Available online at www.sciencedirect.com
ScienceDirect
Comput. Methods Appl. Mech. Engrg. 351 (2019) 718–743
Computer methods in applied mechanics and engineering
www.elsevier.com/locate/cma

Augmented Lagrangian preconditioner for large-scale hydrodynamic stability analysis

Johann Moulin[a], Pierre Jolivet[b,*], Olivier Marquet[a]

[a] ONERA-DAAA, 8 rue des Vertugadins, 92190 Meudon, France
[b] CNRS-IRIT, 2 rue Charles Camichel, 31071 Toulouse Cedex 7, France

Received 15 November 2018; received in revised form 21 March 2019; accepted 25 March 2019
Available online 5 April 2019

Highlights

- Study of the mAL preconditioner for performing hydrodynamic linear stability analyses.
- Comparison of mAL with other state-of-the-art preconditioners such as PCD and SIMPLE.
- Highly scalable method up to 2048 processes using an advanced recycled Krylov method.
- Implementation based on open-source packages such as FreeFEM, PETSc, and SLEPc.

Abstract

Hydrodynamic linear stability analysis of large-scale three-dimensional configurations is usually performed with a "time-stepping" approach, based on the adaptation of existing solvers for the unsteady incompressible Navier–Stokes equations. We propose instead to solve the nonlinear steady equations with the Newton method and to determine the largest growth-rate eigenmodes of the linearized equations using a shift-and-invert spectral transformation and a Krylov–Schur algorithm. The solution of the shifted linearized Navier–Stokes problem, which is the bottleneck of this approach, is computed via an iterative Krylov subspace solver preconditioned by the modified augmented Lagrangian (mAL) preconditioner (Benzi et al., 2011). The well-known efficiency of this preconditioned iterative strategy for solving the real linearized steady-state equations is assessed here for the complex shifted linearized equations. The effect of various numerical and physical parameters is investigated numerically on a two-dimensional flow configuration, confirming the reduced number of iterations over state-of-the-art steady-state and time-stepping-based preconditioners. A parallel implementation of the steady Navier–Stokes and eigenvalue solvers, developed in the FreeFEM language, suitably interfaced with the PETSc/SLEPc libraries, is described and made openly available to tackle three-dimensional flow configurations. Its application on a small-scale three-dimensional problem shows the good performance of this iterative approach over a direct LU factorization strategy, in regards of memory and computational time. On a large-scale three-dimensional problem with 75 million unknowns, a 80% parallel efficiency on 256 up to 2048 processes is obtained.

Keywords: Navier–Stokes equations; Newton method; Krylov–Schur method; Linear stability analysis; Recycled Krylov methods; Distributed computing

* Corresponding author.
E-mail addresses: johann.moulin@onera.fr (J. Moulin), pierre.jolivet@enseeiht.fr (P. Jolivet), olivier.marquet@onera.fr (O. Marquet).

https://doi.org/10.1016/j.cma.2019.03.052

1. Introduction

Over the past century, hydrodynamic linear stability theory was developed to understand the early stage of laminar-turbulence transition in parallel flows, such as boundary layers and shear flows [1]. In the local stability theory, the growth or decay of perturbations developing on parallel flows, described with mono-dimensional velocity profiles, is investigated assuming a normal form decomposition. The resulting eigenproblem is of small size and does not require large computational resources to be solved. Although the local stability theory (mono-dimensional) was then extended to the description of spatially developing flows [2], the linear stability analysis of truly two- and three-dimensional flows has gained in popularity since the beginning of the century [3–9] thanks to the development of computational resources and numerical tools allowing (a) to compute steady solutions of the governing equations and (b) to determine the most unstable eigenmodes of the linearized equations around this steady solution. An efficient and highly-parallel numerical tool is proposed in the present paper to achieve these two steps in the case of the incompressible Navier–Stokes equations.

Two main numerical approaches exist to carry out a linear stability analysis. The first one is the "time-stepping" [9] or "matrix-free" [5] approach based on the use of existing unsteady nonlinear solvers, developed in Computational Fluid Dynamics (CFD). The "matrix-free" denomination indicates that the action of matrices onto vectors is obtained without assembling them. The unsteady solvers are adapted to compute steady solutions and to extract the eigenmodes of largest growth rate, relevant in linear hydrodynamic stability analysis. For computing steady (stable or unstable) solutions, stabilization procedures, such as the recursive projection method [10], the selective frequency damping method [11], or more recently the BoostConv algorithm [12], are applied together with the unsteady nonlinear solver. The computation of leading eigenvalues is then achieved by noticing that the operations performed at each iteration of the *linearized* time-stepping solver correspond to an exponential-based transformation of the Jacobian operator [5,9]. Classical Krylov subspace-based methods like Arnoldi or Krylov–Schur are then commonly used to compute the eigenvalues of the exponential operator with largest magnitude, which are also the leading (rightmost) eigenvalues of the Jacobian operator. One of the advantages of this approach is the computational-time efficiency of applying one time-step of the unsteady solvers. Indeed, these solvers are often highly optimized, not only thanks to very scalable parallel implementation, but also because efficient numerical algorithms have been developed for solving the time-discretized problems (e.g., splitting [13] or fractional step [14] methods). The drawback of this approach is the slow convergence of the Arnoldi method induced by the use of time-steppers. Indeed, small time-steps are required for an application of the linearized time-stepper to approximate accurately the exponential transformation [15]. This leads to a large number of so-called "outer" iterations (in the 10^3–10^4 range) to converge only a few eigenvalues. The efficiency of the "time-stepping" approach is thus mainly based on fast outer iterations at the expense of a large number of such iterations in the Arnoldi process. Note that other strategies for computing matrix exponential allow to relax the small time-step constraint and thus provide better convergence properties [16,17].

The second existing numerical approach to perform linear stability analysis in hydrodynamics [6] is referred here to as the "matrix-based" approach. It relies on the assembly of sparse matrices resulting from the spatial discretization of the underlying problem and the solution of corresponding linear systems using existing parallel libraries that implement direct sparse LU factorization of those matrices (MUMPS [18], SuperLU [19]). The steady-state solutions are then computed by solving the steady nonlinear equations with a (quasi-)Newton method. An invert-based spectral transformation of the Jacobian operator, like the shift-and-invert [6,20–22] or Cayley [23–25] transformations, is then applied with a Krylov subspace-based method (typically, the Arnoldi method [26]) to determine the leading eigenvalues. The Newton method and the shift-and-invert strategy allow, respectively, to achieve fast convergence towards the steady solution and the leading eigenvalues. Usually, the number of Newton iterations is around 10 or so to compute a steady solution, while it may require a few hundred outer iterations in the Arnoldi algorithm to compute a few eigenvalues. This reduced number of applications comes at a price: it requires the ability to invert the linearized steady (and generally shifted) Navier–Stokes equations. Consequently, it has mainly been used for linear stability analysis of two-dimensional flow configurations, for which the number of unknowns remains limited (not much greater than 10^5 unknowns) and the Jacobian matrices remain sparse, so that the LU factorization is affordable. The "matrix-based" strategy is particularly efficient for the eigenvalue computation, since the time-consuming LU factorization of the sparse Jacobian matrix is done once for all, while only the forward eliminations and back substitutions are repeated at each outer iteration. The main drawback of this approach is the large amount of memory needed to perform factorization, especially for three-dimensional flow

configurations [27]. For large-scale hydrodynamics problems, the high cost of forming the Jacobian matrix explicitly, and the prohibitive memory requirements of direct solvers drove many authors [28–30] towards the "matrix-free" strategy.

At the early beginning of nineties, Tuckerman [15,31,32] proposed to improve the slow convergence of the "matrix-free" approach by using a Newton method (resp. a shift-and-invert strategy) for the steady-state (resp. eigenvalue) computation, while still using an existing unsteady solver. This method is based on the observation that one can adapt a (linearized) unsteady solver in order to apply, to some given vector, the steady Navier–Stokes Jacobian operator, left-preconditioned by the (unsteady) Stokes operator. Thus, this technique provides a cheap "matrix-free" way of preconditioning the Navier–Stokes Jacobian operator by the (unsteady) Stokes operator, for use inside Krylov subspace linear solvers typically. The method is nowadays known as the "Stokes" preconditioning technique and has been largely applied during the last decades for the computation of steady-state and leading eigenvalues [33–37]. Recently, it has been adapted and applied to the determination of resolvent modes in large-scale three-dimensional configurations [38]. In the Stokes preconditioning technique, the time-step of the linearized unsteady solver becomes a parameter of the preconditioner. Large time-steps usually provide better preconditioning, but make the application of one linearized time iteration harder. More details and improvements of the method can be found in [39]. In any case, the performance of this method remains limited by the efficiency of the (unsteady) Stokes operator to precondition the linearized steady Navier–Stokes operator.

