}^*(\omega)\hat{\mathbf{v}}(\omega)$, where $\mathbf{R}^*(\omega) = ((i\omega\mathbf{I} - \mathbf{A})^{-1})^* = (-i\omega\mathbf{I} - \mathbf{A}^*)^{-1}$. The adjoint marching is a surrogate for the action of $\mathbf{R}^*$ and it terminates the second and the last bottleneck in the RSVD algorithm. Note that both *direct* and *adjoint equations* are the same throughout the algorithm and only the forcing varies.

Following SSRM introduced by Martini *et al.*,[16] the actions of $\mathbf{R}$ and $\mathbf{R}^*$ can be obtained for a range of frequencies at the same time by applying forcing vectors comprised of all harmonics[31] and taking a subsequent Fourier transform to isolate individual frequencies. Consider the *desired* frequency set as harmonics of ω_{min}, $\Omega = \{\omega_1, \omega_2, ..., \omega_{N_\omega}\} = \{c\omega_{min}\}$, where N_ω is the total number of desired frequencies, $c \in \mathbb{Z}$ ranging from $-c_r$ to c_r, and $c_r = \frac{\omega_{max}}{\omega_{min}}$ is the ratio between the minimum frequency to be resolved and the maximum frequency. We can write down the forcing as

$$\mathbf{v}(t) = \sum_{j=1}^{N_\omega} \hat{\mathbf{v}}_j e^{i\omega_j t} + c.c., \tag{18}$$

where c.c. stands for complex conjugate. Integrating direct and adjoint equations with the forcing defined in equation (18) enables capturing the actions of $\mathbf{R}$ and $\mathbf{R}^*$, respectively, for all $\omega \in \Omega$ at the same time.

III. Method: combining RSVD and time-stepping

Our new algorithm combines RSVD with the SSRM time-stepping surrogates for obtaining the actions of $\mathbf{R}$ and $\mathbf{R}^*$. The steps of this algorithm are summarized in Algorithm 1. Note that since each step of the new algorithm functions identically to corresponding RSVD operations, the final outcome of both algorithms is *theoretically* the same. The time integration, however, must be performed numerically, leading to numerical errors (*e.g.* round-off error, truncation error, etc.). Overall, our approach provides a flexible environment (via a set of user-defined parameters) and one could achieve less accurate results at reduced cost by cutting the calculations and vice versa. These trade-offs are discussed in Section IV.

A. Initial forcing

Obtaining a steady-state solution is necessary for SSRM and it requires the transient response to vanishes. A random initial condition (usually set to zero) introduces a transient response into the solution of the initial value problem (IVP). The effect of this undesirable response, however, decays as time evolves. An estimate of the transient length, i.e., the time T_t it takes for the transient to decay to some desired threshold ϵ_t, can be obtained from a one-time numerical integration of equation

$$\frac{d\mathbf{q}}{dt} = \mathbf{A}\mathbf{q} \tag{19}$$

with a normalized random initial condition. The transient time T_t increases with decreasing ϵ_t.

The initial forcing for our algorithm is constructed via

$$\mathbf{\Theta}(t) = \sum_{j=1}^{N_\omega} \hat{\mathbf{\Theta}}_j e^{i\omega_j t} + c.c., \tag{20}$$

where $\hat{\mathbf{\Theta}}_j$ is taken from a normal distribution. This initial forcing contains all of the spatially random matrices, $\hat{\mathbf{\Theta}}_j(\omega)$, associated with $\omega \in \Omega$. Hence, the forcing is excited only at N_ω desired frequencies corresponding to random matrices defined for the same frequencies for RSVD. Given $\mathbf{\Theta}(t)$, the direct equation (16) can now be integrated. The length of integration should be $T_t + T$, where the period of steady-state forcing is pre-determined by $T = \frac{2\pi}{\omega_{min}}$. The steady-state response should preserve the excitation periodicity to resolve ω_{min}, thus $T = dt \times N_t$, where dt is the time step, and N_t is the total number of time-steps/snapshots.

B. Streaming Fourier sums

A T-periodic block of time data containing the steady-state response to $\mathbf{\Theta}(t)$ is used for acquiring $\hat{\mathbf{Y}}(\omega)$ for all $\omega \in \Omega$. The time marching of equation (16) requires N_t snapshots of forcing to produce N_t snapshots of steady-state solution in the time domain but only N_ω Fourier modes need to be computed. Usually N_t is much larger than N_ω to ensure stable and accurate integration of the direct and adjoint systems. Indeed, all N_t time instants could be used to calculate N_ω Fourier series coefficients, but this is expensive and unnecessary. In fact, twice the number of desired frequencies is the minimum number of snapshots required to recover the frequencies of interest without facing leakage and aliasing. Since Ω is defined to contain both positive and negative integer factors of ω_{min}, N_ω is the minimum number of temporal snapshots. New snapshots of size $N \times k$ are produced at every time step, yielding a total number of N_t snapshots as the steady-state response. N_ω out of N_t snapshots are sufficient for a fast Fourier transform (FFT) to obtain $\hat{\mathbf{Y}}(\omega)$ for all $\omega \in \Omega$. This reduces the memory usage, but the number of forcing snapshots remains large. Memory challenges motivate the idea of streaming calculations that compute the Fourier transform on the fly following the discrete Fourier transform (DFT) formulation, eliminating the memory demand for storing N_t forcing snapshots in the time domain.

The streaming procedure is conceptually similar to the streaming SPOD algorithm proposed by Schmidt & Towne.[32] In both the direct_marching and adjoint_marching functions in Algorithm 1, the Fourier modes of the appropriate forcing are passed in and the output is the response in Fourier space. Critically, neither forcing nor the responses in the time domain are stored in memory. Instead, forcing snapshots are generated one at a time as needed during the time marching, and the effect of the most recent steady-state response on the ensemble of Fourier modes is computed. The procedure in both functions is similar and is illustrated as follows.

The forcing in the time domain can be reconstructed from the Fourier modes of the forcing as

$$\mathbf{f}_p = \sum_{s=1}^{N_\omega} Z_{ps} \hat{\mathbf{f}}_s, \tag{21}$$

where $Z_{ps} = exp(-2\pi i/N_t)^{(p-1)(s-1)}$, $1 \leq p \leq N_t$, and s cycles through the frequencies $\omega \in \Omega$. $\hat{\mathbf{f}}_s$ of size $N \times k \times N_\omega$ denotes Fourier modes that are stored in memory and available. On the other hand, $\mathbf{f}_p$ is the p^{th} time-domain snapshot of the forcing used to obtain $(p+1)^{th}$ response. The process of generating a new forcing continues in a loop of size N_t all the way through the transient and steady-state responses.

Each snapshot of the time-domain solution, via a similar approach, contributes a portion of each Fourier modes by the partial sum

$$\left[\hat{\mathbf{q}}_s\right]_{N_p} = \left[\hat{\mathbf{q}}_s\right]_{N_{p-1}} + Z'_{N_p s}\mathbf{q}_{N_p} = \sum_{l=1}^{N_p} Z'_{ls}\mathbf{q}_l, \tag{22}$$

where $Z'_{ls} = exp(-2\pi i/N_\omega)^{(l-1)(s-1)}$, and $1 \leq l, s \leq N_\omega$. The complete Fourier modes are recovered once the effect of the final steady-state response is found ($N_p = N_\omega$). Notice that the size of the inverse DFT matrix, Z, is different from the DFT matrix, Z', since only N_ω snapshots are required to achieve the response Fourier modes as explained earlier. The streaming procedure for the adjoint equations is analogous to one described above for the direct equations, except that the adjoint equations must be solved backward in time and the identity of the input forcing and response is reversed.

To summarize, as no time snapshots are stored in the streaming procedure, the total required memory is reduced. The trade-off is the computational cost; the streaming DFT scales with $O(N_\omega^2)$ while FFT scales with $O(N_\omega \log(N_\omega))$. However, the cost of converting forcing and responses from/to the time domain to/from Fourier space is a small fraction of the total time-marching cost. So, on balance, the substantial memory benefits outweigh the cost difference.

C. The complete algorithm

The steps of Algorithm 1 can now be stated as follows. The first step is to define spatially random matrices for all $\omega \in \Omega$ (step 4). Next, the Fourier modes $\hat{\mathbf{Y}}$ are computed via streaming computation of the response to the given $\hat{\mathbf{\Theta}}$ by integrating the direct equation (step 5). The orthogonal subspace is immediately constructed for each individual desired frequency by QR decomposition of the corresponding coefficient (step 8). Note that $(\cdot)_\Omega$ indicates that a function is repeated for all $\omega \in \Omega$. To obtain $\hat{\mathbf{S}}$, the adjoint equation should be solved in time for the time domain counterpart of $\hat{\mathbf{Q}}$ which is the input forcing (step 9). This process also uses the streaming calculations. The last two steps (10 and 11) are decomposing $\hat{\mathbf{S}}$ and recovering $\hat{\mathbf{U}}$ for each individual frequency. These steps in our algorithm are identical to RSVD operations.

In addition to the ability to choose the desired trade-off between accuracy and speed, our algorithm benefits from all applicable improvements for stability and accuracy of RSVD such as power iteration and oversampling.[33–35] Note that power iteration as performed in (8) equivalently requires an adjoint and direct marching in a row. So, our algorithm is capable of handling this optional step by employing time-stepping, however, this is the most time-consuming step in Algorithm 1. The power iteration function (PI) in our approach follows the operations as written in Algorithm 2. When $k = 1$ and $q > 0$, the SSRM power-iteration method of Martini *et al.*[16] is recovered as a special case.

Our algorithm is based on RSVD, however, unlike the standard RSVD, creating an explicit matrix for the inverse of the resolvent operator, **L**, is unnecessary. Our approach, instead, finds the actions of **R** and $\mathbf{R}^*$ by carrying out direct and adjoint time integrations, respectively, providing a unique opportunity when an explicit operator for a linearized system is unavailable. Efficient ways have been devised to obtain these actions by exercising perturbation on the NS equations[36] without the need for explicit linearization as in equation (2). In addition, our algorithm is compatible with any time integration method. Therefore, the resolvent modes could be captured by combining the time marching results (as a surrogate for the actions of **R** and $\mathbf{R}^*$) with other steps defined in Algorithm 1.

