SoftwareX 31 (2025) 102286

Contents lists available at ScienceDirect

SoftwareX

journal homepage: www.elsevier.com/locate/softx

Original software publication

Resolvent4py: A parallel Python package for analysis, model reduction and control of large-scale linear systems

Alberto Padovan [a,*], Vishal Anantharaman [b], Clarence W. Rowley [c], Blaine Vollmer [a], Tim Colonius [b], Daniel J. Bodony [a]

[a] Department of Aerospace Engineering, University of Illinois Urbana-Champaign, 104 S. Wright St., Urbana, IL 61802, United States of America

[b] Department of Mechanical and Civil Engineering, California Institute of Technology, 1200 E. California Boulevard, Pasadena, CA 91125, United States of America

[c] Department of Mechanical and Aerospace Engineering, Princeton University, Olden St., Princeton, NJ 08540, United States of America

ARTICLE INFO

Keywords:
Python
Parallel computing
Model reduction
Resolvent analysis
Stability analysis
Harmonic resolvent analysis

ABSTRACT

In this paper, we present `resolvent4py`, a parallel Python package for the analysis, model reduction and control of large-scale linear systems with millions or billions of degrees of freedom. This package provides the user with a friendly Python-like experience (akin to that of well-established libraries such as `numpy` and `scipy`), while enabling MPI-based parallelism through `mpi4py`, `petsc4py` and `slepc4py`. In turn, this allows for the development of streamlined and efficient Python code that can be used to solve several problems in fluid mechanics, solid mechanics, graph theory, molecular dynamics and several other fields.

Code metadata

Current code version	v1.0.1
Permanent link to code/repository used for this code version	https://github.com/albertopadovan/resolvent4py
Permanent link to Reproducible Capsule	–
Legal Code License	MIT
Code versioning system used	Git (GitHub)
Software code languages, tools, and services used	Python, Petsc4py, Slepc4py, Mpi4py
Compilation requirements, operating environments & dependencies	See README file in package distribution
If available Link to developer documentation/manual	https://albertopadovan.github.io/resolvent4py/
Support email for questions	alberto.padovan.94@gmail.com

1. Motivation and significance

The development of this package is motivated by recent (and not-so-recent) advances in the field of fluid mechanics, where linear analysis has been shown to provide great insights into flow physics. For example, linear systems theory can be used to study the stability of fluid flows around equilibria, develop feedback control strategies to modify the flow behavior and compute reduced-order models to accelerate physical simulations. (We refer the reader to the foundational work of [1] and to the review papers in [2,3] for a more in-depth overview of the importance and impact of linear systems theory in the field of fluid mechanics.) For systems of moderate dimension (e.g., fewer than 10,000 states), all these tasks can be performed straightforwardly with a few lines of Python code thanks to scientific computing libraries like `scipy`. For larger systems, the need for distributed-memory parallelism significantly increases the implementation complexity of these algorithms, and it often requires the development of application-specific software written in compiled languages such as C, C++ and Fortran. In fact, until the recent development of Python libraries like `mpi4py` and `petsc4py`, compiled languages represented the only way to write MPI-based[1] code. The objective of

* Corresponding author.
E-mail address: padovan3@illinois.edu (A. Padovan).

[1] MPI stands for Message Passing Interface, and it refers to the communication protocol used to exchange data across different processors in distributed-memory systems.

https://doi.org/10.1016/j.softx.2025.102286
Received 25 June 2025; Received in revised form 12 July 2025; Accepted 20 July 2025
Available online 9 August 2025

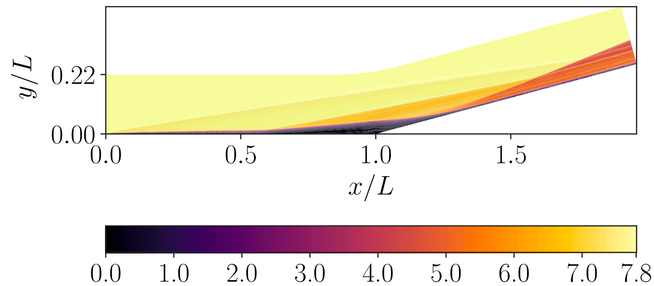

Fig. 1. Mach number contours of two-dimensional steady state solution for the Mach– 7.7 flow over a 15 deg compression ramp.

this package is to address these issues and provide a user-friendly pythonic environment similar to that of `scipy` and `numpy`, while exploiting the large-scale parallelism offered by the `mpi4py`, `petsc4py` and `slepc4py` libraries. In turn, this enables the development of user-friendly, streamlined and efficient code to perform day-to-day engineering tasks and physical analyses of large-scale systems.