In the present paper, we propose to develop a "matrix-based" specific solver for performing linear stability analysis, which relies on state-of-the-art preconditioners for the linearized steady incompressible Navier–Stokes equations, thus avoiding the use of direct solvers on the full problem. Over the last decades, various promising approaches have been developed aiming at overcoming the two main difficulties of this problem: the saddle-point structure of the equations deriving from the incompressibility constraint and the absence of a (small) time-step parameter that greatly enhances the convergence of iterative algorithms, due to the resulting diagonal dominance of the matrix. Among those steady preconditioners are the well-known SIMPLE preconditioner [40], the more recent Pressure Convection–Diffusion (PCD) preconditioner proposed by [41], as well as the original augmented Lagrangian (AL) [42–44] and modified augmented Lagrangian (mAL) [43,45–47] preconditioners. Several authors showed the superiority of the modified augmented Lagrangian approach over other state-of-the-art alternatives for solving the Oseen and linearized incompressible Navier–Stokes equations [48,49]. Moreover, a very recent work [50] proposed an efficient and highly scalable steady Navier–Stokes solver based on the original augmented Lagrangian preconditioner. If the augmented Lagrangian strategy has been regularly used for steady-state computations, it was never tested on practical case of eigenvalue computations. A work in that direction was however proposed by Olshanskii and Benzi [51], who adapted the original augmented Lagrangian preconditioner to solve the shifted linearized Navier–Stokes equations. They showed theoretically and numerically that AL was robust to a real-valued shift on a variety of 2D flow configurations. Complex-valued shifts, as needed in practice to efficiently explore the complex plane with a shift-and-invert strategy, were not considered.

The first objective of the present paper is to assess the efficiency of the modified augmented Lagrangian preconditioner for the computation of steady-state solutions with a Newton method and leading eigenvalues with a complex shift-and-invert strategy. The second objective is to describe, and test on a three-dimensional flow configuration, an open-source parallel implementation of the modified augmented Lagrangian preconditioner for linear stability analysis purposes, using the FreeFEM finite element library [52] interfaced with PETSc [53] and SLEPc [54]. The full code is made available at https://github.com/prj-/moulin2019al.

The paper is organized as follows. The governing equations required to carry out the linear stability analysis of incompressible flows are introduced in Section 2. The Newton method used to solve the steady nonlinear equations and the eigensolver based on the shift-and-invert strategy are also described. The preconditioning technique and the modified augmented Lagrangian preconditioner are introduced in Section 3. The parallel implementation is detailed in Section 4. Numerical results are given in Section 5. First we examine, on a two-dimensional problem, the effect of various numerical and physical parameters on the performance of the mAL preconditioner for solving the complex shifted linearized Navier–Stokes problem. Then we compare the performance of mAL with other state-of-the-art preconditioners. Finally, we evaluate the performance of the proposed parallel implementation by first comparing it to a sparse direct solver on a small-scale three-dimensional test case and then, by testing its scalability on a large-scale configuration that cannot be afforded with a direct sparse solver.

2. Methods for linear stability analysis in hydrodynamics

2.1. Governing equations

Let us consider an incompressible flow, described by the two-dimensional (resp. three-dimensional) velocity field $\mathbf{u} = [u, v]^{\mathrm{T}}$ (resp. $\mathbf{u} = [u, v, w]^{\mathrm{T}}$) and the pressure field p, that satisfy the incompressible Navier–Stokes equations:

$$\frac{\partial \mathbf{u}}{\partial t} + (\mathbf{u}\cdot\nabla)\mathbf{u} + \nabla p - \frac{1}{\mathcal{R}e}\nabla^2\mathbf{u} = 0 \quad , \quad -\nabla\cdot\mathbf{u} = 0$$

The Reynolds number is defined as $\mathcal{R}e = U_\infty L/\nu$, where U_∞ and L are characteristic velocity and length used to make non-dimensional the velocity and pressure fields, and ν is the kinematic viscosity. For conciseness, the Navier–Stokes equations are rewritten in a state-space form as follows

$$\mathcal{M}\frac{\partial \mathbf{q}}{\partial t} + \mathcal{R}(\mathbf{q}) = 0, \qquad \mathcal{M} = \begin{pmatrix} 1 & 0 \\ 0 & 0 \end{pmatrix}, \qquad \mathcal{R}(\mathbf{q}) = \begin{pmatrix} (\mathbf{u}\cdot\nabla)\mathbf{u} + \nabla p - \mathcal{R}e^{-1}\nabla^2\mathbf{u} \\ -\nabla\cdot\mathbf{u} \end{pmatrix} \tag{1}$$

where $\mathbf{q} = (\mathbf{u}, p)^{\mathrm{T}}$ is the state-space vector. Base flows, denoted hereinafter $\mathbf{q}_b(\mathbf{x})$, are time-independent (steady) solutions of the Navier–Stokes equations (1) and thus satisfy the nonlinear steady Navier–Stokes equations

$$\mathcal{R}(\mathbf{q}_b) = 0. \tag{2}$$

Linear stability of base flows is investigated by superimposing infinitesimal perturbations $\mathbf{q}'(\mathbf{x}, t)$ to the base flow solution, i.e. $\mathbf{q}(\mathbf{x}, t) = \mathbf{q}_b(\mathbf{x}) + \epsilon\mathbf{q}'(\mathbf{x}, t)$, where ϵ is an infinitesimal parameter. After inserting this decomposition into Eq. (1), using the definition Eq. (2) of the base flow and neglecting high-order terms in ϵ, one obtains the linearized Navier–Stokes equations governing the temporal evolution of the infinitesimal perturbation,

$$\mathcal{M}\frac{\partial \mathbf{q}'}{\partial t} + \mathcal{J}(\mathbf{q}_b)\mathbf{q}' = 0, \qquad \text{where} \qquad \mathcal{J}(\mathbf{q}_b) = \begin{pmatrix} (\mathbf{u}_b\cdot\nabla)(\bullet) + (\bullet\cdot\nabla)\mathbf{u}_b - \mathcal{R}e^{-1}\nabla^2\bullet & \nabla\bullet \\ -\nabla\cdot\bullet & 0 \end{pmatrix}$$

is the Jacobian operator defined around the base flow $\mathbf{q}_b$. The long-term evolution of any infinitesimal perturbation is conveniently described by assuming a spectral decomposition of perturbations as $\mathbf{q}' = \hat{\mathbf{q}}(\mathbf{x})e^{\sigma t} + \text{c.c.}$, where $\hat{\mathbf{q}}(\mathbf{x})$ is a complex spatial field whose temporal evolution is exponential and given by the complex number $\sigma = \lambda + \mathrm{i}\omega$. λ is the growth rate and ω is the angular frequency. Inserting this modal decomposition into the above linearized equations shows that σ and $\hat{\mathbf{q}}$ are respectively eigenvalues and eigenmodes of the generalized eigenproblem:

$$\sigma\,\mathcal{M}\,\hat{\mathbf{q}} + \mathcal{J}(\mathbf{q}_b)\,\hat{\mathbf{q}} = 0. \tag{3}$$

The stability of the base flow is then determined by considering the leading eigenmode $\hat{\mathbf{q}}_0$ associated to the eigenvalue $\sigma_0 = \lambda_0 + \mathrm{i}\omega_0$ with the largest real part λ_0. When the growth rate of the leading eigenmode is negative ($\lambda_0 < 0$), all the eigenvalues have negative real parts, and the base flow is linearly stable since any perturbations superimposed to the base flow is damped at sufficiently large time. On the other hand, when the growth rate of the leading eigenmode is positive ($\lambda_0 > 0$), the perturbation will grow in time and the base flow is linearly unstable [6].

A linear stability analysis thus consists first in computing a base flow, which is a solution of the steady Navier–Stokes Eq. (2), and then in determining the leading eigenvalues/eigenmodes of the eigenproblem (3) with the largest growth rate.