Algorithm 1 Efficient randomized time domain algorithm based on SSRM

```
 1: MS: Marching scheme (e.g. Crank–Nicolson, Backward Euler, etc.)
 2: Input parameters: $\mathbf{A}, k$, MS, $dt, N_t, T_t, \Omega, N_\omega, q$
 3: procedure                                                              ▷
 4:     $\hat{\mathbf{\Theta}} \leftarrow \text{randn}(N, k, N_\omega)$          ▷ Create initial random forcing in frequency space
 5:     $\hat{\mathbf{Y}} \leftarrow \text{direct\_marching}(\mathbf{A}, \text{MS}, \hat{\mathbf{\Theta}}, Z, Z', dt, N_t, T_t)$   ▷ Solve direct equation on-the-fly for $\hat{\mathbf{Y}}$
 6:     if q > 0 then                                                      ▷ Power iteration to improve RSVD accuracy
 7:         $\hat{\mathbf{Y}} \leftarrow \text{PI}(\Omega, k, \text{MS}, dt, N_t, T_t, q)$
 8:     $\hat{\mathbf{Q}} \leftarrow \text{qr}_\Omega(\hat{\mathbf{Y}})$          ▷ Construct the low-rank orthonormal space $\hat{\mathbf{Q}}$
 9:     $\hat{\mathbf{S}} \leftarrow \text{adjoint\_marching}(\mathbf{A}^*, \text{MS}, \hat{\mathbf{Q}}, Z, Z', dt, N_t, T_t)$   ▷ Solve adjoint equation on-the-fly for $\hat{\mathbf{S}}$
10:     $(\tilde{\mathbf{U}}, \mathbf{\Sigma}, \mathbf{V}) \leftarrow \text{svd}_\Omega(\hat{\mathbf{S}})$   ▷ Obtain SVD of $\hat{\mathbf{S}}$
11:     $\mathbf{U} \leftarrow \hat{\mathbf{Q}}\tilde{\mathbf{U}}$                ▷ Recover $\mathbf{U}$
12: Final output: $\mathbf{U}, \mathbf{\Sigma}, \mathbf{V}$ for each $\omega \in \Omega$
```

Algorithm 2 Power iteration (PI) based on SSRM

```
1: MS: marching scheme (e.g. Crank–Nicolson, Backward Euler, etc.)
2: function PI($\mathbf{A}, \hat{\mathbf{Y}}$, MS, $dt, N_t, T_t, \Omega, N_\omega, q$)
3:     for i = 1:q do
4:         $\hat{\mathbf{Y}} \leftarrow \text{qr}_\Omega(\hat{\mathbf{Y}})$          ▷ For stabilization purposes[25]
5:         $\hat{\mathbf{Y}} \leftarrow \text{adjoint\_marching}(\mathbf{A}^*, \text{MS}, \hat{\mathbf{Y}}, Z, Z', dt, N_t, T_t)$   ▷ Solve adjoint equation on-the-fly for $\hat{\mathbf{Y}}$
6:         $\hat{\mathbf{Y}} \leftarrow \text{qr}_\Omega(\hat{\mathbf{Y}})$          ▷ For stabilization purposes[25]
7:         $\hat{\mathbf{Y}} \leftarrow \text{direct\_marching}(\mathbf{A}, \text{MS}, \hat{\mathbf{Y}}, Z, Z', dt, N_t, T_t)$   ▷ Solve direct equation on-the-fly for $\hat{\mathbf{Y}}$
8: Final output: $\hat{\mathbf{Y}}$
```

D. Computational cost and memory usage

The ability to estimate the total cost of our algorithm with high confidence before running the entire algorithm is one of its features. The main steps of the algorithm include the DFT and inverse DFT, time integration, SVD, and QR decompositions. Among these, the most cost-demanding function for large stiff systems (>95% of total cost for most systems we have tested) is the time integration. The following explains how to estimate the CPU cost and memory requirements for our approach and compares it with the standard RSVD.

After discretizing the linearized NS equations in time, the general form of the obtained system looks like

$$\mathcal{A}\mathbf{x} = \mathbf{b}, \tag{23}$$

where $\mathbf{b} \in \mathbb{C}^{N\times k}$ is the forcing that varies in time. $\mathcal{A} \in \mathbb{C}^{N\times N}$ is the temporal discretized operator and can be written as a first-order polynomial like $P(\mathbf{A}) = \mathcal{A} = c_1\mathbf{I} + c_2\mathbf{A}$, where constants are determined based on integration scheme and time step. For instance, the Backward (implicit) Euler method results in $\mathcal{A} = \mathbf{I} - dt\mathbf{A}$. Direct and iterative solvers could be used to solve equation (23) depending on conditioning of the system, desired accuracy, cost and memory allocations.

When using a direct solver, since $\mathcal{A}$ remains the same throughout the entire integration, LU decomposition leads to a significant speedup. The nominal cost of LU decomposition (T_{LU}) of a dense matrix is $O(N^3)$, while the cost of solving LU-decomposed system (T_{sub}) scales with $O(N^2)$. These scalings can be much less for a sparse system depending on the size, sparsity ratio, and sparsity pattern of the system.[37] As $\mathcal{A}$ is usually sparse, the cost of direct solution of equation (23) scales theoretically like $O(N^{1.5})$ for 2D problems and $O(N^2)$ for 3D problems using multi-frontal sparse direct solvers according to analysis by Amestoy *et al.*[38] The total CPU time can be written as $T_{LU} + N_T \times T_{\text{sub}}$, where N_T is the number of time steps (including the transient and steady-state periods). On the other hand, cost of performing adjoint marching is $N_T \times T_{\text{sub}}$, where T_{LU} is dropped due to the fact that the conjugate transpose of the decomposed matrices are sufficient for solving $\mathcal{A}^*\mathbf{x} = \mathbf{b}$. Therefore, the entire computational cost of our algorithm is roughly $T_{LU} + 2 \times N_T \times T_{\text{sub}}$. Performing q power iteration, the total CPU time becomes $T_{LU} + 2 \times N_T \times T_{\text{sub}} + q \times 2 \times N_T \times T_{\text{sub}} = T_{LU} + 2 \times (q+1) \times N_T \times T_{\text{sub}}$

The most costly step in RSVD is also solving linear systems such as equations (14) and (15). Again, LU decomposition is performed first, and the LU-decomposed system is then solved twice. The overall cost of RSVD really depends on LU decomposition of the $\mathbf{L}$ for each frequency of interest, which typically differ. However, a rough CPU time estimate for N_ω frequencies of interest and q power iteration is $N_\omega \times (T'_{LU} + 2 \times (q+1) \times T'_{\text{sub}})$. Two key differences that distinguish our algorithm from RSVD are *(i)* we require the LU decomposition of a better conditioned matrix and *(ii)* we must perform this LU decomposition just once, rather than once for every frequency of interest. The LU decomposition in our algorithm is performed on a time discretized system, $\mathcal{A}$, as opposed to the inverse of the resolvent operator, $\mathbf{L}$. $\mathcal{A}$ is usually much better conditioned than $\mathbf{L}$ and ill-conditioning causes a noticeable increase in the CPU time and memory cost of the LU decomposition. Hence, for large 3D problems $T'_{LU} \gg T_{LU}$ is expected. The second important difference is that our algorithm requires LU decomposition once. However, RSVD requires this costliest operation once for each frequency of interest. Therefore, even by assuming $T_{LU} \approx T'_{LU}$, our algorithm is N_ω times faster for the decomposition stage. Now, all computational benefit from our algorithm relies on the ratio between T_{LU} and T_{sub}. The larger gap between these two would increase the computational advantage of our algorithm. T_{LU} usually scales worse with dimension than T_{sub}, resulting in $T_{LU} \gg T_{\text{sub}}$ for large problems which makes our algorithm advantageous for high-dimensional systems.

The second option to obtain the solution of equation (23) is to use an iterative solver. Numerous studies have developed advanced and fast iterative solvers.[39,40] The condition number difference between the two linear systems makes using an iterative solver for RSVD more challenging than applying it to equation (23). In fact, bringing down both computational time and memory allocation at the same time when dealing with a poorly conditioned matrix could be cumbersome. Solving equation (23) with an iterative solver eliminates the prerequisite of computing LU decomposition matrices as in the direct solver process. This can significantly decrease the memory requirement and CPU cost associated with both decomposition and integration, but the trade-off is the accuracy. Iterative solvers such as generalized minimal residual method (GMRES), stabilized bi-conjugate gradient (BICGS), etc. compute the solution of a linear system up to a specified relative residual norm. The convergence becomes more expensive as this norm is set to lower values. The key initial step before using an iterative solver is to build a good preconditioner. The main objective of applying a preconditioner is reducing the condition number (which accelerates the convergence rate) and many preconditioners have been developed for various systems.[41,42] Assuming T_{pre} is the CPU cost of finding a preconditioner and T_{it} is the cost of iteratively solving the linear system, the total CPU time of both direct and adjoint marching plus q power iterations becomes $T_{\text{pre}} + 2 \times (q+1) \times N_T \times T_{it}$. Only once computing a preconditioner should be enough when it is found via incomplete LU (iLU), decomposition since the transpose conjugate of iLU for the direct marching provides a decent preconditioner for the adjoint marching.

The most memory-demanding step in solving linear systems is also the LU decomposition. The peak of memory consumption depends on many factors including the sparsity pattern and the condition number of a matrix. An ill-conditioned problem requires more memory storage than a well-conditioned system with a similar size. Therefore, our new algorithm that deals with operators of the form $\mathcal{A}$ requires less memory allocation for LU decomposition. The memory savings become larger when an iterative solver is used. The preprocessing step is finding iLU or other preconditioners that usually involves a very sparse matrix and do not require significant memory. The expected memory demand for each iteration, also, would not drastically deviate from the memory needed for solving an LU-decomposed system.

The approximation of the computational wall time of our algorithm is reliable due to the fact that all steps remain roughly the same cost throughout the integration. This, however, is not true for RSVD. The cost of LU decomposition of the resolvent inverse matrix can vary significantly depending on the frequency of interest. Specifically, when ω is small (close to zero), the condition number of the operator **A** is dominant which leads to an extensive amount of calculations. The condition number gets closer to 1 (identity matrix condition number) as higher frequencies are desired and the LU decomposition expenses slightly fluctuate over an average value for a sufficiently large ω range.

E. Parameter selection for the new algorithm

The overall accuracy, CPU cost, and memory consumption of the proposed algorithm, specifically for high-dimensional systems, are determined by the time integration function that carries out the direct and adjoint actions. The optimal way of numerically integrating the linearized NS equations depends on many factors including the problem size, geometry, discretization scheme, Reynolds number, etc. Nevertheless, some general guidelines are herein provided for typical systems. We particularly explain the process of the time evolution of large (and usually stiff) ODEs to highlight the cost and memory requirements.

1. Transient effects

Even under harmonic forcing, the solution of a linear IVP has a transient component if the initial condition consistent with the harmonic response is unknown. As mentioned, the steady-state solutions of equations (16) and (17) are required to find the actions of $\mathbf{R}$ and $\mathbf{R}^*$, respectively. Therefore, two sources of error contaminate the steady-state response to a given forcing; truncation error due to time-stepping and transient effects. The transient solution decays as time evolves and the transient effect becomes less intense. A good practice for choosing the length of the transient part is to make sure the error of the transient response is roughly the same magnitude as the truncation error since the error of the final solution is just a sum of two sources of errors.

As discussed in Section III.A, to find an approximation of how transient effects decay in time, the linear system (19) can be integrated once (transient run). The input is a normalized random initial condition with no exogenous forcing. This preprocessing step is a low-cost operation compared to running the entire algorithm. The solution could be found only for one test vector, but adding a few more ($\sim$3) provides a better estimate.

To further decrease the cost associated with removing the transient, we propose to divide the transient time into several sections, or levels, that are each integrated with time steps of different sizes. The idea here is to exploit the fact that the transient solution does not need to be accurately resolved and our algorithm only needs to remove its effect up to a specific threshold. Accordingly, the transient can be passed more quickly with larger time steps. A large time step can effectively remove the transient as time evolves until the truncation error due to time-stepping becomes dominant. Then switching to a smaller time step will continue removing the transient. This process repeats and is illustrated as follows.