Remark 1. Although the development of this package is motivated by problems in the field of fluid mechanics, `resolvent4py` is application agnostic, and its functionalities can be leveraged for any linear system of equations regardless of their origin.

2. Software description

`resolvent4py` is a Python library for the analysis, model reduction and control of large-scale linear systems. It relies on `mpi4py` [4] for distributed-memory parallelism, and it leverages the data structures and functionalities provided by the `petsc4py` [5] and `slepc4py` [6] libraries.

2.1. Software architecture

At the core of this package is an abstract class, called `LinearOperator`, which serves as a blueprint for user-defined child classes that can be used to define a linear operator L. All child classes must implement at least three methods: `apply()`, which defines the action of L on a vector, `apply_mat()`, which defines the action of L on a matrix and `destroy()` to free the memory. The implementation of the second method may seem redundant to the reader who is familiar with native Python, where the `dot()` method can be used to apply the action of a linear operator to vectors and matrices alike. However, the need for this method is dictated by the fact that matrix–vector products and matrix–matrix products are performed with different functions and data structures in `petsc4py`.

If desired, additional methods can be implemented. For example, we often define the action of the hermitian transpose of a linear operator on vectors and matrices (`apply_hermitian_transpose()` and `apply_hermitian_transpose_mat()`, respectively), the action of the inverse L^{-1} on vectors, `solve()`, etc. Below is a list of concrete subclasses of `LinearOperator` that are provided by `resolvent4py`:

1. `MatrixLinearOperator`: the linear operator L is defined by a sparse, MPI-distributed PETSc matrix.
2. `LowRankLinearOperator`: the linear operator L is defined as the product of low-rank factors,

$$L = U\Sigma V^*, \tag{1}$$

where U and V are "tall and skinny" matrices (i.e., matrices with much fewer columns than rows) stored as MPI-distributed SLEPc BV structures,[2] while Σ is a sequential (i.e., non-distributed) two-dimensional `numpy` array of appropriate size.
3. `LowRankUpdatedLinearOperator`: the linear operator L is defined as $L = A + M$, where A is itself a `resolvent4py` linear operator and $M = U\Sigma V^*$ is a low-rank linear operator, as described in the previous bullet point. An operator of this form arises in several applications (e.g., in linear feedback control theory [7]) and, if A and L are full-rank, then the inverse L^{-1} can be computed using the Woodbury matrix inversion lemma [8],

$$L^{-1} = A^{-1} - A^{-1}U\Sigma\left(I + V^*A^{-1}U\Sigma\right)^{-1}V^*A^{-1}. \tag{2}$$

4. `ProjectionLinearOperator`: the linear operator L is defined either as $L = \Phi(\Psi^*\Phi)^{-1}\Psi^*$ or $L = I - \Phi(\Psi^*\Phi)^{-1}\Psi^*$, where Φ and Ψ are tall and skinny matrices of size $N \times r$ stored as SLEPc BVs. In both cases, L defines a projection (i.e., $L^2 = L$).
5. `ProductLinearOperator`: the linear operator L is defined as the product of other `resolvent4py` linear operators L_i of conformal dimensions,

$$L = L_m L_{m-1} \dots L_2 L_1. \tag{3}$$

2.2. Software functionalities

Once a linear operator is properly defined, `resolvent4py` currently allows for several types of analyses.

1. Eigendecomposition: the user can compute the left and right eigendecomposition of the linear operator L using the Arnoldi algorithm [9]. The shift-and-invert technique [10] can be used to compute the eigenvalues that are closest to a target value $s \in \mathbb{C}$, where $\mathbb{C}$ is the set of complex numbers.
2. Singular value decomposition (SVD): the user can compute an SVD of the linear operator using randomized linear algebra [11,12]. This is useful for resolvent analysis [1,13,14] and harmonic resolvent analysis [15–18]. For the specific case of resolvent analysis, the package also offers the possibility of computing the SVD using time-stepping techniques [19,20].
3. Linear time-invariant balanced truncation: given the linear operator L, and appropriately-defined input and output matrices B and C, the user may compute and balance the associated controllability and observability Gramians for model reduction purposes [21–24]. Specifically, we implement the algorithm outlined in [25]. We remark that there exist additional model reduction formulations for linear system, (see, e.g., [26–29] for $\mathcal{H}_2$ and $\mathcal{H}_\infty$ model reduction and variations thereof), but these are not yet implemented in the package.