2.2. Spatial discretization

In the present paper, a finite element method is used for the spatial discretization of the nonlinear steady equations (2) and of the linear eigenproblem (3) on a d-dimensional ($d = 2, 3$) domain Ω. A grad–div stabilizated weak formulation [55] of Eq. (2) is used, which consists in finding $\mathbf{u}_b$ in $\mathcal{V}_\Gamma = \left\{\mathbf{u} \in (\mathcal{H}^1(\Omega))^d, \text{ s.t. } \mathbf{u} = \mathbf{u}_\Gamma \text{ on } \Gamma\right\}$ and p_b in $\mathcal{Q} = \mathcal{L}^2(\Omega)$ such that:

$$\mathcal{R}_{\mathbf{u}}(\mathbf{q}_b; \check{\mathbf{u}}) = \langle \mathbf{u}_b\cdot\nabla\mathbf{u}_b, \check{\mathbf{u}}\rangle + \langle \mathcal{R}e^{-1}\nabla\mathbf{u}_b, \nabla\check{\mathbf{u}}\rangle - \langle p_b, \nabla\cdot\check{\mathbf{u}}\rangle + \gamma\langle\nabla\cdot\mathbf{u}_b, \nabla\cdot\check{\mathbf{u}}\rangle = 0 \tag{4a}$$

$$\mathcal{R}_p(\mathbf{q}_b; \check{p}) = -\langle\nabla\cdot\mathbf{u}_b, \check{p}\rangle = 0 \tag{4b}$$

for all $(\check{\mathbf{u}}, \check{p})$ in $\mathcal{V}_0 \times \mathcal{Q}$, where $\langle\bullet, \bullet\rangle$ denotes the $\mathcal{L}^2$ inner-product and $\mathcal{V}_0 = \{\mathbf{u} \in (\mathcal{H}^1(\Omega))^d, \text{ s.t. } \mathbf{u} = 0 \text{ on } \Gamma\}$ is the velocity space with vanishing velocity on the boundary Γ. The weak residuals of the momentum and mass

conservation equations are $\mathcal{R}_{\mathbf{u}}$ and $\mathcal{R}_p$, respectively. The last term in the momentum residual $\mathcal{R}_{\mathbf{u}}$ is the grad–div stabilization (also called augmentation) term that corresponds to the weak form of $-\gamma\nabla(\nabla\cdot\mathbf{u}_b)$, with $\gamma \geq 0$ a numerical parameter. In the above continuous weak formulation, the stabilization term strictly vanishes on the solution: $\langle\nabla\cdot\mathbf{u}_b, \nabla\cdot\check{\mathbf{u}}\rangle = 0$. Indeed, the divergence-free condition $\langle\nabla\cdot\mathbf{u}_b, \check{p}\rangle = 0$ is satisfied for all $\check{p}\in\mathcal{Q}$, and in particular for $\nabla\cdot\check{\mathbf{u}}\in\mathcal{Q}$.

A Delaunay triangulation of the domain $\mathcal{T}_h = \{K\}$, consisting in triangular ($d = 2$) or tetrahedral ($d = 3$) elements K, is used. In order to satisfy the inf–sup Ladyženskaja–Babuška–Brezzi (LBB) condition (see [56]), the Taylor–Hood finite element pair is chosen, so that the discrete velocity $\mathbf{u}_b^h$ and pressure p_b^h are sought respectively in $\mathcal{V}_\Gamma^h = \{\mathbf{u}^h \in \mathcal{C}^0(\Omega),\ \text{s.t.}\ \mathbf{u}^h\big|_K \in \mathbb{P}_2(K),\ \forall K\in\mathcal{T}_h\ ,\ \mathbf{u}^h = \mathbf{u}_\Gamma \text{ on } \Gamma\}$ and $\mathcal{Q}^h = \{p^h \in \mathcal{C}^0(\Omega),\ \text{s.t.}\ p^h\big|_K \in \mathbb{P}_1(K),\ \forall K\in\mathcal{T}_h\}$. Note that, with the Taylor–Hood finite element pair, the discrete divergence of the velocity test functions does not belong to the discrete pressure space, i.e. $\nabla\cdot\check{\mathbf{u}}^h \notin \mathcal{Q}^h$. Therefore, contrary to the continuous case, the stabilization term does not vanish from the discrete momentum equation ($\langle\nabla\cdot\mathbf{u}_b^h, \nabla\cdot\check{\mathbf{u}}^h\rangle \neq 0$), and the discrete solution depends on the value of the stabilization parameter γ. Here, the grad–div stabilization is mainly introduced to improve, thanks to an efficient preconditioner (see Section 3), the iterative solution of linear systems involved when solving the nonlinear discrete equations (5). The question of whether the grad–div stabilized discrete solution is closer or further from the continuous weak solution is out of the scope of this paper. However, several studies (e.g. [44,55,57–59]) showed that the grad–div stabilization often improves the mass conservation property and the velocity error of the discrete solution, for adequate values of γ. Numerical experiments are performed in Section 5.2.1 to assess the accuracy of the stabilized discrete solution and to determine adequate values of γ.

Such a discretization yields the following discrete version of the nonlinear base flow Eq. (2):

$$\mathbf{R}(\mathbf{q}_b^h) = \mathbf{0} \tag{5}$$

where $\mathbf{q}_b^h$ denotes now the vector of coefficients of $\mathbf{u}_b^h$ and p_b^h in the finite elements basis.

The generalized eigenproblem (3) is discretized similarly, yielding

$$\sigma\,\mathbf{M}\,\hat{\mathbf{q}}^h + \mathbf{J}(\mathbf{q}_b{}^h)\,\hat{\mathbf{q}}^h = 0 \tag{6}$$

where $\mathbf{M}$ and $\mathbf{J}(\mathbf{q}_b{}^h)$, the finite element matrices obtained after discretization of the mass $\mathcal{M}$ and Jacobian operator $\mathcal{J}(\mathbf{q}_b{}^h)$, are respectively defined as

$$\mathbf{M} = \begin{pmatrix} \mathbf{M}_{\mathbf{u}} & 0 \\ 0 & 0 \end{pmatrix} \qquad \text{and} \qquad \mathbf{J}(\mathbf{q}_b{}^h) = \begin{pmatrix} \mathbf{A}_\gamma & \mathbf{B}^{\mathrm{T}} \\ \mathbf{B} & 0 \end{pmatrix}. \tag{7}$$

The rectangular matrix $\mathbf{B}$ is the discretization of the divergence operator and its transpose $\mathbf{B}^{\mathrm{T}}$ represents the discrete gradient. The mass matrix on the velocity space $\mathbf{M}_{\mathbf{u}}$ can be written as a 3-by-3 block diagonal matrix corresponding to the three velocity components. The 3-by-3 block matrix $\mathbf{A}_\gamma = \mathbf{A} + \gamma\boldsymbol{\Gamma}$ is the sum of $\mathbf{A}$, which represents the linearized diffusion and convection terms in the momentum conservation equation, and $\boldsymbol{\Gamma}$, obtained after discretization of the grad–div stabilization term. They write:

$$\mathbf{A}_\gamma = \begin{pmatrix} \mathbf{A}_{uu}+\gamma\boldsymbol{\Gamma}_{uu} & \mathbf{A}_{uv}+\gamma\boldsymbol{\Gamma}_{uv} & \mathbf{A}_{uw}+\gamma\boldsymbol{\Gamma}_{uw} \\ \mathbf{A}_{vu}+\gamma\boldsymbol{\Gamma}_{vu} & \mathbf{A}_{vv}+\gamma\boldsymbol{\Gamma}_{vv} & \mathbf{A}_{vw}+\gamma\boldsymbol{\Gamma}_{vw} \\ \mathbf{A}_{wu}+\gamma\boldsymbol{\Gamma}_{wu} & \mathbf{A}_{wv}+\gamma\boldsymbol{\Gamma}_{wv} & \mathbf{A}_{ww}+\gamma\boldsymbol{\Gamma}_{ww} \end{pmatrix} \quad , \quad \mathbf{M}_{\mathbf{u}} = \begin{pmatrix} \mathbf{M}_u & 0 & 0 \\ 0 & \mathbf{M}_v & 0 \\ 0 & 0 & \mathbf{M}_w \end{pmatrix}. \tag{8}$$

In the following, we will mostly refer to the discrete solutions. Therefore, the superscript ^{h} is dropped unless confusion is possible.

2.3. Nonlinear steady-state solver

The nonlinear solution $\mathbf{q}_b$ of the discrete problem Eq. (5) is obtained by the classical Newton method. The approximated solution at the kth iteration is obtained as

$$\mathbf{q}_b^k = \mathbf{q}_b^{k-1} + \boldsymbol{\delta}\mathbf{q}_b^k, \tag{9}$$

where $\boldsymbol{\delta}\mathbf{q}_b{}^k$ denotes the solution increment, obtained by solving the linear problem

$$\mathbf{J}(\mathbf{q}_b^{k-1})\,\boldsymbol{\delta}\mathbf{q}_b^k = -\mathbf{R}(\mathbf{q}_b^{k-1}) \tag{10}$$

where $\mathbf{J}(\mathbf{q}_b^{k-1})$ is the Jacobian matrix defined in Eq. (7) with the known approximation of the steady solution $\mathbf{q}_b^{k-1}$. The solution of this linear system is repeated for each iteration of the Newton algorithm, that is considered to be converged when the l^2 norm of the residual $\|\mathbf{R}(\mathbf{q}_b)\|_2$ is below some numerical tolerance.