Assume T is the duration of the steady-state solution to resolve all frequencies and T_t is the transient duration to remove the transient solution up to some desired threshold ϵ_t. The original version of our algorithm requires $(T_t + T)/dt_s$ operations, where dt_s is the small enough to reach ϵ_t. Our *multilevel* idea suggests splitting the transient part into smaller levels with different time steps sizes to reduce the number of operations and achieve computational savings. At each level, dt_l will be used for some duration T_l and the final solution will be passed to the next level. The initial condition from one level to the other generates a transient solution since the time step has been changed. After integrating for long enough such that the transient error is smaller than the truncation error, the solution will be passed to the next level. The period of integration at each level depends on the time step of two successive levels and the decay rate of the least damped eigenvalue of **A**. As shown in figure 1, repeating this process, the transient duration can be partitioned into m smaller levels $T_1, T_2, ..., T_m$ associated with $dt_1, dt_2, ..., dt_m$ where $dt_1 = dt_s$. The modified number of operations is now $T_m/dt_m + T_{m-1}/dt_{m-1} + ... + T_1/dt_1 + T/dt_s$.

At the end of each level, the final time instance is passed as an initial condition to the next level with an updated time step. To minimize the effect of the transient solution associated with a wrong initial condition, the forcing and the initial condition must be synchronized, i.e. the initial condition and the forcing have to be on the same time point. This can be easily achieved by assuming $dt_i = n_i dt_s$ where n_i is an integer. Other arbitrary values for dt_i are possible in case synchronization is taken care of. Another note here is that passing the whole transient using a very large time step and directly jumping to steady-state time period is not effective since the truncation error becomes dominant. The best practice is to integrate long enough such that the transient effect become close to the truncation error at each level.

One can see that this idea only affects the transient part and the steady solution is intact. The maximum ratio of cost reduction is always less than $T/(T_t + T)$ (when ideally the transient cost completely fades away). As a result, systems with slow transients (when $T_t \gg T$) will benefit more from this idea.

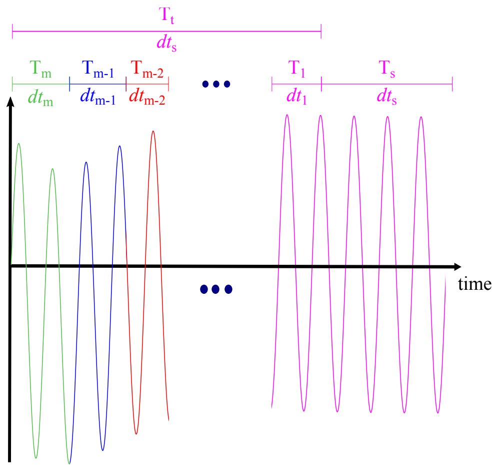

Fig. 1 A schematic of the multilevel idea. The number of operations required to pass through the transient time T_t is reduced by segmenting it into m levels with large time steps at the beginning of the transient time that are progressively reduced to the desired time step dt_s in the steady-state region.

2. Integration scheme and time step

The first choice to be made when selecting parameters for the new algorithm is the numerical integration scheme. Many numerical schemes are available and categorized as explicit, implicit, or a combination of both.[43] Some classic methods include one-step methods such as forward/backward Euler, multistage methods such as the Runge-Kutta family, and linear multistep methods (LMM) such as the backward differentiation formula (BDF) family.[44]

For the stiff systems typical of the linear NS equations, implicit LMMs offer an attractive combination of stability and accuracy using time steps of reasonable size. We also found LMMs straightforward to implement without requiring extra effort to incorporate the streaming calculations in the direct and adjoint functions. For LMMs, the zero-frequency (steady) response has zero truncation error, regardless of the linear system. The error caused by time marching is mainly due to transient effects for low frequencies and due to truncation error for higher frequencies. All classic marching methods are consistent and, if stable, converge to the exact solution with $O(h^p)$, where h is the time step and p is the order of the numerical scheme.

3. Effect of time integration error on the accuracy of the new algorithm

The accuracy of the estimated singular values and singular vectors from the RSVD method are sensitive to the properties of $\mathbf{R}$ and RSVD parameters such as k and q. Our algorithm only substitutes direct and adjoint actions and approximates RSVD outcome with some noise that time integration incorporates into the RSVD. The ultimate goal, however, is to approximate the optimal modes of the true SVD of the resolvent operator. The accuracy of our approach could be bounded by one of these two sources of error. The first one is the error of RSVD as it approximately captures the optimal modes through a randomized process. The upper bound of the computed modes from RSVD has been theoretically found by Halko *et al.*[25] The second source is due to time-stepping. This source of error could be dominant when the time-stepping error becomes larger than the RSVD approximation. This is true especially when power iteration is used.

Based on the SVD perturbation theory of Weyl,[45] the singular values of a matrix are bounded by

$$|\hat{\sigma}_i - \sigma_i| \leq ||\mathbf{E}||_2, \tag{24}$$

where $\mathbf{E}$ is the (matrix) perturbation to $\mathbf{R}$. Therefore, the absolute error between the computed singular values from our approach has an upper bound, but usually the computed modes (and specifically the optimal mode) are far more accurate than this upper bound. Better error bounds can be found for systems with large gain separations.[46]

Perturbation expansion also could be useful especially when $\mathbf{E}$ approaches zero.[45] Let $\sigma \neq 0$ be a simple singular value of $\mathbf{R}$ with left singular vector $\mathbf{u}$ and right singular vector $\mathbf{v}$. Then as $\mathbf{E}$ approaches zero, there is a unique singular value $\hat{\sigma}$ of the perturbed matrix $\mathbf{R} + \mathbf{E}$ such that

$$\hat{\sigma} = \sigma + \mathbf{u}^T \mathbf{E} \mathbf{v} + O\left(||\mathbf{E}||^2\right). \tag{25}$$

The perturbation theory for singular vectors is more complicated since singular vectors corresponding to close singular values can be sensitive. We will instead use an a posteriori measure of the error between two corresponding singular vectors of RSVD and the new algorithm, namely the cosine similarity

$$e_u = 1 - \langle \mathbf{u}_{RSVD}, \mathbf{u}_T \rangle, \tag{26}$$

where $\langle \cdot, \cdot \rangle$ denotes the inner product between two vectors. The subscripts T stands for our approach.

IV. Results

In this section, we analyze and demonstrate our algorithm by applying it to three example problems. First, we validate the method using a linearized Ginzburg-Landau problem via comparisons to results obtained from RVSD and a fully converged Arnoldi-based SVD routine. Then, we investigate the CPU cost and memory requirements of the method using axisymmetric and 3D jets.

A. Complex Ginzburg-Landau equation

The complex Ginzburg-Landau (GL) equation provides complex dynamics that mimic real flow behavior. In particular, the evolving spatial instabilities have made GL a benchmark problem for developing tools and assessing linear stability models.[47–51] The GL equation can be written in the form of equation (2) with

$$\mathbf{A} = -\nu \frac{\partial}{\partial x} + \gamma \frac{\partial^2}{\partial x^2} + \mu(x), \tag{27}$$

where $\mu(x) = (\mu_0 - c_\mu^2) + \frac{\mu_2}{2} x^2$. The free model parameters are set to identical values as used by Towne *et al.*,[12] where $\nu = 2 + 0.2i, \gamma = 1 - i, \mu_0 = 0.23, c_\mu = 0.2$, and $\mu_2 = -0.01$. These values ensure a globally stable model and provide a large gain separation between the leading resolvent mode and the second mode at their peak frequencies. The computational domain $x \in [-85, 85]$ is extended enough to mimic infinite boundaries. The spatial discretization uses central finite difference method. The dimension of our mesh, N, is 500 and the set of desired frequencies Ω consists of values between -4 and 4 with a resolution of $\Delta\omega = 0.05$. In the following, we evaluate the effect of various parameters on our proposed approach and validate our algorithm.

1. Resolvent modes obtained from the new algorithm

Before assessing the sensitivity of results obtained from our algorithm to parameter selection, we examine the resolvent modes of GL. The results in figure 2 compare the first three leading modes of SVD and our approach for various values of q and $k = 10$. We set the integration parameters as $dt = 0.1$, numerical scheme to 4^{th} order backward difference formula (BDF4), and $T_t = 80$. The only change is the number of power iterations. The error between the gains improves by a few orders of magnitude when power iteration is performed once. The improvement is greater at frequencies for which the gain separation is larger. Since the relative error between SVD and our approach becomes much smaller from $q = 0$ to $q = 1$ without changing the integration parameters, it suggests that the error of direct and adjoint marching within our approach is smaller than the error of RSVD approximations when $q = 0$.

The optimal mode obtained from RSVD usually has the smallest error, especially when the gain separation is large. This is evident in figure 2 (c), where the error gap between the optimal and suboptimal gains is larger at the peak frequency. Plot (d) also shows a similar trend but it seems the optimal gain error is limited by integration error as opposed to suboptimal gains. This happens when the RSVD approximation is so accurate that errors introduced by the direct and adjoint marching become dominant. Further comparison between the relative error between our approach and RSVD provides more information to interpret the dominant error.

2. Effect of numerical integration scheme and time step

Two purposes are pursued here. The first purpose is to explain the relative error between RSVD and our algorithm due to the truncation error, which depends on the integration scheme and time step. The second objective is to validate our algorithm by showing that the relative error between our approach and RSVD can be reduced to machine precision.

Three sets of parameters are considered for these purposes. The common parameters between RSVD and our algorithm are $k = 10$ and $q = 0$. The transient duration, $T_t = 280$, is long enough such that the effect of the transient vanishes up to machine precision. The time integration-related parameters include (i) BDF2 and $dt = 0.1$, (ii) BDF4