Additionally, `resolvent4py` ships with several functions — available under the `resolvent4py/utils` directory and directly accessible to the user via the `resolvent4py` namespace — that further facilitate the use of our package and allow for streamlined application-specific code development. These include support for:

1. parallel I/O through `petsc4py`,
2. MPI-based communications using `mpi4py`,
3. manipulation of PETSc matrices/vectors and SLEPc BVs.

All these features are thoroughly documented using `sphinx`, and demonstrated with several examples that ship with the package.

[2] The SLEPc BV structure (where BV stands for "Basis Vector") holds a tall and skinny dense matrix whose rows are distributed across different MPI processors. The columns, however, are not distributed, and this allows for significantly reduced parallel overhead when the number of MPI processors is much larger than the number of columns, as is usually the case in practice.

2.3. Sample code snippet

In this subsection, we show a code snippet similar to the one used in Section 3.2 to perform resolvent analysis on the hypersonic flow over a cone. The first thing performed by this piece of code is to read in the sparse complex-valued matrix A of size $N \times N$. This matrix is stored on disk in sparse COO format by means of PETSc-compatible binary files ``rows.dat'', ``columns.dat'' and ``values.dat''. After assembling A as a sparse PETSc matrix, we create a corresponding `MatrixLinearOperator` object and we perform resolvent analysis using the RSVD-Δt algorithm presented in [19,20]. The desired singular values and vectors of the resolvent operator are then saved to file using the parallel I/O routines available under the `resolvent4py` namespace. Running this piece of code with, e.g., 20 processors can be done with the command `mpiexec -n 20 python code_snippet.py`.

Listing 1: Resolvent analysis via time stepping

```
import resolvent4py as res4py
import numpy as np

# Define problem size and read jacobian from file
N = 121404 # Number of global rows (and columns) of A
Nloc = res4py.compute_local_size(N) # Number of local
    rows
file_names = ["rows.dat", "columns.dat", "values.dat"]
A = res4py.read_coo_matrix(file_names, ((Nloc, N), (
    Nloc, N)))

# Create resolvent4py linear operator
L = res4py.linear_operators.MatrixLinearOperator(A)

# Define the inputs to the rsvd-dt routine
dt = 1e-5 # Time step size
omega = 1.0 # Fundamental frequency (sets period T = 2pi
    /omega)
n_omegas = 5 # Number of harmonics of omega to resolve
n_periods = 100 # Number of periods to integrate through
n_loop = 1 # Number of power iterations
n_rand = 30 # Number of random vectors
n_svals = 3 # Number of singular values/vectors to
    resolve
tol = 1e-3 # Tolerance to declare that transients have
    decayed
verbose = 1 # Verbosity level
Ulst, Slst, Vlst = res4py.linalg.
    resolvent_analysis_rsvd_dt(L, dt, omega, n_omegas
    , n_periods, n_loops, n_rand, n_svals, tol,
    verbose)

# Save to file
np.save("S.npy", Slst[1])
res4py.write_to_file("U.dat", Ulst[1])
res4py.write_to_file("V.dat", Vlst[1])
```

Remark 2. We would like to stress how this snippet exposes the application-agnostic nature of our package. The governing equations (e.g., the linearized fluid dynamics equations, the linearized molecular dynamics equations, or any other linear equation from other branches of physics) are embedded in the sparse matrix A that is passed as an input to the script. Once A is loaded from file, `resolvent4py` performs the desired analysis without additional knowledge of the underlying physics.