2.4. Linear eigensolver

The Krylov–Schur algorithm [60] is used in the present study to solve the generalized eigenproblem (6). In order to compute the leading eigenvalues, which lie in the complex plane close to the zero growth-rate axis ($\lambda = 0$) for any frequency ω, a shift-and-invert spectral transformation is first applied, yielding the transformed eigenproblem

$$\mu\,\hat{\mathbf{q}} + \mathbf{T}\hat{\mathbf{q}} = 0, \qquad \mathbf{T} = (\mathbf{J}(\mathbf{q}_b) + s\mathbf{M})^{-1}\,\mathbf{M} \tag{11}$$

where $s = s_r + \mathrm{i}s_\mathrm{i}$ is the so-called complex shift. The eigenvalues μ of the transformed problem are related to the eigenvalues σ of Eq. (6) through $\mu = (\sigma - s)^{-1}$ while the eigenvectors are left unchanged by the spectral transformation. Like the classical power method, the Krylov–Schur algorithm allows to compute the eigenvalues of largest magnitude. When applied to the transformed problem, it gives the eigenvalues μ of largest magnitude, which correspond to the eigenvalues σ closest to the complex shift s. To determine the leading eigenvalue of Eq. (6), the eigenproblem (11) is solved for several values of the complex s close to the real axis, spanning appropriately the imaginary axis. For each eigenvalue computation, the Krylov–Schur algorithm requires multiple "matrix–vector" applications of the matrix $\mathbf{T}$. In other words, repeated solutions of the linear system $(\mathbf{J}(\mathbf{q}_b) + s\mathbf{M})\,\mathbf{q}_o = \mathbf{q}_i$ are required, where the right-hand side vectors $\mathbf{q}_i$ are given by the Krylov–Schur algorithm.

In the present work, linear stability analysis is thus performed using a nonlinear steady-state solver and a linear eigensolver, that both rely on multiple solutions of linear systems involving the complex shifted Jacobian matrix $(\mathbf{J} + s\,\mathbf{M})$. For the steady-state solver, this matrix reduces to the real Jacobian matrix $\mathbf{J}$ as the complex shift vanishes $s = 0$. The next section introduces a preconditioned iterative method used to solve efficiently such systems.

3. An augmented Lagrangian approach for the shifted Jacobian matrix

As explained in the previous section, the main challenge of an hydrodynamic stability analysis is to solve efficiently the following linear equation:

$$\begin{pmatrix} \mathbf{A}_{\gamma,uu} + s\mathbf{M}_u & \mathbf{A}_{\gamma,uv} & \mathbf{A}_{\gamma,uw} & \mathbf{B}_u^{\mathrm{T}} \\ \mathbf{A}_{\gamma,vu} & \mathbf{A}_{\gamma,vv} + s\mathbf{M}_v & \mathbf{A}_{\gamma,vw} & \mathbf{B}_v^{\mathrm{T}} \\ \mathbf{A}_{\gamma,wu} & \mathbf{A}_{\gamma,wv} & \mathbf{A}_{\gamma,ww} + s\mathbf{M}_w & \mathbf{B}_w^{\mathrm{T}} \\ \mathbf{B}_u & \mathbf{B}_v & \mathbf{B}_w & 0 \end{pmatrix} \begin{pmatrix} u_o \\ v_o \\ w_o \\ p_o \end{pmatrix} = \begin{pmatrix} u_i \\ v_i \\ w_i \\ p_i \end{pmatrix} \tag{12}$$

where $\mathbf{A}_{\gamma,\alpha\beta} = \mathbf{A}_{\alpha\beta} + \gamma\,\Gamma_{\alpha\beta}$ $(\alpha, \beta = u, v, w)$, $\mathbf{q}_o = (u_o, v_o, w_o, p_o)^{\mathrm{T}}$ is the solution vector and $\mathbf{q}_i = (u_i, v_i, w_i, p_i)^{\mathrm{T}}$ is a right-hand side vector. In the perspective of large-scale computations, we must avoid the use of direct solvers applied directly to Eq. (12), due to their huge memory cost [27]. Instead, we use the flexible Generalized Minimal Residual algorithm (GMRES) [61] for solving iteratively Eq. (12). The shifted-Jacobian matrix being indefinite and ill-conditioned, the use of an iterative method without preconditioning is inefficient as it requires a very large number of iterations [37]. To improve the numerical efficiency of the iterative solution, the above linear system is replaced by the right-preconditioned linear system:

$$\begin{pmatrix} \mathbf{A}_{\gamma,uu} + s\mathbf{M}_u & \mathbf{A}_{\gamma,uv} & \mathbf{A}_{\gamma,uw} & \mathbf{B}_u^{\mathrm{T}} \\ \mathbf{A}_{\gamma,vu} & \mathbf{A}_{\gamma,vv} + s\mathbf{M}_v & \mathbf{A}_{\gamma,vw} & \mathbf{B}_v^{\mathrm{T}} \\ \mathbf{A}_{\gamma,wu} & \mathbf{A}_{\gamma,wv} & \mathbf{A}_{\gamma,ww} + s\mathbf{M}_w & \mathbf{B}_w^{\mathrm{T}} \\ \mathbf{B}_u & \mathbf{B}_v & \mathbf{B}_w & 0 \end{pmatrix} \mathbf{P}^{-1} \begin{pmatrix} \tilde{u}_o \\ \tilde{v}_o \\ \tilde{w}_o \\ \tilde{p}_o \end{pmatrix} = \begin{pmatrix} u_i \\ v_i \\ w_i \\ p_i \end{pmatrix} \tag{13}$$

where the matrix $\mathbf{P}$ is the so-called preconditioner and $(\tilde{u}_o, \tilde{v}_o, \tilde{w}_o, \tilde{p}_o)^T$ is an intermediate solution. The final solution is found by solving the following linear system

$$\mathbf{P} \begin{pmatrix} u_o \\ v_o \\ w_o \\ p_o \end{pmatrix} = \begin{pmatrix} \tilde{u}_o \\ \tilde{v}_o \\ \tilde{w}_o \\ \tilde{p}_o \end{pmatrix} \tag{14}$$

The GMRES algorithm is applied to the right-preconditioned Eq. (13) which, in addition to matrix–vector products with the shifted-Jacobian matrix, requires the repeated application of $\mathbf{P}^{-1}$, i.e., the solution of Eq. (14). A good preconditioner achieves a compromise between a fast application of the preconditioner and a small number of iterations to solve the preconditioned system.

The augmented Lagrangian preconditioner allows to solve iteratively the Oseen [42–45] and linearized Navier–Stokes equations [46,51] in a very limited number of iterations, regardless of the mesh refinement and the Reynolds number value. Nevertheless, these interesting properties are counterbalanced by the difficulty of solving iteratively the coupled (two or three-dimensional) velocity subproblem arising in the application of the original preconditioner, as it requires highly specific multigrid solvers [42,50]. In order to circumvent this particular issue, the so-called modified augmented Lagrangian (mAL) preconditioner was introduced in [45]. It is derived from the original augmented Lagrangian preconditioner by neglecting either the lower block matrices [45] or the upper block matrices [49], as follows

$$\mathbf{P}_{\mathrm{mAL}} = \begin{pmatrix} \mathbf{A}_{\gamma,uu} + s\mathbf{M}_u & 0 & 0 & 0 \\ \mathbf{A}_{\gamma,vu} & \mathbf{A}_{\gamma,vv} + s\mathbf{M}_v & 0 & 0 \\ \mathbf{A}_{\gamma,wu} & \mathbf{A}_{\gamma,wv} & \mathbf{A}_{\gamma,ww} + s\mathbf{M}_w & 0 \\ \mathbf{B}_u & \mathbf{B}_v & \mathbf{B}_w & \mathbf{S}_p \end{pmatrix}, \tag{15}$$

where $\mathbf{S}_p$ is an approximation of the pressure Schur complement $-\mathbf{B}\left(\mathbf{A}_\gamma + s\mathbf{M}_\mathbf{u}\right)^{-1}\mathbf{B}^{\mathrm{T}}$. Rather than being explicitly specified, this matrix is defined by the action of its inverse as

$$\mathbf{S}_p{}^{-1} = -(\gamma + \mathcal{R}e^{-1})\mathbf{M}_p{}^{-1} - s\mathbf{L}_p{}^{-1}, \tag{16}$$

where $\mathbf{M}_p$ is the mass matrix and $\mathbf{L}_p$ the Laplacian matrix, both defined on the discrete pressure space. Note that, for base flow computations $s = 0$, only the first term remains in the definition of the approximated Schur complement Eq. (16). The lower block-triangular version of the preconditioner is chosen for practical reasons explained in Section 4. In the original preconditioner proposed by [42,45], the augmentation term of the Jacobian matrix $\boldsymbol{\Gamma}$ was defined algebraically as $\mathbf{B}^T\mathbf{M}_p{}^{-1}\mathbf{B}$, thus requiring two sparse matrix products to be constructed explicitly. As proposed in [44], the construction cost can significantly be reduced by building the matrix $\boldsymbol{\Gamma}$ from the finite element discretization of the grad–div stabilization term. They showed that $\boldsymbol{\Gamma}$ is then the sum of the algebraic augmentation $\mathbf{B}^T\mathbf{M}_p{}^{-1}\mathbf{B}$ and of a stabilization matrix. The efficiency of the mAL preconditioner is thus conserved while significantly reducing the construction costs. Finally, the grad–div augmentation matrix $\boldsymbol{\Gamma}$ is much more sparse than its algebraic counterpart, thus motivating our choice for this implementation of the mAL preconditioner, in the perspective of large-scale three-dimensional computations.