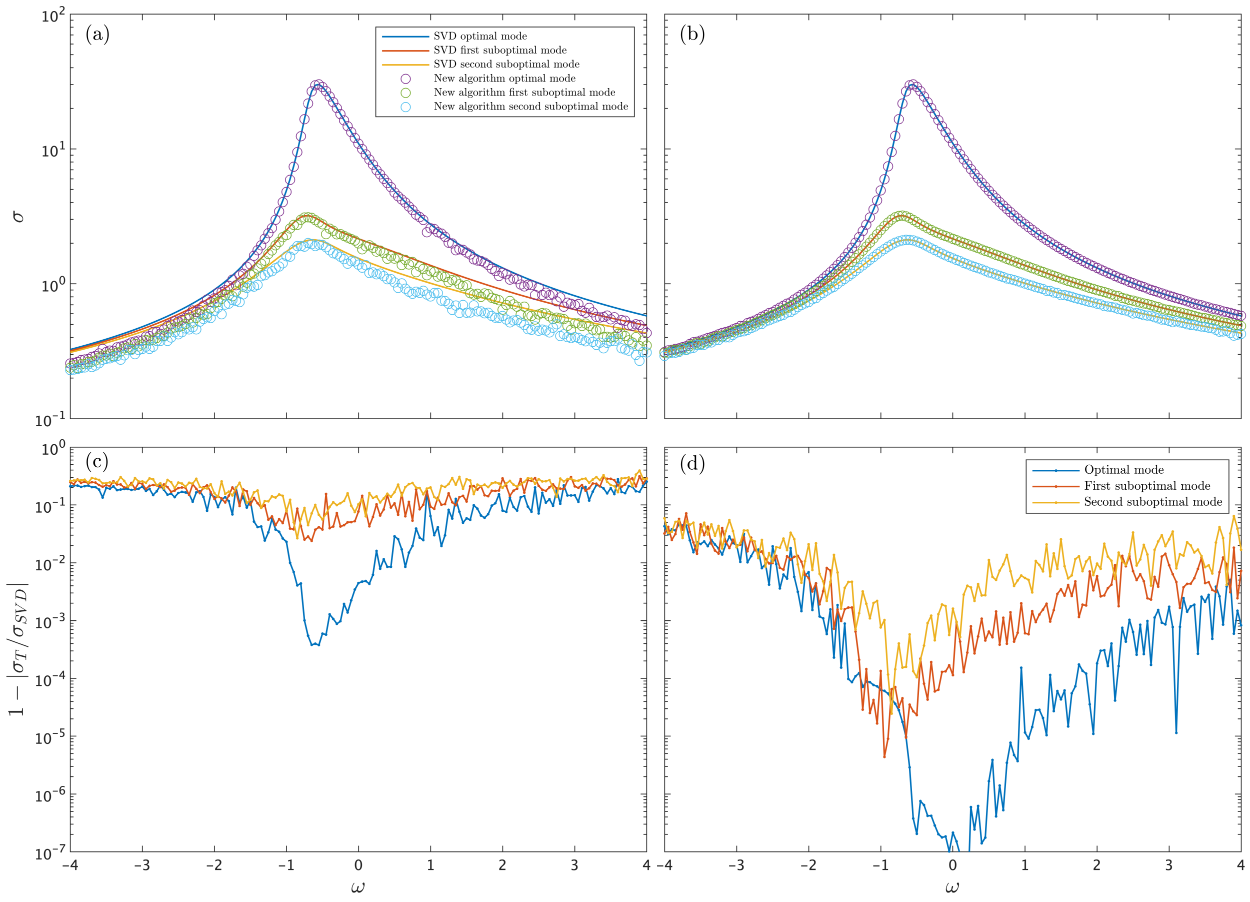

Fig. 2 The gains of the first three optimal modes of SVD and our algorithm and the relative error between them are shown in the first and second rows, respectively. The left and right columns correspond to $q = 0$ and $q = 1$, respectively. The parameters of time marching are the same in both simulations. The subscript T stands for our approach.

and $dt = 0.01$, and (iii) BDF6 and $dt = 0.001$. The initial random forcing is identical for both algorithms and all three cases. One extra set with BDF6 and $dt = 0.0005$ is also considered to demonstrate our validation is independent of chosen parameters.

Figure 3 shows the relative error of the first three optimal gains compared to SVD (top row) and RSVD (bottom row). The first row results show almost identical plots, which is expected since the second row error plots show smaller relative errors between RSVD and our approach. The time-stepping error is not dominant even for the least accurate set (i). A comparison between the second row error plots indicates that the integration error can be made as small as desired. Plot (f) reaches close to machine precision. In fact, it is slightly higher than the true machine precision $O(10^{-16})$ due to the accumulation of roundoff errors. Moreover, no noticeable change in the results corresponding to the extra set with finest $dt = 5 \times 10^{-4}$ is observed (results not shown), which shows the results are independent of the chosen parameters. These results imply that our proposed algorithm is capable of reproducing the exact outcome of RSVD for a careful set of time-stepping parameters, validating our algorithm. Also, they suggest that a very accurate integration is unnecessary and would not change the accuracy of the obtained resolvent modes when the RSVD approximation bears more error.

One observation here is that machine precision error is observed for $\omega = 0$, and larger errors are obtained for higher frequencies. As explained earlier, the low frequency error is dominant by transient effects (which are small since the time allowed for the transient to decay is long), and high frequency error is mainly due to the truncation error, which changes significantly for the different sets of parameters considered here. This observation is further evaluated in the next section.

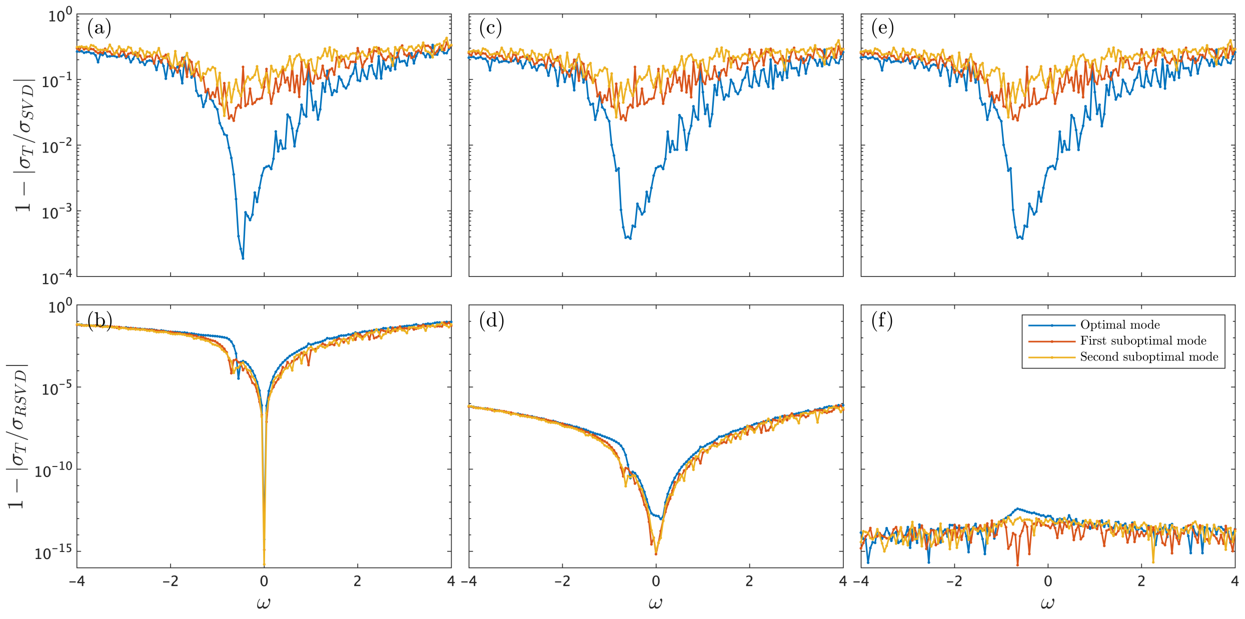

Fig. 3 Error in gains computed using our algorithm relative to SVD (top row) and RSVD (bottom row). Columns from left to right correspond to parameter sets (i), (ii), and (iii), as defined in the text.

3. Effect of the transient

Transient effects are defined as the response of the IVP to initialization when the initial condition consistent with the steady-state response is unknown. This is part of the error of direct and adjoint marching. We consider three new sets of parameters to assess the effect of the transient. The common parameters between RSVD and our algorithm are the same as in the last section. For the time integration-related parameters, BDF4 and $dt = 0.01$ are fixed and only the transient duration varies as $(iv)T_t = 40$, $(v)T_t = 100$, and $(vi)T_t = 280$. Note that the last set has exactly the same parameters as set (ii) in the previous section.

The transient disappears at different rates, depending on the position of the least damped eigenvalues of $\mathbf{A}$. In figure 4, plot (b) demonstrates that transient error can become larger than the truncation error even for higher frequencies. The transient error decays almost at the same rate for all modes for the GL system. This could vary from one system to another. Similar to the previous section, the first row results remain unchanged as the integration error is small enough. The second row error plots show that the error is dominated by the truncation error for higher frequencies and is determined by the transient effects for lower frequencies (close to $\omega = 0$). This is not a feature of the linear system, but instead, it is a feature of LMM scheme. In light of the competition between transient and truncation error, the efficiency of the algorithm in maximized when these two errors have similar magnitudes.

B. Axisymmetric jet

In this section, we consider one of the jet cases described by Schmidt *et al.*[52] In brief, a large-eddy simulation (LES) of a turbulent jet with Mach number $M = \frac{U}{a} = 0.4$ and Reynolds number $Re = \frac{UD}{\nu} = 10^6$ was performed on an unstructured grid using the 'Charles' solver developed by Cascade Technologies.[53] Here, a is the ambient speed of sound, ν is the kinematic viscosity at the nozzle exit, U is the centerline jet velocity, and D is the diameter of the nozzle. Temporal data was interpolated onto a cylindrical grid with $656 \times 138 \times 128$ points in $x \in [0, 30]$, $r \in [0, 6]$, $\theta \in [0, 2\pi]$, respectively. The frequency is reported in terms of the Strouhal number $St = \frac{\omega D}{2\pi U}$. For details on LES and numerical setup please refer to Brès *et al.*[54]

Since the jet is round, the mean flow is axisymmetric, depending only on streamwise and radial coordinates. The region of interest in this study is $x \times r \in [0, 10] \times [0, 3]$ surrounded by a sponge region. The full domain is spatially discretized using fourth-order central finite differences with 632×285 grid points in the streamwise and radial directions, respectively. The Reynolds number within the linearized system is reduced to $Re = 10,000$ to account for un-modeled Reynolds stresses.[55]

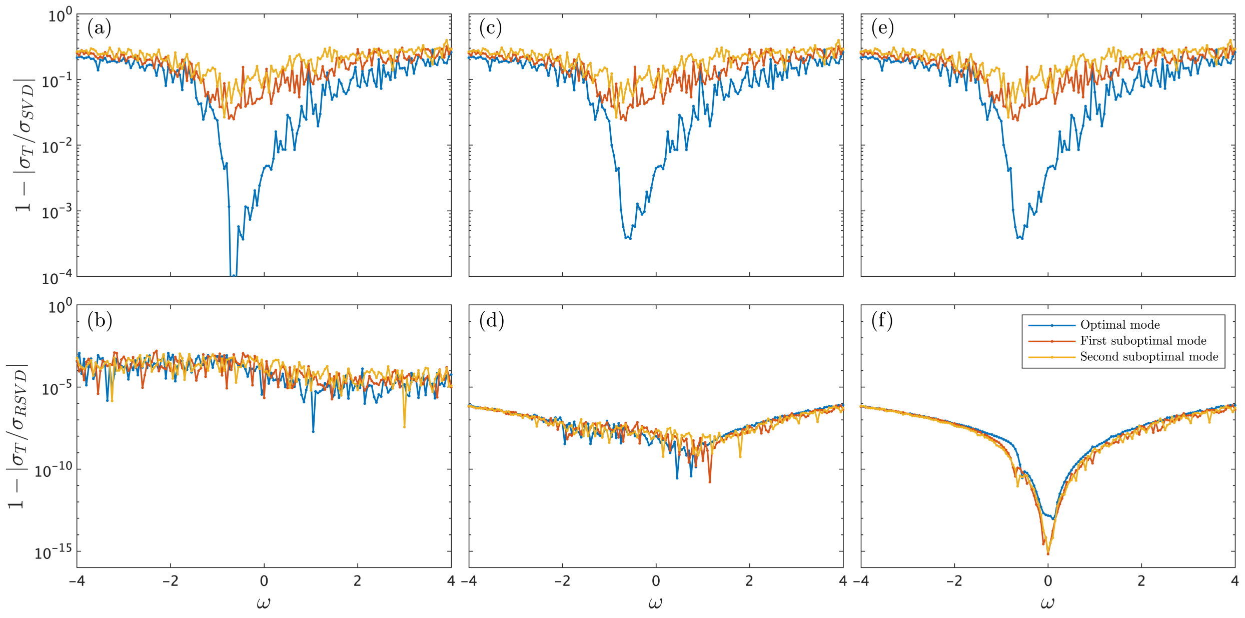

Fig. 4 Error in gains computed using our algorithm relative to SVD (top row) and RSVD (bottom row). Columns from left to right correspond to parameter sets (iv), (v), and (vi), as defined in the text.