3. Illustrative examples

3.1. Hypersonic flow over a compression ramp

The first example is a Mach–7.7 flow over a 15 deg compression ramp of length $L = 0.1$ m. In the freestream, the Mach number is $M_\infty = 7.7$, the pressure is $p_\infty = 760$ Pa, the temperature is $T_\infty = 125$ K, the streamwise velocity is $u_\infty = 1726$ m/s and the Reynolds number based on L is $Re_{\infty,L} = 4.2 \times 10^5$. This flow configuration is studied extensively in [30], and we refer to the latter for details on boundary conditions, spatial discretization, mesh and non-dimensionalization. As discussed in [30], this flow admits a two-dimensional steady state solution (see Fig. 1) that is unstable with respect to three-dimensional perturbations. These instabilities can be detected by linearizing the governing equations about the aforementioned steady state and studying the dynamics of three-dimensional, spanwise-periodic perturbations with wavenumber β. Upon linearization, the spatially-discretized compressible Navier–Stokes equations can be written compactly as

$$\frac{d}{dt} q_\beta = A_\beta q_\beta, \quad q_\beta \in \mathbb{C}^N, \tag{4}$$

where the state vector q_β denotes the wavenumber-β Fourier coefficients of the spatially-discretized flow variables. Here, given a computational grid of size $n_y \times n_x = 236 \times 1076$ and 5 flow variables, the size of the state vector is $N = 5 n_y n_x \approx 1.27 \times 10^6$. The eigenvalues of A_β and corresponding eigenvectors for (non-dimensional) wavenumber $\beta = 2\pi/0.066$ are computed using `resolvent4py` with 240 processors on Stampede3 at Texas Advanced Computing Center. The eigenvalues are shown in Fig. 2, where we see good agreement with the results in Cao et al. [30]. Indeed, we find that the flow exhibits multiple instabilities corresponding to the five eigenvalues λ with real part greater than zero. The flow structures associated with these instability modes are shown in Fig. 3, where we plot contours of the real part of the spanwise velocity component from the eigenvectors corresponding to three unstable eigenvalues. Here, we find that the eigenvector corresponding to λ_1 exhibits very good qualitative agreement with its counterpart in figure 14a in [30]. The eigenvectors corresponding to λ_2 and λ_3 are very similar in nature to those in figures 14b and 14c in [30], but it is important to observe that a direct comparison cannot be made since the modes shown in figures 14b and 14c in [30] correspond to wavenumbers $\beta = 2\pi/0.070$ and $\beta = 2\pi/0.079$, respectively.

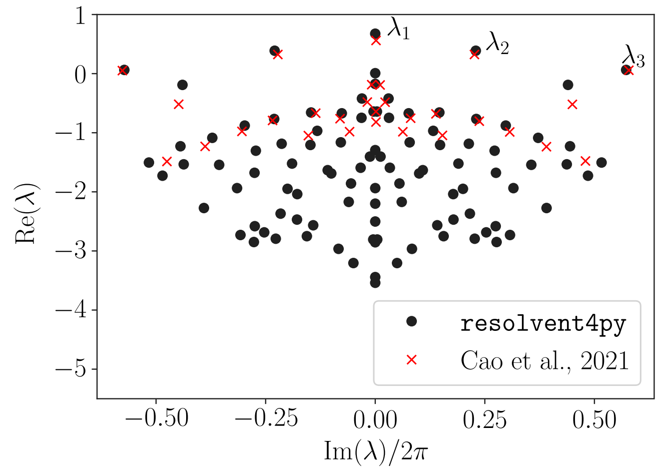

Fig. 2. Eigenvalues of the linearized Navier–Stokes equations at non-dimensional spanwise wavenumber $\beta = 2\pi/0.066$ computed using `resolvent4py` (black dots). The red crosses denote the eigenvalues in [30] for comparison. The eigenvectors associated with λ_1, λ_2 and λ_3 are shown in Fig. 3.

3.2. Hypersonic flow over a cone

Another illustrative example is a Mach–5.9 flow over a 7 deg half-angle blunt cone. The freestream quantities are $M_\infty = 5.9$, $p_\infty = 3396.3$ Pa, $T_\infty = 76.74$ K, $u_\infty = 1036.1$ m/s, and $Re_\infty = 18 \times 10^6$ and the domain length is $L = 0.4893$ m. Bluntness effects produce