4. Parallel implementation with FreeFEM and its interface to PETSc/SLEPc

For realistic three-dimensional geometries, the approach derived in the previous sections requires the solution of nonlinear systems and generalized eigenproblems of large dimensions. Thus, high-performance computing becomes necessary. The goal of this section is to show how this is done using a finite element domain specific language, FreeFEM [52,62], interfaced with distributed linear algebra backends, PETSc [53] and SLEPc [54]. A thorough introduction of these libraries may be found in their respective manuals.[1,2,3] Our implementation is openly available at https://github.com/prj-/moulin2019al and the rest of this section follows the available source code.

4.1. Outer solvers

In this section, we describe how the outer solvers (i.e. the nonlinear steady-state solver and the eigensolver) are implemented.

The Newton method described in Section 2.3 is implemented using FreeFEM. Only the inversion of the Jacobian matrix $\mathbf{J}$ of Eq. (10) is performed by PETSc. Given a FreeFEM distributed version of the Jacobian matrix `dJ`, PETSc options defining the linear solver for Eq. (10) are set using the following FreeFEM syntax:

```
set(dJ, sparams = params, fields = vX[], names = names,
  schurPreconditioner = S, schurList = listX[]);
```

[1] https://doc.freefem.org/pdf/FreeFEM-documentation.pdf.
[2] http://www.mcs.anl.gov/petsc/petsc-current/docs/manual.pdf.
[3] http://slepc.upv.es/documentation/slepc.pdf.

The keyword `sparams` is a string defined by the user gathering the PETSc runtime options for the Krylov subspace solver (KSP) and preconditioner (PC). The interested reader should refer to PETSc manual for details on the use of runtime options. The keywords `fields`, `names`, `schurPreconditioner`, and `schurList` allow to implement specific block preconditioners, like mAL, and their use is detailed in the next sections.

For the eigenvalue computation presented in Section 2.4, only the finite element matrices are built by FreeFEM. Then, the Krylov–Schur algorithm is performed entirely by SLEPc through the use of the eigenvalue problem solver framework (EPS), which is called from within FreeFEM.

```
int k = EPSSolve(dJ, dM, vectors = vec, values = val, sparams = params,
  fields = vX[], names = names, schurPreconditioner = S, schurList = listX[]);
// solves the eigenvalue problem dJq̂ = σdMq̂
```

Contrary to the case of the linear solver interface, two matrices `dJ` and `dM` that define the generalized eigenproblem (6) must now be passed to SLEPc. In addition, `sparams` must also contain the SLEPc runtime options defining the eigensolver.

4.2. Inner mAL-preconditioned linear solvers

The inner linear solves of system Eq. (12) with a mAL-preconditioned GMRES require the implementation of the block structure of the preconditioner (Eq. (15)). This is done in PETSc by using the so-called fieldsplit structure that gives to the users a high-level of abstraction to define operators by blocks. The following PETSc runtime options define such a preconditioner:

```
string params = paramsXYZ + " " + paramsP + " " + paramsKrylov +
  " -pc_type fieldsplit -pc_fieldsplit_type multiplicative";
```

The desired lower block-triangular structure of the preconditioner is obtained by the use of PETSc keyword `multiplicative`. The strings `paramsXYZ` and `paramsP` respectively contain the innermost velocity and pressure block solvers options that will be detailed later on. The string `paramsKrylov` contains the definition of the Krylov subspace linear solver. For example, one should simply write `paramsKrylov = "-ksp_type fgmres"` to use the flexible GMRES. In order to implement the modified augmented Lagrangian preconditioner $\mathbf{P}_{\mathrm{mAL}}$ through the fieldsplit structure, in 3D, the four fields u, v, w, and p must be defined. Assuming the problem is formulated in the full vectorial finite element space `Wh`, containing the velocities and pressure unknowns, one must be able to differentiate the degrees of freedom belonging to each field. To that aim a finite element function taking a different integer value for each one of the four fields is defined in FreeFEM and passed to PETSc/SLEPc through the keyword `fields`. Then, for simplicity, each field is attributed a name that will be used to identify it when defining the different innermost solvers associated to the diagonal blocks of $\mathbf{P}_{\mathrm{mAL}}$, c.f. Section 4.3. Those names are contained in an array of strings, that is provided to the solver through the keyword `names`.

```
Wh [vX, vY, vZ, p] = [1, 2, 3, 4]; // numbering of each field
string[int] names(4); // prefix of each field
names[0] = "vX"; // x-velocity
names[1] = "vY"; // y-velocity
names[2] = "vZ"; // z-velocity
names[3] = "p"; // pressure
```

Approximate Schur complements The default setting in PETSc, when using a multiplicative fieldsplit preconditioner, is to define the preconditioner as the lower block triangular part of the system matrix in Eq. (12). Thus, on the block diagonal of such a preconditioner, one would have $\mathbf{A}_{\gamma,uu} + s\mathbf{M}_u$, $\mathbf{A}_{\gamma,vv} + s\mathbf{M}_v$, $\mathbf{A}_{\gamma,ww} + s\mathbf{M}_w$ and the null matrix. In order to implement $\mathbf{P}_{\mathrm{mAL}}$, one must replace, in the preconditioner only, the default operator for the pressure field (the null matrix) by the ones necessary to implement the desired Schur complement approximation Eq. (16). This is done in PETSc using `PCFieldSplitGetSubKSP` to retrieve the operators linked to each field of the fieldsplit structure and then `KSPSetOperators` to set the new operators that define Eq. (16).

When the shift s is null, for base flow computations, the approximate Schur complement only requires the assembly of one operator: $(\gamma + \mathcal{R}e^{-1})^{-1}\mathbf{M}_p$. This is done in FreeFEM as shown below, and then passed to PETSc with the keyword `schurPreconditioner`.

```
matrix[int] S(1); // array with a single matrix
varf vSchur(p, q) = int3d(th, qforder = 3)
  (-1.0/(gamma + 1.0/Re) * p * q); // Eq. (16) with s = 0
S[0] = vSchur(Qh, Qh); // matrix assembly
```

For eigenvalue computations, two auxiliary operators are now needed: $(\gamma + \mathcal{R}e^{-1})^{-1}\mathbf{M}_p$ and $s^{-1}\mathbf{L}_p$. The construction in FreeFEM is performed using the following lines, and then again passed to SLEPc with the keyword `schurPreconditioner`.

```
matrix<complex>[int] S(2); // array with two matrices
complex scale;
varf vMp(p, q) = int3d(th, qforder = 3)(scale * p * q); // Eq. (7)
scale = 1.0/(gamma + 1.0/Re);
S[0] = vMp(Qh, Qh); // first matrix assembly
macro grad(p)[dx(p), dy(p), dz(p)]// macro for computing ∇p
varf vLp(p, q) = on(3, p = 1) // inlet boundary condition
  + int3d(th, qforder = 2)(scale * (grad(p)' * grad(q)));
// shift value s
complex s = getARGV("-shift_real", 1.0e-6) + getARGV("-shift_imag", 0.6) * 1i;
scale = 1.0/s;
S[1] = vLp(Qh, Qh); // second matrix assembly
```

Finally, we note that the operators needed for the Schur complement approximation are built on the pressure space `Qh`. However, in FreeFEM, it is not possible to know a priori the correspondence between the numbering of `Wh`, where the full solution is defined, and `Qh`. To circumvent this issue, we compute this correspondence in FreeFEM from an interpolation between `Qh` and `Wh`, and then pass it to PETSc/SLEPc with the keyword `schurList`:

```
Qh pIdx; // function from the pressure space
pIdx[] = 1:pIdx[].n; // numbering of the unknowns of Qh
// renumbering into the complete space by doing an interpolation on Wh
Wh [listX, listY, listZ, listP] = [0, 0, 0, pIdx];
```