The input-output system is derived by finding the Fourier-domain counterpart of the linearized NS equation (2). We write the equations in terms of specific volume, three components of velocity, and pressure, yielding the state vector $\mathbf{q}(\mathbf{x}, t) = (\xi, u_x, u_r, u_\theta, p)^T (x, r, \theta, t)$. The jet is statistically stationary and is periodic in the azimuthal direction, so, Fourier modes in azimuthal direction and time can be written as

$$\mathbf{q}'(x, r, \theta, t) = \sum_{m,\omega} \hat{\mathbf{q}}_{m,\omega}(x, r) e^{im\theta} e^{i\omega t}, \tag{28}$$

where m denotes azimuthal wavenumber and ω is the angular frequency. For a chosen m and ω, the relation between inputs and outputs is governed by

$$\hat{\mathbf{q}}_{m,\omega} = \mathbf{R}_{m,\omega} \hat{\mathbf{f}}_{m,\omega}, \tag{29}$$

where

$$\mathbf{R}_{m,\omega} = \mathbf{C}(-i\omega\mathbf{I} - \mathbf{A}_m)^{-1}\mathbf{B}, \tag{30}$$

and $\mathbf{B}$ and $\mathbf{C}$ matrices are spatial filters that determine the region of interest for output and input states, respectively. The sponge region is filtered out using appropriate $\mathbf{B}$ and $\mathbf{C}$.

Following Schmidt *et al.*,[30] a modified (weighted) resolvent operator is defined as

$$\tilde{\mathbf{R}}_{m,\omega} = \mathbf{W}_y^{1/2}\mathbf{C}(-i\omega\mathbf{I} - \mathbf{A}_m)^{-1}\mathbf{B}\mathbf{W}_f^{-1/2}, \tag{31}$$

where $\mathbf{W}_y$ and $\mathbf{W}_f$ are weight matrices that define the inner products for the output and input spaces and also account for numerical quadrature such that the inner product in the discretized system can be written as $\langle \mathbf{q}_y, \mathbf{q}'_y \rangle = \mathbf{q}_y^* \mathbf{W}_y \mathbf{q}'_y$. For simplicity, we use the weight matrices to account for quadrature weights but do not consider special weighting of different flow variables; this has no impact on our study since our aim is to compare algorithms rather than elucidate physics or build a model.

The resolvent operator in equation (31) is slightly different than the resolvent defined in equation (5). The $(-i\omega\mathbf{I} - \mathbf{A}_m)^{-1}$ term is still associated with time marching, however, the caveat here is to apply $\mathbf{B}\mathbf{W}_f^{-1/2}$ to the given forcing in the frequency domain and $\mathbf{W}_y^{1/2}\mathbf{C}$ to the Fourier counterpart of the time-stepping output. For instance, a single step of the streaming direct marching (step 5 in Algorithm 1) becomes three steps:

1) $\hat{\mathbf{Q}} \leftarrow \mathbf{B}\mathbf{W}_f^{-1/2}\hat{\mathbf{Q}}$

```
2) $\hat{\mathbf{S}} \leftarrow$ direct_marching($\mathbf{A}, \mathrm{MS}, \hat{\mathbf{\Theta}}, Z, Z', dt, N_t, T_t$)
3) $\hat{\mathbf{S}} \leftarrow \mathbf{W}_y^{1/2}\mathbf{C}\hat{\mathbf{S}}$
```

These modifications can be used to restrict the input or output or define arbitrary inner products for any linear system (not just the jet considered here). The only modification in Algorithm 1 is to update the input (forcing) with weight and filter matrices before passing it into the time marching function and also update the output (response) of the the same function with appropriate matrices, both in the Fourier space, as explained above. The cost of the first and third steps are negligible compared to the second step as long as weight and input/output matrices are sparse (they are typically diagonal in practice).

1. *Resolvent modes obtained by RSVD and the new algorithm*

The linearized NS system for the jet is stiff, so we choose the time integration scheme to be one step Adams–Moulton (AM1), which is the most accurate second order A-stable LMM. A transient run suggested a $T_t = 500$ transient duration, and to reduce the cost of transient, the multilevel method is used with $dt = [16, 8, 4, 2, 1] \times 0.05$ with $T_t = [500, 20, 20, 20, 20]$. This reduces the transient number of operations from $500/0.05 = 10,000$ to $500/(16 \times 0.05) + 20/(8 \times 0.05) + 20/(4 \times 0.05) + 20/(2 \times 0.05) + 20/(1 \times 0.05) = 1,375$. This is a 86.25% reduction which means that we pass the transient almost 7 times faster. Ω consists of $St \in [-2, 2]$ with a resolution of $\Delta St = 0.05$. The iterative solver is conjugate gradients squared (CGS) with iLU preconditioner with drop tolerance of 10^{-3}. The residual norm for CGS is set to 10^{-4}. The RSVD parameters are $k = 6$ and $q = 1$.

The integration scheme has LMM form, so the error of zero frequency depends only on the transient effects. The slowest decaying mode for this problem has zero frequency; all modes associated with higher frequencies decay much faster for the same transient duration, so the dominant error at these frequencies is due to truncation. The error plots in figure 5 between singular vectors and singular values confirm that the truncation error has similar magnitude as the transient error. Further extension of the transient run (> 500) only affects the error of low frequencies ($-0.2 \le St \le 0.2$). The captured gains and singular vectors from our algorithm maintain a relatively low error compared to the RSVD outcome for all frequencies and all modes.

Excellent visual agreement is observed between the modes obtained using our algorithm and RSVD in figure 6 which shows the real part of the pressure for the first three modes. Other states such as streamwise and radial velocities obtained from our approach are also very close to RSVD outcome. The results could be further improved by decreasing the time step until the accuracy is limited by the RSVD approximation. This will indeed increase the computational cost but the memory usage remains the same.

2. *Computational cost and memory scaling*

The CPU cost and memory requirements for various functions within the RSVD and new algorithms are computed for a range of problem dimension N to better demonstrate their scaling. The base state dimension is $N_B = 5 \times N_x \times N_r = 5 \times 200 \times 90$, which might not be resolved enough to capture the resolvent modes properly. However, the purpose of this section is to estimate the computational expenses for mid-size and larger problems. Finer discretizations are used to create larger systems of size $N = G \times N_B = 5 \times (\sqrt{G} \times N_x) \times (\sqrt{G} \times N_r)$, where G varies between 1 and 25. All other parameters are held constant.

LU decomposition is performed using a nonsymmetric pivoting strategy and permutation (exchanging rows and columns) is allowed for faster calculation, increasing numerical stability and obtaining sparser L and U with minimum fill-ins. All the functions are well suited for multithreaded computations, however, the number of processors on our high-memory workstation is restricted to one to measure the wall time. This ensures an even comparison between all the operations. To further reduce the uncertainties associated with wall time measurements, the elapsed times are reported based on averages taken over 2-50 repetitions of a function. Since we are using a single processor, wall time is a surrogate for counting the FLOPS. The quantitative times could vary for other processors, but scalings are expected to be preserve.

Comparing the results of figure 7 (a) and (c), the computational cost and memory usage of LU decomposition follow similar trends. The CPU cost scales like $O(N^{2.5})$ and memory like $O(N^{1.8})$ when $St = 0$, worse than all the other functions as expected. The LU decomposition cost scaling varies between $O(N^{1.8})$ to $O(N^{1.4})$ for higher frequencies and for the decomposition of $\mathcal{A}$, respectively. The corresponding memory scalings are $O(N^{1.3})$ and $O(N^{1.0})$.

After LU is done, computing the solution of the LU decomposed linear system becomes much faster and the scalings are between $O(N^{1.2})$ to $O(N^{1.7})$, where $St = 0$ again has the highest slope. This happens due to the denser LU matrices obtained for the poorly-conditioned system. Adding up the cost of these functions, the total cost for RSVD

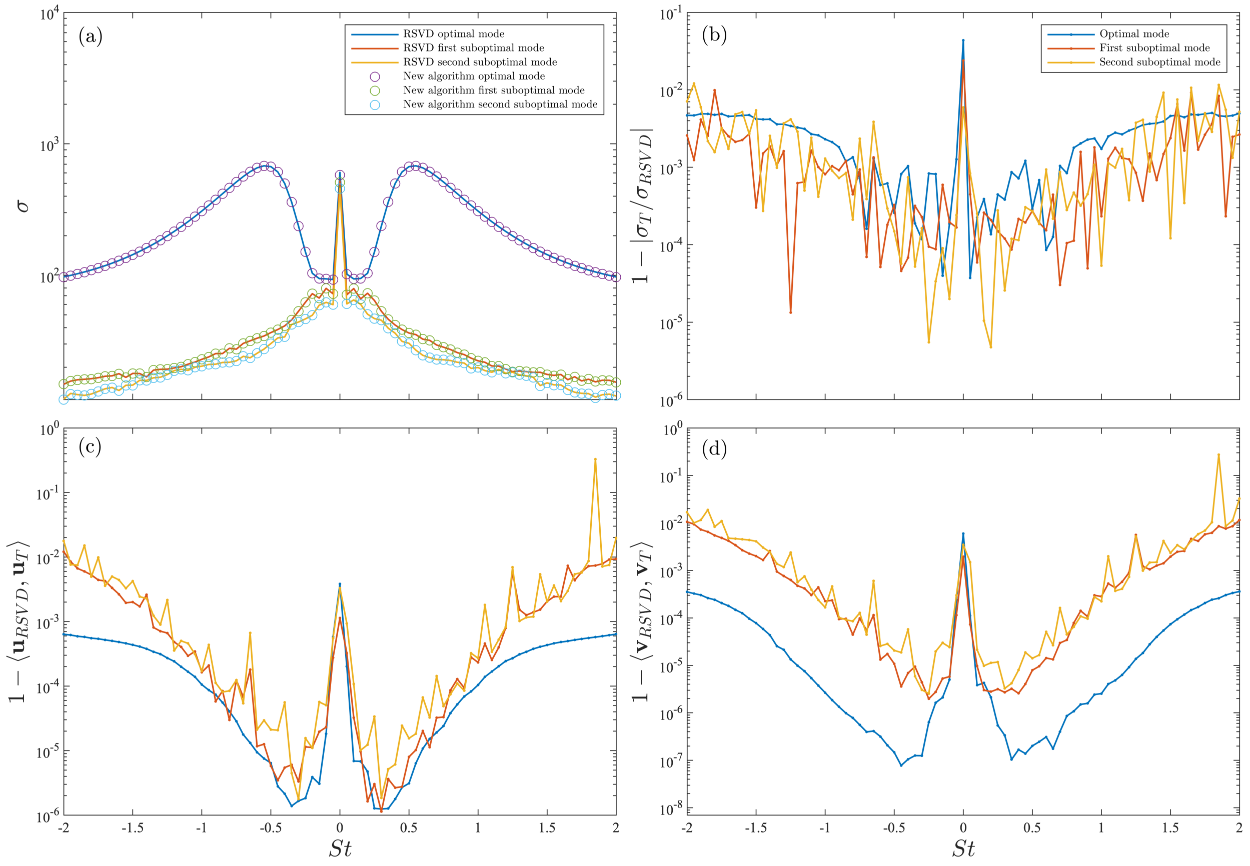

Fig. 5 Axisymmetric jet results: (a) singular values of both algorithms within the desired frequency range; (b) relative error between singular values for the first three leading modes; (c) and (d) cosine similarity between the optimal response and forcing modes, respectively, captured using RSVD and our approach.