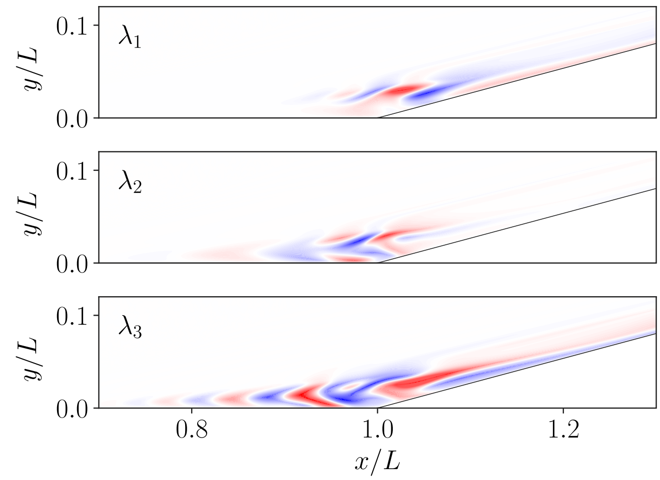

Fig. 3. Real part of the spanwise velocity component from the eigenvectors corresponding to eigenvalues λ_1, λ_2 and λ_3 in Fig. 2.

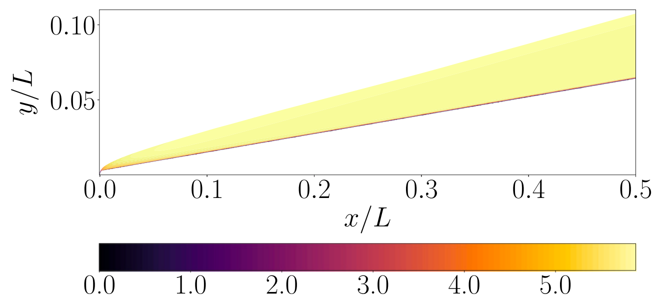

Fig. 4. Mach number contours of two-dimensional steady state solution for the Mach–5.9 flow over a 7 deg blunt cone, truncated to solely the post-shock regime and normalized by domain length.

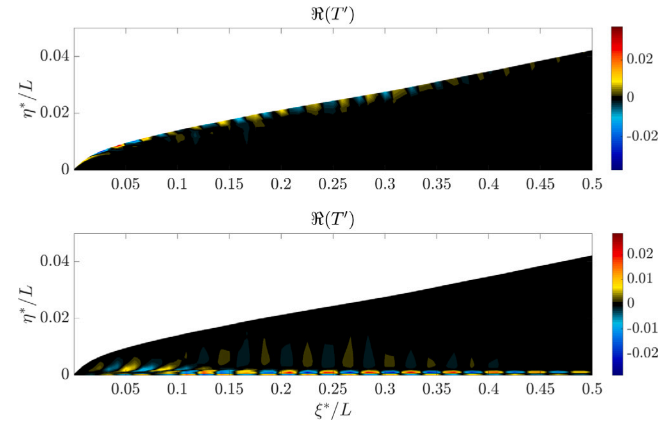

Fig. 5. Temperature forcing and response modes computed using RSVD-Δt plotted in streamwise and wall-normal coordinates normalized by the streamwise length of the domain.

unique instability phenomena in comparison with sharp cones, and are extensively studied using various DNS and operator theoretic methods in receptivity studies [31]. At high Mach numbers, a discrete higher-order mode termed the second Mack mode becomes destabilized and dominates laminar-to-turbulent transition. This mechanism manifests itself as acoustic waves trapped within the boundary layer. Likewise, response modes are modulated by the entropy layer observed near the front of the cone, which can contribute to distinct turbulent transition mechanisms.

The spatially-discretized Navier–Stokes equations are linearized about the base flow shown in Fig. 4, and the singular value decomposition of the resolvent operator is computed using the RSVD-Δt algorithm using a script almost identical to the one shown in Section 2.3. The temperature perturbations extracted from the leading right and left singular vectors of the resolvent are seen in Fig. 5. The response mode corresponding to maximal amplification appears trapped in the boundary layer, while the forcing mode follows the decaying entropy layer's contour within the boundary layer, indicative of the importance of the high entropy gradients of the flow due to bluntness.

4. Impact

In the fluid mechanics community alone, linear systems tools like stability analysis [1,30,32], resolvent analysis [13,14,33], open-loop and feedback control design [34–36], model reduction of linear time-invariant and time-periodic systems via balanced truncation [21,23,25,37], etc., are featured in hundreds of peer-reviewed publications and industry applications. However, despite the popularity of these methods and algorithms, there is no unified platform that helps with their streamlined implementation. `resolvent4py` aims to bridge this gap by offering a Python package that can significantly reduce the upfront cost associated with developing parallel code to answer pressing research questions in physics and engineering. Specifically, not only does `resolvent4py` ship with fully-parallel state-of-the-art algorithms [9,11,12,25] that are used on a daily basis by researchers and engineers in several branches of physics, but it also offers a user-friendly, pythonic environment that allows for rapid code development, testing and deployment. In other words, our package can help scientists and engineers solve very large-scale problems of practical interest at a fraction of the human cost that would otherwise be necessary to develop application-specific software. The simple and self-contained code snippet in Section 2.3 is a testament to the friendly environment enabled by our package.