4.3. Innermost velocity and pressure linear solvers

The approximate inverses of the diagonal blocks in Eq. (15) are defined using off-the-shelf iterative methods from PETSc. For each velocity field, the GMRES is used, right-preconditioned by an additive Schwarz method (ASM) with one-level of algebraic overlap, as well as exact LU factorizations for each subdomain solver. These factorizations are carried out by MUMPS [18]. A maximum Krylov dimension of 50 is prescribed and the GMRES is stopped when the relative unpreconditioned residual norm is lower than 10^{-1}. In our implementation, the PETSc runtime options defining the approximate inverse of the diagonal velocity blocks are contained in the string `paramsXYZ` detailed below:

```
real tolV = getARGV("-velocity_tol", 1.0e-1); // default to 10^-1
// monodimensional velocity solver
string paramsV = "-ksp_type gmres -ksp_converged_reason -ksp_pc_side right " +
  "-ksp_rtol " + tolV + " -ksp_gmres_restart 50 -pc_type asm " +
  "-pc_asm_overlap 1 -sub_pc_type lu -sub_pc_factor_mat_solver_type mumps";
// each velocity component gets the same monodimensional solver
// defined by paramsV
string paramsXYZ = "-prefix_push fieldsplit_vX_ " + paramsV + " -prefix_pop"
                 + " -prefix_push fieldsplit_vY_ " + paramsV + " -prefix_pop"
                 + " -prefix_push fieldsplit_vZ_ " + paramsV + " -prefix_pop";
```

For the pressure Schur complement approximate inverse Eq. (16), the PETSc runtime options defining the solver are contained in the string `paramsP`. We must distinguish the cases of the base flow and eigensolvers. For the

former ($s = 0$), only the action of $\mathbf{M}_p{}^{-1}$ has to be evaluated. For that purpose, we use at most five iterations of the Jacobi-preconditioned conjugate gradient algorithm:

```
string paramsP = "-prefix_push fieldsplit_p_ " +
  "-ksp_type cg -ksp_max_it 5 -pc_type jacobi -prefix_pop";
```

For the eigensolver ($s \neq 0$), the action of the inverse of the Schur complement is approximated by the sum of the action of $(\gamma + \mathcal{R}e^{-1})\mathbf{M}_p{}^{-1}$ and $s\mathbf{L}_p{}^{-1}$. This is done through PETSc composite preconditioner:

```
string paramsP = "-prefix_push st_fieldsplit_p_ " +
  "-ksp_type preonly -pc_type composite " +
    "-prefix_push sub_0_ " + // action of (γ + Re^-1)M_p^-1
      "-pc_type bjacobi -sub_pc_type icc -prefix_pop " +
    "-prefix_push sub_1_ " + // action of sL_p^-1
      "-pc_type gamg -pc_gamg_square_graph 10 -prefix_pop " +
  "-prefix_pop";
```

Here only one application of the block Jacobi preconditioner with ICC(0) subsolvers [63] is used for approximating the mass matrix inverse while one V-cycle of GAMG [64] is used for the Laplacian term.

5. Numerical results

The efficiency of the modified augmented Lagrangian (mAL) preconditioner is investigated in this section by performing the linear stability analysis of two- and three-dimensional flow configurations described in Section 5.1. The two-dimensional computations are always performed on one process as they are of limited size. For the three-dimensional case, the fully parallel implementation presented in Section 4 is used. The influence of various numerical and physical parameters, such as the augmentation parameter, the mesh size and the Reynolds number, is first assessed in Section 5.2 for the two-dimensional configuration, before comparing the performance of mAL preconditioner with other block preconditioners (PCD, SIMPLE, Stokes) in Section 5.3. The efficiency of the parallel implementation is finally investigated in Section 5.4 for the three-dimensional configuration.

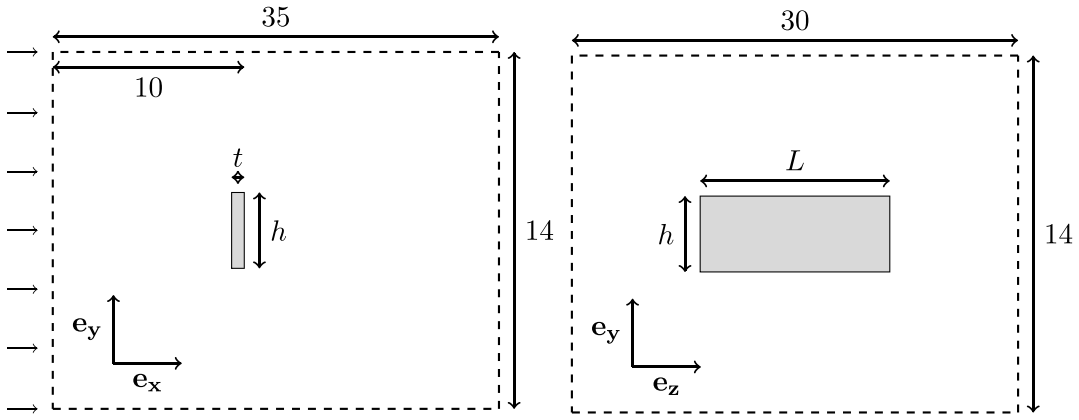
5.1. Two- and three-dimensional test cases

The two-dimensional flow configuration is sketched in Fig. 1(a). A thin plate of height h and thickness $t = h/6$ is immersed in an incoming flow of uniform velocity U_∞. The size of the computational box indicated in the figure and the flow variables are made non-dimensional using h as characteristic length and U_∞ as characteristic velocity, so that the Reynolds number is defined as $\mathcal{R}e = U_\infty h/\nu$, where ν is the kinematic viscosity.

Triangulations of the computational domain are obtained with the internal mesh generator of FreeFEM. The no-slip boundary condition $u = v = 0$ is applied on the plate, symmetry boundary conditions ($\partial_y u = 0$ and $v = 0$) are applied at the top and bottom boundaries of the computational domain, and a stress-free boundary condition is applied at the outlet boundary.

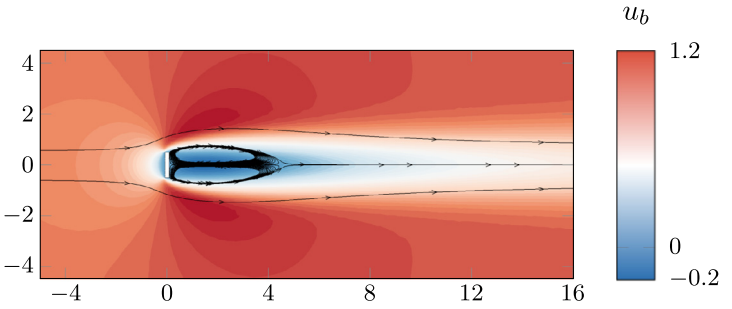
A typical steady solution of the incompressible Navier–Stokes equation is displayed in Fig. 2(a) for $\mathcal{R}e = 40$. The flow recirculates in two symmetric regions in the wake of the plate, as indicated by the streamlines. The linear stability analysis of this base flow yields the eigenvalue spectrum shown in Fig. 2(c) with circles. A pair of complex conjugate unstable eigenvalues is found ($\lambda \geqslant 0$) characterized by an angular frequency $\omega = 0.70$. For a lower value of the Reynolds number $\mathcal{R}e = 30$, this eigenvalue is stable as shown by the square symbols. The real part of the eigenmode associated to this leading eigenvalue is depicted in Fig. 2(b) with isocontours of the streamwise velocity. The spatial structure of this eigenmode breaks the symmetry of the steady solution and is responsible for the onset of the well-known Von Kármán vortex-street that becomes visible behind bluff bodies once the exponential growth of the linear instability saturates due to nonlinearities.

The three-dimensional flow configuration is a plate of height and thickness identical to the two-dimensional plate, but of finite length L in the spanwise direction z, as sketched in Fig. 1(b). The computational domain is discretized using Gmsh [65] by a Delaunay mesh composed of 17 million tetrahedra, which are then partitioned between processes with ParMETIS [66]. Using Taylor–Hood finite element pair, cf. Section 2.2, this leads to a

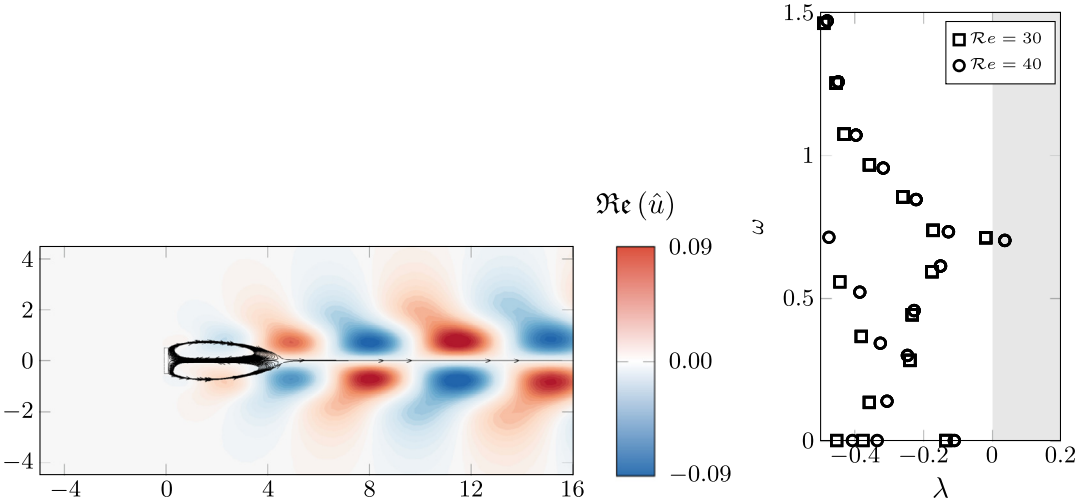


(a) Two-dimensional geometry (b) Three-dimensional geometry (span view)

Fig. 1. Two-dimensional and three-dimensional flow configurations. Sketch of the computational domains used for (a) the two-dimensional plate of height $h = 1$ and thickness $t = 1/6$ and (b) the three-dimensional plate of span $L = 2.5$ immersed in an upstream uniform streamwise flow U_∞.