when $N_\omega = 82, k = 10$ is less than our approach when a direct solver is used and $N_T = 10,000$. We assumed all the frequencies of interest cost roughly the same as $St = 0.5$, and only added the cost of $St = 0$ once even though other low frequencies are closer to the scalings of $St = 0$. In our approach, $dt = 0.05$ and $N_T \sim O(10^4)$ covers a fairly long time duration. The multilevel time-stepping is also avoided in our comparisons. The total time scales with $O(N^2)$ for RSVD and $O(N)$ for our algorithm over the range of N considered. The overall CPU cost scalings are calculated based on the first four points (due to memory restriction for the RSVD calculation). As N is further increased, the LU and iLU costs will dominate, causing the overall cost of each method to scale with the highest powers observed in 7 (a).

The iterative solver speeds up our approach significantly. The most costly step is finding an effective preconditioner. To this end, we used iLU with a 10^{-3} drop tolerance. To reduce the cost of obtaining the preconditioner, we compute the iLU of a permuted form of $\mathcal{A}$, $\mathbf{P}_1\mathcal{A}\mathbf{P}_2$, rather than $\mathcal{A}$ itself, where $\mathbf{P}_1$ and $\mathbf{P}_2$ can be obtained using one of several available pre-ordering methods that have been developed for this purpose.[56] After finding a preconditioner, the cost of the iterative solver scales like $O(N^{1.2})$, but since the iLU parameters are kept the same, this is not the best scaling that could be obtained. A close connection exists between the preconditioner and the performance of iterative solvers, and one must tune these together for a specific system in order to optimize the cost. The total cost is calculated for the same number of snapshots as the direct solver and scales with $O(N^{1.2})$. In terms of the total CPU cost, our approach scales better with the problem size and for larger systems, outperforms RSVD. The memory quickly becomes the main bottleneck for large systems even if the CPU cost is tractable. Our approach is devised in a way that reduces the peak memory requirement by a few orders of magnitude, especially if an iterative solver is employed.

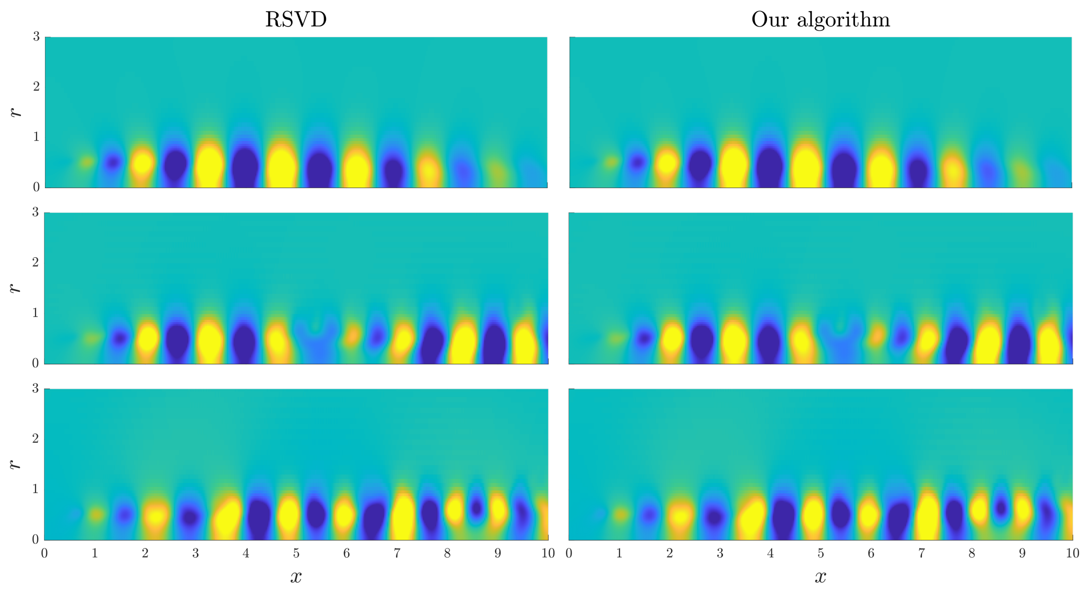

Fig. 6 The first three optimal modes of pressure from top to bottom are displayed in the region of interest for $St = 0.5$. The real part of eigen functions are shown.

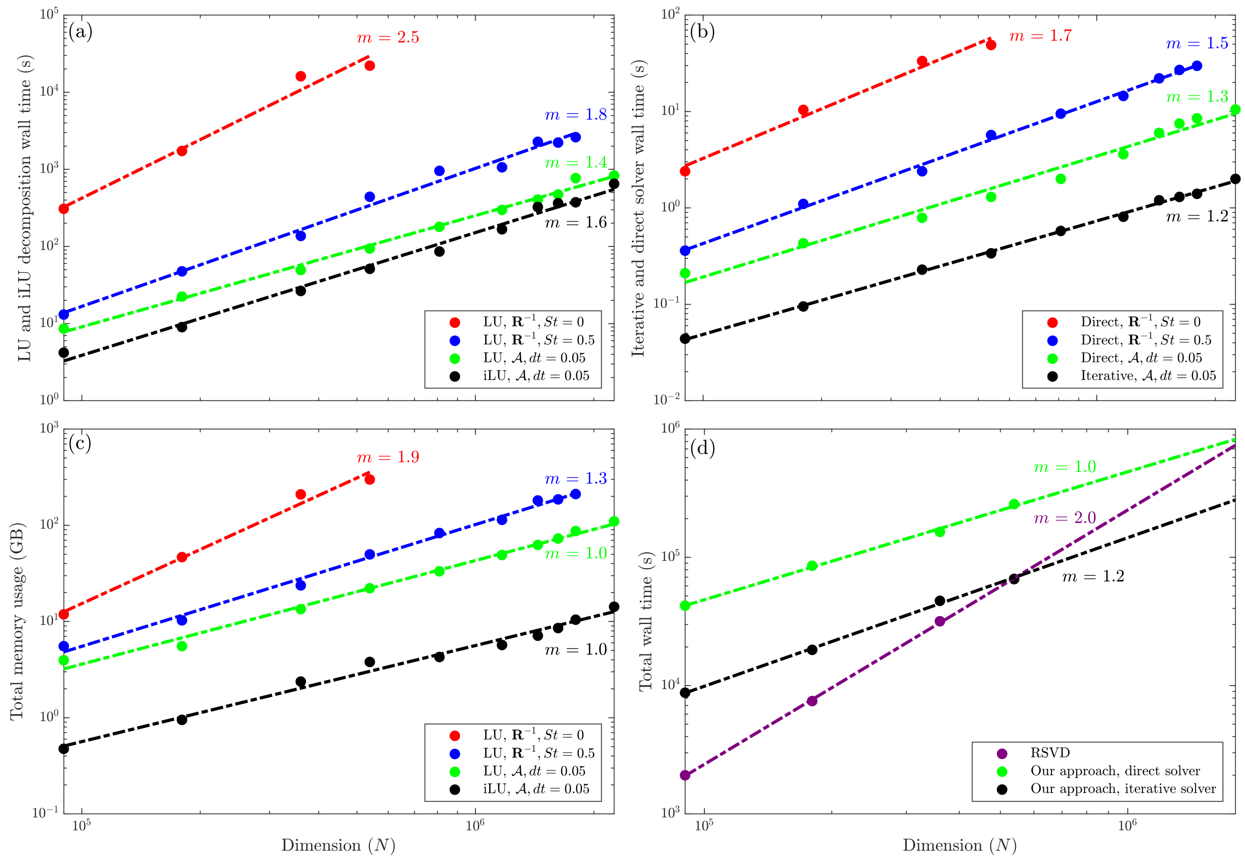

Fig. 7 Cost scaling for the axisymmetric jet problem: (a) wall time for LU or iLU decomposition; (b) wall time for solving the LU decomposed or iLU preconditioned linear system; (c) peak memory usage for the complete algorithm; (d) total wall time for the complete algorithm. All calculations were performed on a single core so that wall time serves as a surrogate for FLOPS.

C. Three-dimensional jet

This section further motivates our study for high-dimensional systems. The 3D jet is obtained by interpolating the axisymmetric LES mean flow data onto a 3D Cartesian grid. The region of interest is $x \in [0, 10]$, $y \times z \in [-4, 4] \times [-4, 4]$ and is surrounded by a sponge region for BCs. The domain is spatially discretized by $200 \times 60 \times 60$ grid points in x, y, and z, respectively. Appropriate weight and spatial filter matrices have been used in the 3D setup. The same set of five variables is of interest, however, the Fourier modes are obtained only in time as

$$\mathbf{q}'(x, y, z, t) = \sum_{\omega} \hat{\mathbf{q}}_{\omega}(x, y, z)e^{i\omega t}. \tag{32}$$

Given the underlying axisymmetry of the mean flow, we still expect that each optimal mode will correspond to a particular azimuthal mode,[57] but this is not explicitly enforced in order to obtain a 3D system. While solving an axisymmetric problem in 3D Cartesian coordinated would not be a sensible strategy in practice, doing so here provides a useful test case for our algorithm.

1. Resolvent modes obtained by our algorithm

As discussed in Section IV.C.2, RSVD requires a prohibitively high amount of memory preventing us from computing resolvent modes of large 3D problems. In contrast, by leveraging a CGS iterative solver with iLU preconditioner in our algorithm, we are able to obtain the $k = 6$ leading modes of the resolvent operator for the 3D turbulent jet. Similar to the axisymmetric jet case, Ω consists of St between -2 and 2 with a resolution of $\Delta St = 0.05$. The stiffness of the problem again motivates the use of the same AM1 integration scheme. As no basis is available for error analysis (RSVD requires more than the 791 GB of RAM available on our workstation), we choose a short transient $T_t = 50$ that chiefly affects the low St range. The objective here is to make sure that our algorithm is capable of capturing physically meaningful resolvent modes in a reasonable amount of time without exceeding the available memory. One power iteration is performed to increase the accuracy. The resolvent mode for $St = 0.5$ is shown in figure 8. The isosurface and 2D slice of pressure show that the optimal mode corresponds to $m = 1$ azimuthal wavenumber.