Additionally, we believe that the current version of `resolvent4py` can serve as a stepping stone for additional software development — both by the original developers and the scientific community at large — that will be featured in future releases. For example, the infrastructure of `resolvent4py` enables the straightforward implementation of time-periodic balanced truncation using "Frequential Gramians" [37], harmonic resolvent analysis via time stepping [38], wavelet-based resolvent analysis [39] and One-Way Navier–Stokes (OWNS) for slowly-developing flows [40]. Furthermore, it should be possible to implement the algorithms discussed in [41,42] for the efficient computation of low-rank solutions to Lyapunov and Riccati equations of large-scale, sparse linear systems such as those that arise from the spatial discretization of partial differential equations. These solutions could then be leveraged for model reduction and closed-loop control design.

5. Conclusions

In this manuscript, we introduced `resolvent4py`, a parallel Python package for the analysis, model reduction and control of large-scale linear systems. Large-scale linear systems with millions or billions of degrees of freedom are ubiquitous in the sciences and engineering, and this package is built to provide a friendly pythonic environment for rapid development and deployment of MPI-based parallel software to solve problems of practical interest. Our library leverages `mpi4py` for distributed-memory parallelism, and it uses the functionalities and data structures provided by `petsc4py` and `slepc4py` to enable stability analysis, input/output analysis, model reduction, control and optimization of linear time-invariant and time-periodic systems. Additionally, the package ships with several functions and features that are available to the user through the `resolvent4py` namespace and that facilitate user-specific and application-specific software development within the larger resolvent4py infrastructure. Finally, although the primary focus of this package is on linear systems, we wish to remark that the data structures and functionalities of resolvent4py can be extended straightforwardly to accommodate the analysis, model reduction and control of large-scale nonlinear systems.

CRediT authorship contribution statement

Alberto Padovan: Writing – original draft, Validation, Software, Conceptualization. **Vishal Anantharaman:** Writing – original draft, Validation, Software. **Clarence W. Rowley:** Writing – review & editing, Software, Funding acquisition. **Blaine Vollmer:** Writing – review & editing, Software. **Tim Colonius:** Writing – review & editing, Funding acquisition. **Daniel J. Bodony:** Writing – review & editing, Funding acquisition.

Declaration of competing interest

The authors declare the following financial interests/personal relationships which may be considered as potential competing interests: Clarence W. Rowley reports financial support was provided by Air Force Office of Scientific Research. Daniel J. Bodony reports financial support was provided by Office of Naval Research. Tim Colonius reports financial support was provided by Office of Naval Research. Tim Colonius reports financial support was provided by The Boeing Company. Daniel J. Bodony reports financial support was provided by National Science Foundation. Daniel J. Bodony reports financial support was provided by Sandia National Laboratories. If there are other authors, they declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgments

This material is based upon work supported by the National Science Foundation, United States under Grant No. 2139536, issued to the University of Illinois at Urbana-Champaign by the Texas Advanced Computing Center under subaward UTAUS-SUB00000545 with Dr. Daniel Stanzione as the PI. DB gratefully acknowledges support from the Office of Naval Research, United States (N00014-21-1-2256), and CR gratefully acknowledges support from the Air Force Office of Scientific Research, United States (FA9550-19-1-0005). TC and VA gratefully acknowledge support from The Boeing Company, United States (CT-BA-GTA-1) and the Office of Naval Research, United States (N00014-25-1-2072). BV was supported by the LDRD Program at Sandia National Laboratories, United States. Sandia is managed and operated by NTESS under DOE NNSA contract DE-NA0003525. The computations in Section 3.1 were performed on TACC's Stampede3 under ACCESS allocation CTS090004.

References

[1] Schmid PJ, Henningson DS. In: Stability and transition in shear flows, vol. 142, Springer-Verlag New York; 2001.

[2] Rowley CW, Dawson STM. Model reduction for flow analysis and control. Annu Rev Fluid Mech 2017;49:387–417.