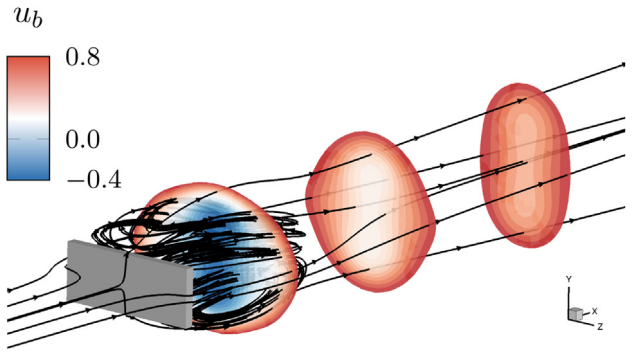


(a) Base flow ($\mathcal{R}e = 40$)

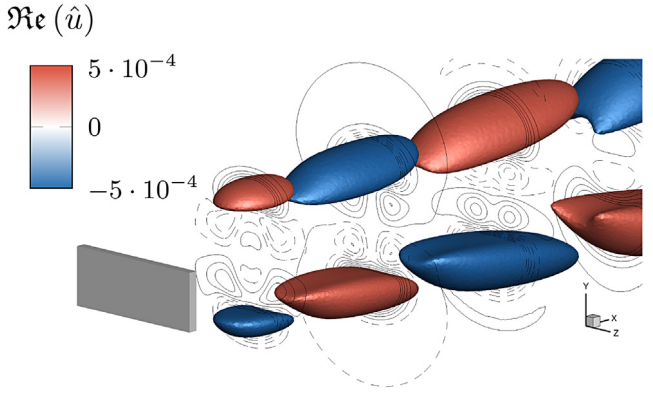


(b) Leading eigenmode ($\mathcal{R}e = 40$) (c) Eigenvalue spectra

Fig. 2. Results of the linear stability analysis for the two-dimensional configuration. Streamwise velocity u of (a) the steady-state solution and (b) the real part of the unstable eigenmode. (c) Eigenvalues are depicted with circles in the complex plane (growth rate λ and frequency ω). The unstable region is shown in gray. Only eigenvalues with positive frequencies are shown, the spectrum being symmetric.



(a) Steady solution



(b) High-frequency unstable eigenmode

Fig. 3. Linear stability analysis for the three-dimensional flow around of plate of spanwise length $L = 2.5$ and $\mathcal{R}e = 100$. Streamwise velocity contours of (a) the steady solution and (b) the high-frequency unstable eigenmode ($\omega = 0.57$) are presented.

total of 75 million unknowns. The boundary conditions are similar to those detailed above for the two-dimensional configuration.

The linear stability analysis of this flow configuration has been performed by [27] who determined the neutral curves of various unstable eigenmodes in the range of Reynolds number $40 \leq \mathcal{R}e \leq 200$ and length $1 \leq L \leq 6$. Here, we specifically investigate the plate of length $L = 2.5$ for the Reynolds number $\mathcal{R}e = 100$. The steady solution, depicted in Fig. 3(a), exhibits a large three-dimensional recirculation region in the wake of the plate. The stability analysis performed in [27] revealed that two pairs of complex eigenvalues get unstable above $\mathcal{R}e \simeq 101$ for this parameter choice, with respective angular frequencies of $\omega \simeq 0.3$ and $\omega \simeq 0.57$. Hereinafter, we focus on the high-frequency eigenmode, depicted in Fig. 3(b). As shown in the figure, the three-dimensional eigenmode breaks the top/bottom symmetry of the steady-state solution, as for the two-dimensional plate.

5.2. *Influence of numerical and physical parameters*

We investigate in this section the influence of various numerical and physical parameters on the performance of the mAL preconditioner. Tests are performed on the two-dimensional flow configuration previously introduced. The effect of the augmentation parameter on the preconditioner efficiency and solution accuracy is reported in Section 5.2.1. The performance of the preconditioner is tested in Section 5.2.2 for many values of the complex shift parameter used in the shift-and-invert strategy to compute the leading eigenvalues. The behavior of the preconditioner in regards to the mesh refinement and the Reynolds number is finally tested in Section 5.2.3.

In all the numerical tests performed in this section, the full GMRES without restart is used in order to fairly assess the performance of the preconditioner. The diagonal blocks defined by the mAL preconditioner Eq. (15) are here inverted using the sparse direct solver MUMPS.

5.2.1. Effect of the augmentation parameter

The effect of the augmentation parameter γ on the performance of the mAL preconditioner is first assessed by considering the number of GMRES iterations. Using the Newton method described in Section 2.3, steady solutions are computed for the Reynolds number $\mathcal{R}e = 40$ and several values of the augmentation parameters reported in the first columns of Tables 1a and 1b, that correspond to results obtained with a coarse mesh (14,674 triangles) and a finer mesh (128,874 triangles), respectively. The GMRES relative tolerance being fixed to 10^{-6}, the average numbers of inner (GMRES) iterations per outer (Newton) iteration are reported in the second columns of those tables. For both meshes, there exists an optimal value of the augmentation parameter, $\gamma \simeq 1$, for which a minimum number of iterations is reached. Similar observations are reported in other studies [44–47,49] for different flow configurations such as the lid-driven cavity flow, the backward facing step or the flow over a flat plate. Note also that the number of iterations is quite similar for the coarse and fine meshes, regardless of the augmentation parameter value. For the optimal γ, the average number of inner iterations is around 50.

As briefly discussed in Section 2.2, the introduction of the grad–div stabilization term in the weak formulation (4a) does not modify the conservation of momentum at the continuous level, since the continuous solution is divergence-free. However, with the spatial discretization chosen in the present study (Taylor–Hood finite element), the divergence of the velocity is only weakly satisfied and the grad–div stabilization term modifies the discrete momentum equation. The augmentation parameter has therefore an influence on the accuracy of the discrete solution. To assess this effect, a reference steady solution, denoted $(\mathbf{u}_b^r, p_b^r)$, is computed without stabilization parameter ($\gamma = 0$) on a very-fine mesh made of 512,872 triangles. The corresponding leading eigenvalue denoted σ^r is also computed. The two last columns of Tables 1a and 1b report the relative errors of the steady velocity and the leading eigenvalue computed with the coarse and fine meshes, respectively, for several values of γ. Examining first the results obtained with the coarse mesh (see Table 1a), a minimal error is obtained for $\gamma \simeq 1$, not only for the steady solution but also for the leading eigenvalue. When the mesh is refined (see Table 1b), a minimal error is still obtained for $\gamma \simeq 1$, although less pronounced. Compared to results obtained with the coarse mesh, the relative error is decreased whatever the value of the augmentation parameter. As expected, the augmentation parameter less affects the accuracy of the discrete solution when the mesh is refined, since the discrete solution tends towards the continuous solution. It is worth noticing that the use of the stabilization term can significantly improve the accuracy of the solution. For instance, the accuracy of the eigenvalue obtained for the coarse mesh with $\gamma = 1$ is identical to the one obtained for the fine mesh without stabilization $\gamma = 0$. In other words, the same accuracy is obtained but with ten times fewer mesh elements.

The present results clearly indicate that $\gamma \simeq 1$ is an optimal value from both the solution accuracy point of view and the preconditioning efficiency point of view when considering not only steady solutions, as reported before [44], but also leading eigenvalues. As a consequence, in the following, we consider that γ can be chosen on preconditioning efficiency criteria only without compromising accuracy.

5.2.2. Effect of the shift parameter

The shift-and-invert strategy, adopted in the present study to compute the eigenvalues with largest real part, requires to specify the complex value $s = s_r + \mathbf{i}s_i$ that appears in the spectral transformation Section 2.4. When investigating the transition of a steady solution from a stable to an unstable state, a common practice is to choose the shift parameter as a pure imaginary number, i.e. $s = \mathbf{i}s_i$, and to vary the imaginary part in order to compute complex eigenvalues with growth rates close to $\lambda = 0$. Depending on the flow configuration investigated, the steady solution may get unstable for eigenmodes characterized by very different frequencies. Ideally, the number of preconditioned GMRES iterations should be insensitive to the value of the complex shift, for the Krylov–Schur algorithm to converge rapidly whatever the eigenvalue of interest. To the best of our knowledge, only the case of a real-valued shift has been considered so far, either positive when solving the unsteady Oseen problem [44,45] or negative when solving the linearized Navier–Stokes equation [51].