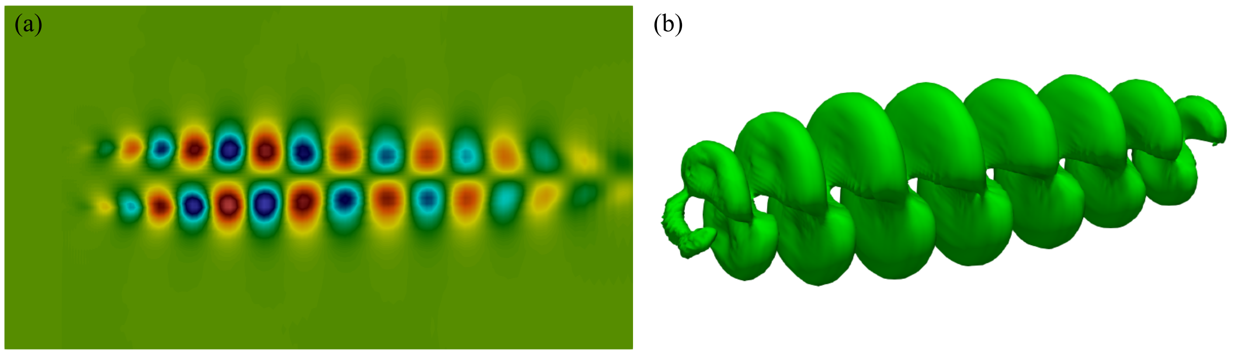

Fig. 8 The real part of pressure for the optimal mode for $St = 0.5$ computed using our algorithm: (a) $y = 0$ plane; (b) isosurface.

2. Computational cost and memory scaling

The last section of this paper looks into the scalings of various functions similar to Section IV.B.2, but this time for the 3D setup. The base state dimension is $N_B = 5 \times N_x \times N_y \times N_z = 5 \times 25 \times 20 \times 20$ for region of interest of $x \in [0, 10]$, $y \times z \in [-3, 3] \times [-3, 3]$. This is seriously under-resolved for this problem, but such small grids are needed to obtain cost scalings for RSVD. Finer grids are then constructed with $N = G \times N_B = 5 \times (\sqrt[3]{G} \times N_x) \times (\sqrt[3]{G} \times N_y) \times (\sqrt[3]{G} \times N_z)$, where G varies between 1 and 8. The rest of the parameters remain the same. The calculations are performed in exactly the same manner as for the axisymmetric jet, so we skip directly to results.

All of the plots in figure 9 show similar trends to those observed in figure 7. The main difference is that all slopes for the 3D problem have been increased with respect to the corresponding slopes for the axisymmetric jet. The LU decomposition CPU time scales like $O(N^{2.9})$ for RSVD when $St = 0$. This is worse scaling than $O(N^{2.4})$ and $O(N^{2.3})$ when our approach is taken with direct and iterative solvers, respectively. Besides better scaling, the iLU CPU cost is lower by a few orders of magnitudes. A similar gap is observed between the computational cost of solving the LU-decomposed system (in either RSVD and our algorithm) versus iteratively solving equation (23) within our algorithm. The scaling for the iterative solver is $O(N^{1.2})$, better than $O(N^{1.8}) - O(N^{2.0})$ for the rest. The overall CPU time is calculated for four data points for which the memory allowed us to compute the LU decomposition. Then the best fit line for those 4 points is extrapolated to cover all dimensions considered (up to $G = 8$) in this section. The total computational costs scale like $O(N^{2.5})$, $O(N^{1.8})$ and $O(N^{1.2})$ for RSVD, our approach with a direct solver, and our approach with an iterative solver, respectively. As seen in figure 9 (d), the total cost is less for the iterative scenario for all values of N.

The limiting factor in this problem is actually the memory consumption. The total memory quickly hits the maximum value available on our workstation when computing the LU decomposition even though the grids under consideration in this section are exceedingly coarse for the considered region of interest. The memory scalings observed in figure 9 (c) are $O(N^{1.7})$ and $O(N^{2.2})$ when decomposing $\mathcal{A}$ and $\mathbf{L}$, respectively. Although our approach is helpful, memory usage still remains the primary challenge for 3D problems using direct solvers. The remedy for memory is to use an iterative solver to avoid computing LU decomposition under any circumstances. In this problem, the iterative solver not only overcomes memory issues, but also provides the fastest way to obtain the resolvent modes of the 3D turbulent jet. Finally, we note that even the finest mesh considered in this analysis is still under-resolved. For a more realistic choice of $N = 5 \times 300 \times 100 \times 100$, the scalings in figure 9 suggest that our time-domain approach with an iterative solver would reduce the CPU cost and memory requirements relative to RSVD by factors of 6×10^3 and 6×10^4, respectively.

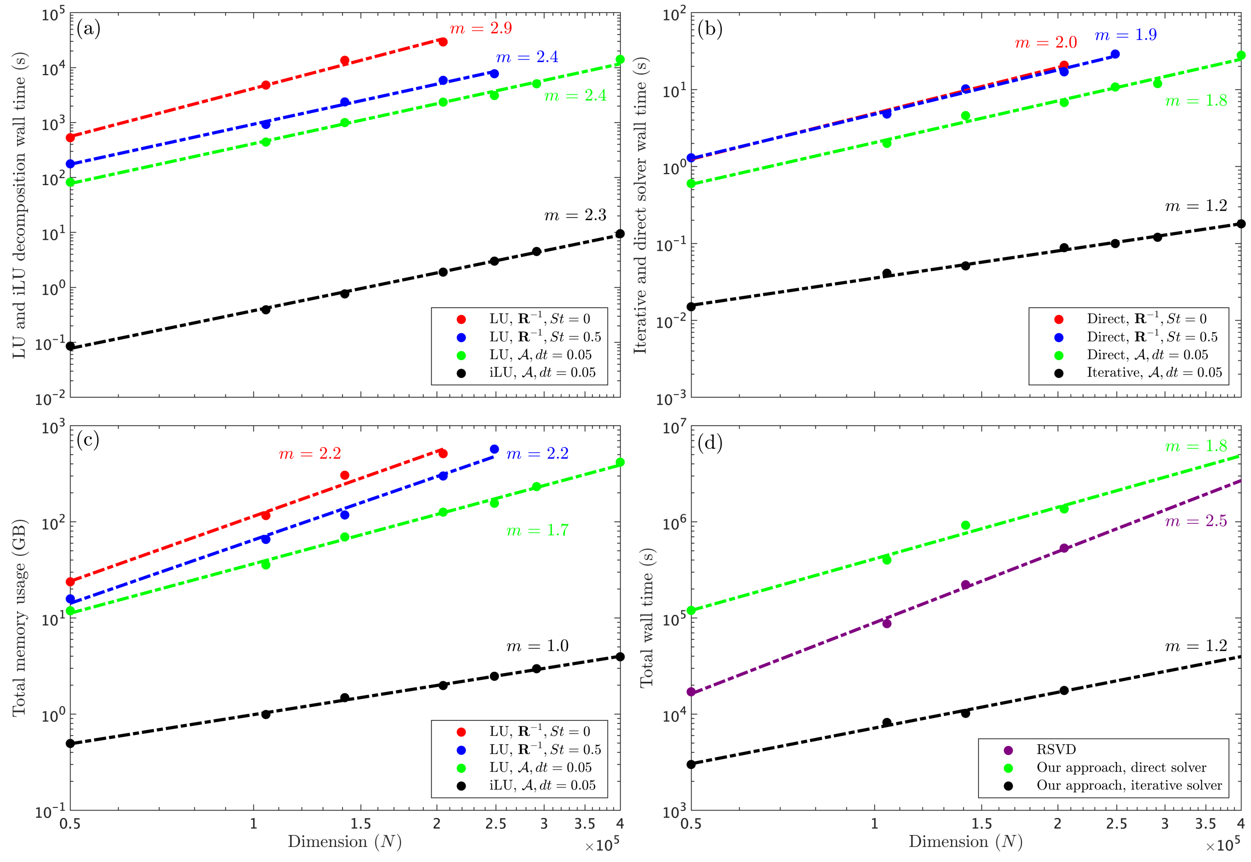

Fig. 9 Cost scaling for the 3D jet problem: (a) wall time for LU or iLU decomposition; (b) wall time for solving the LU decomposed or iLU preconditioned linear system; (c) peak memory usage for the complete algorithm; (d) total wall time for the complete algorithm. All calculations were performed on a single core so that wall time serves as a surrogate for FLOPS.

V. Conclusions

We have presented a new algorithm based on RSVD[20,25] in conjunction with efficient time-stepping methods.[16] Our algorithm is best suited for large problems when straightforward application of the RSVD algorithm becomes intractable. Direct and adjoint time marching schemes enable improved scalability of our proposed algorithm that facilitates resolvent analysis for 3D turbulent flows or other high-dimensional dynamical systems.

The total memory usage, which is arguably the main roadblock to the application of resolvent analysis to large systems in general and 3D problems in particular, is reduced in comparison to RSVD and exhibits better scaling with problem size. An iterative solver can reduce the memory consumption by a few orders of magnitude mainly due to the fact that it never computes an LU decomposition. The total CPU cost of our algorithm also scales better with dimension than RSVD, resulting in computational savings for high-dimensional systems. Applying iterative solvers directly in the frequency domain was found to be substantially less effective due to the inferior conditioning of the associated linear system.

The error of our algorithm compared to RSVD can be driven to machine precision, or can be set to a higher tolerance to further reduce costs. Since the streaming calculations avoid any unnecessary data storage, longer simulations to reach smaller error thresholds do not affect the memory requirement. This flexibility is a consequence of the time-marching approach embedded in our algorithm. The sources of error are transient effects due to initializing the IVP and truncation error due to time-stepping. The CPU cost to obtain a certain accuracy for a range of frequencies is optimized when these two sources of error are of similar magnitude.

An important advantage of the new algorithm is simultaneously computing the leading resolvent modes for a range of frequencies while adding higher harmonics requires a little extra effort due to the nature of time marching. Our algorithm also imposes no restriction on the way one can integrate the linear system of interest. Furthermore, unlike RSVD, which needs access to the explicit inverse of the resolvent operator, our approach captures the resolvent modes for systems where no explicit resolvent operator is defined, *e.g.*, it can be easily incorporated within existing simulation codes with linear direct and adjoint capabilities.

References

[1] Dunford, N., and Schwartz, J. T., *Linear operators part I: general theory*, Vol. 243, 1958.

[2] Kato, T., *Perturbation theory for linear operators*, Vol. 132, Springer Science & Business Media, 2013.

[3] Brandt, L., "The lift-up effect: the linear mechanism behind transition and turbulence in shear flows," *European Journal of Mechanics-B/Fluids*, Vol. 47, 2014, pp. 80–96.

[4] Padovan, A., Otto, S. E., and Rowley, C. W., "Analysis of amplification mechanisms and cross-frequency interactions in nonlinear flows via the harmonic resolvent," *Journal of Fluid Mechanics*, Vol. 900, 2020.

[5] Taira, K., Brunton, S. L., Dawson, S. T. M., Rowley, C. W., Colonius, T., McKeon, B. J., Schmidt, O. T., Gordeyev, S., Theofilis, V., and Ukeiley, L. S., "Modal analysis of fluid flows: An overview," *AIAA Journal*, Vol. 55, No. 12, 2017, pp. 4013–4041.