[3] Jovanović MR. From bypass transition to flow control and data-driven turbulence modeling: An input–output viewpoint. Annu Rev Fluid Mech 2021;53(2021):311–45. http://dx.doi.org/10.1146/annurev-fluid-010719-060244.

[4] Dalcin L, Kler P, Paz R, Cosimo A. Parallel distributed computing using python. Adv Water Resour 2011;34(9):1124–39. http://dx.doi.org/10.1016/j.advwatres.2011.04.013.

[5] Balay S, Abhyankar S, Adams M, Benson S, Brown J, Brune P, et al. PETSc/TAO Users Manual, ANL-21/39 - Revision 3.23. 2025, http://dx.doi.org/10.2172/2476320.

[6] Hernandez V, Roman JE, Vidal V. SLEPc: A scalable and flexible toolkit for the solution of eigenvalue problems. ACM Trans Math Software 2005;31(3):351–62.

[7] Natarajan M, Freund JB, Bodony DJ. Actuator selection and placement for localized feedback flow control. J Fluid Mech 2016;809:775–92. http://dx.doi.org/10.1017/jfm.2016.700.

[8] Woodbury MA. Inverting modified matrices. In: Memorandum rept. 42, statistical research group, Princeton Univ.; 1950, p. vol. 4.

[9] Arnoldi WE. The principle of minimized iterations in the solution of the matrix eigenvalue problem. Quart Appl Math 1951;9:17–29. http://dx.doi.org/10.1090/qam/42792.

[10] Saad Y. Numerical methods for large eigenvalue problems. SIAM; 2011, http://dx.doi.org/10.1137/1.9781611970739.

[11] Halko N, Martinsson PG, Tropp JA. Finding structure with randomness: Probabilistic algorithms for constructing approximate matrix decompositions. SIAM Rev 2011;53:217–88.

[12] Ribeiro JHM, Yeh C-A, Taira K. Randomized resolvent analysis. Phys Rev Fluids 2020;5:033902. http://dx.doi.org/10.1103/PhysRevFluids.5.033902.

[13] Jovanović MR, Bamieh B. Componentwise energy amplification in channel flows. J Fluid Mech 2005;534:145–83.

[14] McKeon BJ, Sharma AS. A critical-layer framework for turbulent pipe flow. J Fluid Mech 2010;658:336–82.

[15] Padovan A, Otto SE, Rowley CW. Analysis of amplification mechanisms and cross-frequency interactions in nonlinear flows via the harmonic resolvent. J Fluid Mech 2020;900. http://dx.doi.org/10.1017/jfm.2020.497.

[16] Padovan A, Rowley CW. Analysis of the dynamics of subharmonic flow structures via the harmonic resolvent: Application to vortex pairing in an axisymmetric jet. Phys Rev Fluids 2022;7:073903. http://dx.doi.org/10.1103/PhysRevFluids.7.073903.

[17] Wu W, Meneveau C, Mittal R, Padovan A, Rowley CW, Cattafesta L. Response of a turbulent separation bubble to zero-net-mass-flux jet perturbations. Phys Rev Fluids 2022;7:084601. http://dx.doi.org/10.1103/PhysRevFluids.7.084601.

[18] Islam MR, Sun Y. Identification of cross-frequency interactions in compressible cavity flow using harmonic resolvent analysis. J Fluid Mech 2024;1000:A13. http://dx.doi.org/10.1017/jfm.2024.949.

[19] Martini E, Rodríguez D, Towne A, Cavalieri AV. Efficient computation of global resolvent modes. J Fluid Mech 2021;919:A3. http://dx.doi.org/10.1017/jfm.2021.364.

[20] Farghadan A, Martini E, Towne A. Scalable resolvent analysis for three-dimensional flows. J Comput Phys 2025;524:113695. http://dx.doi.org/10.1016/j.jcp.2024.113695.

[21] Moore B. Principal component analysis in linear systems: Controllability, observability, and model reduction. IEEE Trans Autom Control 1981;26(1):17–32.

[22] Dullerud GE, Paganini F. A course in robust control theory: a convex approach. Springer New York; 2000.

[23] Rowley CW. Model reduction for fluids using balanced proper orthogonal decomposition. Int J Bifurc Chaos 2005;15(03):997–1013.

[24] Baur U, Benner P, Feng L. Model order reduction for linear and nonlinear systems: a system-theoretic perspective. 2014;21(4):331–58.

[25] Dergham G, Sipp D, Robinet J-C, Barbagallo A. Model reduction for fluids using frequential snapshots. Phys Fluids 2011;23:064101. http://dx.doi.org/10.1063/1.3590732.

[26] Gugercin S, Antoulas AC, Beattie C. $\mathcal{H}_2$ model reduction for large-scale linear dynamical systems. 2008;30(2):609–38.

[27] Flagg G, Beattie CA, Gugercin S. Interpolatory $\mathcal{H}_\infty$ model reduction. Systems Control Lett 2013;62(7).

[28] Benner P, Gugercin S, Willcox K. In: A survey of projection-based model reduction methods for parametric dynamical systems, vol. 57, (4):SIAM; 2015, p. 483–531.

[29] Benner P, Mehrmann V, Sorensen DC. Dimension reduction of large-scale systems. In: Dimension reduction of large-scale systems. Lecture notes in computational science and engineering, vol. 45, Springer; 2005, p. 3–32. http://dx.doi.org/10.1007/3-540-27909-1_1.

[30] Cao S, Hao J, Klioutchnikov I, Olivier H, Wen C-Y. Unsteady effects in a hypersonic compression ramp flow with laminar separation. J Fluid Mech 2021;912:A3. http://dx.doi.org/10.1017/jfm.2020.1093.

[31] Cook DA, Nichols JW. Three-dimensional receptivity of hypersonic sharp and blunt cones to free-stream planar waves using hierarchical input-output analysis. Phys Rev Fluids 2024;9:063901. http://dx.doi.org/10.1103/PhysRevFluids.9.063901.

[32] Vollmer B, Padovan A, Bodony DJ. Surface instabilities in laminar compressible boundary layers with sublimation. 2025, arXiv:2503.11631. URL https://arxiv.org/abs/2503.11631.

[33] Herrmann B, Baddoo PJ, Semaan R, Brunton SL, McKeon BJ. Data-driven resolvent analysis. J Fluid Mech 2021;918:A10. http://dx.doi.org/10.1017/jfm.2021.337.

[34] Semeraro O, Pralits JO, Rowley CW, Henningson DS. Riccati-less approach for optimal control and estimation: an application to two-dimensional boundary layers. J Fluid Mech 2013;731:394–417.

[35] Yeh C-A, Taira K. Resolvent-analysis-based design of airfoil separation control. J Fluid Mech 2019;867:572–610. http://dx.doi.org/10.1017/jfm.2019.163.

[36] Woo J, Murthy SR, Bodony DJ. Resolvent-based framework for jet noise reduction of a low-bypass ratio coannular nozzle. In: AIAA Scitech 2024 forum. 2024, http://dx.doi.org/10.2514/6.2024-2805.

[37] Padovan A, Rowley CW. Continuous-time balanced truncation for time-periodic fluid flows using frequential gramians. J Comput Phys 2024;496:112597. http://dx.doi.org/10.1016/j.jcp.2023.112597.

[38] Farghadan A, Jung J, Bhagwat R, Towne A. Efficient harmonic resolvent analysis via time stepping. Theor Comput Fluid Dyn 2024;38(3):331. http://dx.doi.org/10.1007/s00162-024-00694-1.

[39] Ballouz E, Lopez-Doriga B, Dawson ST, Bae HJ. Wavelet-based resolvent analysis of non-stationary flows. J Fluid Mech 2024;999:A53. http://dx.doi.org/10.1017/jfm.2024.903.

[40] Towne A, Rigas G, Kamal O, Pickering E, Colonius T. Efficient global resolvent analysis via the one-way Navier–Stokes equations. J Fluid Mech 2022;948:A9. http://dx.doi.org/10.1017/jfm.2022.647.

[41] Benner P, Li J-R, Penzl T. Numerical solution of large-scale Lyapunov equations, riccati equations, and linear-quadratic optimal control problems. Numer Linear Algebra Appl 2008;15(9):755–77. http://dx.doi.org/10.1002/nla.622.

[42] Benner P, Saak J. Numerical solution of large and sparse continuous time algebraic matrix riccati and Lyapunov equations: a state of the art survey. GAMM-Mitt 2013;36(1):32–52. http://dx.doi.org/10.1002/gamm.201310003.