[6] McKeon, B. J., "The engine behind (wall) turbulence: perspectives on scale interactions," *Journal of Fluid Mechanics*, Vol. 817, 2017.

[7] Trefethen, L. N., Trefethen, A. E., Reddy, S. C., and Driscoll, T. A., "Hydrodynamic stability without eigenvalues," *Science*, Vol. 261, No. 5121, 1993, pp. 578–584.

[8] Jovanovic, M. R., and Bamieh, B., "Componentwise energy amplification in channel flows," *Journal of Fluid Mechanics*, Vol. 534, 2005, p. 145.

[9] McKeon, B. J., and Sharma, A. S., "A critical-layer framework for turbulent pipe flow," *Journal of Fluid Mechanics*, Vol. 658, 2010, p. 336382.

[10] Moarref, R., Sharma, A. S., Tropp, J. A., and McKeon, B. J., "Model-based scaling of the streamwise energy density in high-Reynolds-number turbulent channels," *Journal of Fluid Mechanics*, Vol. 734, 2013, pp. 275–316.

[11] Luhar, M., Sharma, A. S., and McKeon, B. J., "On the design of optimal compliant walls for turbulence control," *Journal of Turbulence*, Vol. 17, No. 8, 2016, pp. 787–806.

[12] Towne, A., Schmidt, O. T., and Colonius, T., "Spectral proper orthogonal decomposition and its relationship to dynamic mode decomposition and resolvent analysis," *Journal of Fluid Mechanics*, Vol. 847, 2018, pp. 821–867.

[13] Symon, S., Illingworth, S. J., and Marusic, I., “Energy transfer in turbulent channel flows and implications for resolvent modelling,” *Journal of Fluid Mechanics*, Vol. 911, 2021.

[14] Chavarin, A., and Luhar, M., “Resolvent analysis for turbulent channel flow with riblets,” *AIAA Journal*, Vol. 58, No. 2, 2020, pp. 589–599.

[15] Towne, A., Lozano-Durán, A., and Yang, X., “Resolvent-based estimation of space–time flow statistics,” *Journal of Fluid Mechanics*, Vol. 883, 2020.

[16] Martini, E., Rodríguez, D., Towne, A., and Cavalieri, A. V. G., “Efficient computation of global resolvent modes,” *Journal of Fluid Mechanics*, Vol. 919, 2021.

[17] Yeh, C., and Taira, K., “Resolvent-analysis-based design of airfoil separation control,” *Journal of Fluid Mechanics*, Vol. 867, 2019, pp. 572–610.

[18] Jung, J., Martini, E., Cavalieri, A., Jordan, P., Lesshafft, L., and Towne, A., “Optimal resolvent-based estimation for flow control,” *Bulletin of the American Physical Society*, 2020.

[19] Monokrousos, A., Åkervik, E., Brandt, L., and Henningson, D. S., “Global three-dimensional optimal disturbances in the Blasius boundary-layer flow using time-steppers,” *Journal of Fluid Mechanics*, Vol. 650, 2010, pp. 181–214.

[20] Ribeiro, J. H. M., Yeh, C., and Taira, K., “Randomized resolvent analysis,” *Physical Review Fluids*, Vol. 5, No. 3, 2020, p. 033902.

[21] Towne, A., and Colonius, T., “One-way spatial integration of hyperbolic equations,” *Journal of Computational Physics*, Vol. 300, 2015, pp. 844–861.

[22] Rigas, G., Schmidt, O. T., Colonius, T., and Bres, G. A., “One way Navier-Stokes and resolvent analysis for modeling coherent structures in a supersonic turbulent jet,” *23rd AIAA/CEAS Aeroacoustics Conference*, 2017, p. 4046.

[23] Herrmann, B., Baddoo, P. J., Semaan, R., Brunton, S. L., and McKeon, B. J., “Data-driven resolvent analysis,” *Journal of Fluid Mechanics*, Vol. 918, 2021.

[24] Towne, A., Colonius, T., Jordan, P., Cavalieri, A. V., and Bres, G. A., “Stochastic and nonlinear forcing of wavepackets in a Mach 0.9 jet,” *21st AIAA/CEAS Aeroacoustics Conference*, 2015, p. 2217.

[25] Halko, N., Martinsson, P., and Tropp, J. A., “Finding structure with randomness: Probabilistic algorithms for constructing approximate matrix decompositions,” *SIAM Review*, Vol. 53, No. 2, 2011, pp. 217–288.

[26] Brynjell-Rahkola, M., Tuckerman, L. S., Schlatter, P., and Henningson, D. S., “Computing optimal forcing using Laplace preconditioning,” *Communications in Computational Physics*, Vol. 22, No. 5, 2017, pp. 1508–1532.

[27] Stewart, G. W., “On the early history of the singular value decomposition,” *SIAM review*, Vol. 35, No. 4, 1993, pp. 551–566.

[28] Tropp, J. A., Yurtsever, A., Udell, M., and Cevher, V., “Practical sketching algorithms for low-rank matrix approximation,” *SIAM Journal on Matrix Analysis and Applications*, Vol. 38, No. 4, 2017, pp. 1454–1485.

[29] Jeun, J., Nichols, J. W., and Jovanović, M. R., “Input-output analysis of high-speed axisymmetric isothermal jet noise,” *Physics of Fluids*, Vol. 28, No. 4, 2016, p. 047101.

[30] Schmidt, O. T., Towne, A., Rigas, G., Colonius, T., and Brès, G. A., “Spectral analysis of jet turbulence,” *Journal of Fluid Mechanics*, Vol. 855, 2018, pp. 953–982.

[31] Rowley, C. W., “Model reduction for fluids, using balanced proper orthogonal decomposition,” *International Journal of Bifurcation and Chaos*, Vol. 15, No. 03, 2005, pp. 997–1013.

[32] Schmidt, O. T., and Towne, A., “An efficient streaming algorithm for spectral proper orthogonal decomposition,” *Computer Physics Communications*, Vol. 237, 2019, pp. 98–109.

[33] Rokhlin, V., Szlam, A., and Tygert, M., “A randomized algorithm for principal component analysis,” *SIAM Journal on Matrix Analysis and Applications*, Vol. 31, No. 3, 2010, pp. 1100–1124.

[34] Gu, M., “Subspace iteration randomization and singular value problems,” *SIAM Journal on Scientific Computing*, Vol. 37, No. 3, 2015, pp. A1139–A1173.

[35] Erichson, N. B., Voronin, S., Brunton, S. L., and Kutz, J. N., "Randomized Matrix Decompositions Using R," *Journal of Statistical Software*, Vol. 89, No. 1, 2019, pp. 1–48.

[36] de Pando, M. F., Sipp, D., and Schmid, P. J., "Efficient evaluation of the direct and adjoint linearized dynamics from compressible flow solvers," *Journal of Computational Physics*, Vol. 231, No. 23, 2012, pp. 7739–7755.

[37] Davis, T. A., and Duff, I. S., "An unsymmetric-pattern multifrontal method for sparse LU factorization," *SIAM Journal on Matrix Analysis and Applications*, Vol. 18, No. 1, 1997, pp. 140–158.

[38] Amestoy, P. R., Buttari, A., l'Excellent, J. Y., and Mary, T. A., "Bridging the gap between flat and hierarchical low-rank matrix formats: The multilevel block low-rank format," *SIAM Journal on Scientific Computing*, Vol. 41, No. 3, 2019, pp. A1414–A1442.

[39] Saad, Y., *Iterative methods for sparse linear systems*, SIAM, 2003.

[40] Greenbaum, A., *Iterative methods for solving linear systems*, 1997.

[41] Saad, Y., "Finding exact and approximate block structures for ILU preconditioning," *SIAM Journal on Scientific Computing*, Vol. 24, No. 4, 2003, pp. 1107–1123.

[42] Yang, X. I. A., and Mittal, R., "Acceleration of the Jacobi iterative method by factors exceeding 100 using scheduled relaxation," *Journal of Computational Physics*, Vol. 274, 2014, pp. 695–708.

[43] Bank, R., Graham, R. L., Stoer, J., and Varga, R., "Springer Series in 8," 1993.

[44] Hairer, E., and Wanner, G., "Solving ordinary differential equations. II, volume 14 of," , 1996.

[45] Stewart, G. W., "Perturbation theory for the singular value decomposition," Tech. rep., 1998.

[46] Cai, T. T., Zhang, A., et al., "Rate-optimal perturbation bounds for singular subspaces with applications to high-dimensional statistics," *The Annals of Statistics*, Vol. 46, No. 1, 2018, pp. 60–89.

[47] Doering, C. R., Gibbon, J. D., Holm, D. D., and Nicolaenko, B., "Low-dimensional behavior in the complex Ginzburg-Landau equation," *Nonlinearity*, Vol. 1, No. 2, 1988, p. 279.

[48] Becker, A., and Kramer, L., "Linear stability analysis for bifurcations in spatially extended systems with fluctuating control parameter," *Physical Review Letters*, Vol. 73, No. 7, 1994, p. 955.

[49] Bagheri, S., Henningson, D. S., Hoepffner, J., and Schmid, P. J., "Input-output analysis and control design applied to a linear model of spatially developing flows," *Applied Mechanics Reviews*, Vol. 62, No. 2, 2009.

[50] Cavalieri, A. V. G., Jordan, P., and Lesshafft, L., "Wave-packet models for jet dynamics and sound radiation," *Applied Mechanics Reviews*, Vol. 71, No. 2, 2019.

[51] Gupta, V., He, W., Wan, M., Chen, S., and Li, L. K. B., "A Ginzburg–Landau model for linear global modes in open shear flows," *Journal of Fluid Mechanics*, Vol. 904, 2020.

[52] Schmidt, O. T., Towne, A., Colonius, T., Cavalieri, A. V. G., Jordan, P., and Brès, G. A., "Wavepackets and trapped acoustic modes in a turbulent jet: coherent structure eduction and global stability," *Journal of Fluid Mechanics*, Vol. 825, 2017, pp. 1153–1181.

[53] Brès, G. A., Ham, F. E., Nichols, J. W., and Lele, S. K., "Unstructured large-eddy simulations of supersonic jets," *AIAA journal*, Vol. 55, No. 4, 2017, pp. 1164–1184.

[54] Brès, G. A., Jordan, P., Jaunet, V., Le Rallic, M., Cavalieri, A. V. G., Towne, A., Lele, S. K., Colonius, T., and Schmidt, O. T., "Importance of the nozzle-exit boundary-layer state in subsonic turbulent jets," *Journal of Fluid Mechanics*, Vol. 851, 2018, pp. 83–124.

[55] Pickering, E., Rigas, G., Schmidt, O. T., Sipp, D., and Colonius, T., "Optimal eddy viscosity for resolvent-based models of coherent structures in turbulent jets," *Journal of Fluid Mechanics*, Vol. 917, 2021.

[56] Amestoy, P. R., Davis, T. A., and Duff, I. S., "An approximate minimum degree ordering algorithm," *SIAM Journal on Matrix Analysis and Applications*, Vol. 17, No. 4, 1996, pp. 886–905.

[57] Sirovich, L., "Turbulence and the dynamics of coherent structures. II. Symmetries and transformations," *Quarterly of Applied mathematics*, Vol. 45, No. 3, 1987, pp. 573–582.