Here, we vary s in the whole complex plane and assess its effect on the efficiency of the mAL preconditioner by performing the following numerical experiment. The linear system Eq. (12) is solved with right-hand side vectors

Table 1
Effect of the grad–div augmentation parameter γ on the mAL preconditioning efficiency and the solution accuracy. For both tables, the first column indicates values of γ. The second column represents the average number of mAL preconditioned GMRES iterations per Newton iteration. The last two columns give the relative errors of the steady solution and the leading eigenvalue compared with a reference solution $(\mathbf{u}_b^r, \sigma^r)$ computed on a very fine mesh without stabilization ($\gamma = 0$).

γ	# of GMRES iterations	$\frac{\left\|\mathbf{u}_b^h - \mathbf{u}_b^r\right\|_2}{\left\|\mathbf{u}_b^r\right\|_2}$	$\frac{\left\|\sigma^h - \sigma^r\right\|}{\left\|\sigma^r\right\|}$	# of GMRES iterations	$\frac{\left\|\mathbf{u}_b^h - \mathbf{u}_b^r\right\|_2}{\left\|\mathbf{u}_b^r\right\|_2}$	$\frac{\left\|\sigma^h - \sigma^r\right\|}{\left\|\sigma^r\right\|}$
0	860	$2.8 \cdot 10^{-4}$	$9.8 \cdot 10^{-4}$	873	$3.2 \cdot 10^{-5}$	$1.1 \cdot 10^{-4}$
10^{-1}	191	$2.1 \cdot 10^{-4}$	$7.9 \cdot 10^{-4}$	194	$2.7 \cdot 10^{-5}$	$1.1 \cdot 10^{-4}$
10^{0}	52	$8.4 \cdot 10^{-5}$	$1.1 \cdot 10^{-4}$	53	$1.4 \cdot 10^{-5}$	$5.4 \cdot 10^{-5}$
10^{1}	337	$4.4 \cdot 10^{-4}$	$1.5 \cdot 10^{-3}$	363	$2.7 \cdot 10^{-5}$	$8.5 \cdot 10^{-5}$
10^{2}	>1000	$1.3 \cdot 10^{-3}$	$4.6 \cdot 10^{-3}$	>1000	$6 \cdot 10^{-5}$	$2.1 \cdot 10^{-4}$

(a) Coarse mesh (14,674 triangles)

(b) Fine mesh (128,874 triangles)

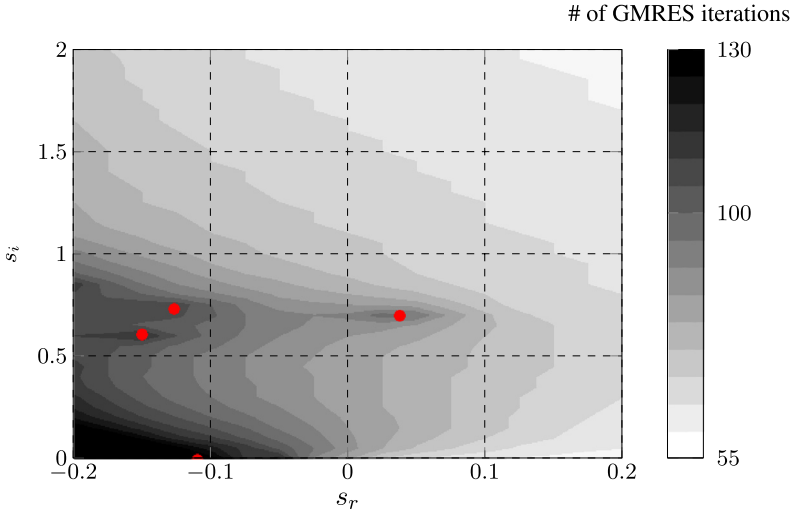
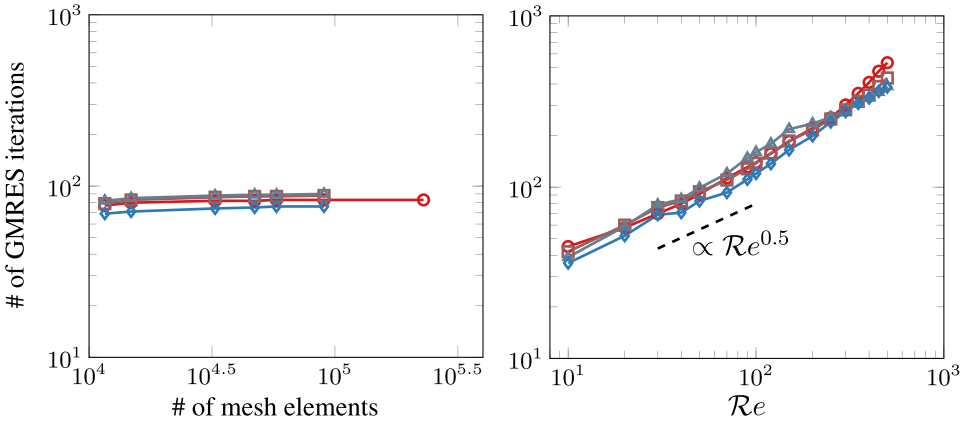


Fig. 4. Influence of the complex shift. Isocontours represent the number of GMRES iterations needed to solve the linear system (13) with the mAL preconditioner, depicted in the complex plane (s_r, s_i), and computed for $\mathcal{R}e = 40$ and $\gamma = 0.7$. The red circles are the eigenvalues of the Jacobian matrix (6).

whose coefficients are randomly generated in $[0, 1] + [0, 1]\mathbf{i}$, as done for instance in [67]. The Jacobian matrix $\mathbf{J}$ of the linear system is computed with the steady solution at $\mathcal{R}e = 40$ and the augmentation parameter $\gamma = 0.7$. In other words, only the inner solver is studied, no outer iteration (Newton or Krylov–Schur) is performed. The isocontours shown in Fig. 4 in the complex plane (s_r, s_i) correspond to the number of inner (GMRES) iterations required to decrease the relative residual to 10^{-6}. The red circles are eigenvalues of the Jacobian matrix. First, the number of iterations increases when the shift gets closer to any eigenvalue. In that case, the matrix $\mathbf{J} + s\mathbf{M}$ involved in the spectral transformation 2.4 becomes singular, leading to a very ill-conditioned linear system and thus high iteration counts. Second, the number of iterations is reduced when increasing s_r. Solving the linear system for $s_r < 0$ is generally more expensive than for $s_r > 0$. According to [51], this is due to the indefiniteness of the velocity block $\mathbf{A}_\gamma + s\mathbf{M}_\mathbf{u}$ in Eq. (12) for sufficiently large negative values of s_r. On the contrary, the velocity block is definite when $s_r > 0$. The contribution of the shift can be interpreted as a (positive definite) time-step term, which reinforces the diagonal dominance of the problem and thus improve its spectral properties. For more details, the reader can refer to [45, section 2.6], where the mAL preconditioner is used to solve the unsteady Oseen problem. Finally, no particular trend is observed in the number of inner iterations when fixing the real part s_r and varying the imaginary part s_i, except when getting closer to an eigenvalue. For $s_r = 0$, the number of iterations is roughly constant for $s_i < 0.5$ and increases around $s_i = 0.6$ due to the proximity of the unstable eigenvalue marked by the red circle. By further increasing s_i, the number of iterations then decreases.



(a) Mesh dependence ($\mathcal{R}e = 40$) (b) Reynolds dependence (14,828 triangles)

Fig. 5. Effect of mesh refinement (a) and Reynolds number (b) on mAL preconditioning efficiency. The GMRES iteration count for solving Eq. (12) to a relative tolerance of 10^{-6} is presented for different shifts: $s = 0$ (—o—), $s = 0.3\mathbf{i}$ (—□—), $s = 0.6\mathbf{i}$ (—△—) and $s = 1\mathbf{i}$ (—◇—).

5.2.3. *Effect of the mesh refinement and Reynolds number*

The modified augmented Lagrangian preconditioner allows to compute steady solution in a number of GMRES iterations independent of the mesh refinement, as previously observed in Table 1, and mildly dependent of the Reynolds number [43,45]. The influence of the mesh refinement and Reynolds number on the number of iterations needed to solve the complex-shifted linear system Eq. (12) has not been investigated so far. The numerical experiment consists in solving the linear system to the relative tolerance of 10^{-6} for right-hand side vectors with randomly generated coefficients as explained before. First, the Reynolds number is fixed ($\mathcal{R}e = 40$) as well as the augmentation parameter ($\gamma = 0.7$) while the mesh refinement changes. The number of inner GMRES iterations is reported in Fig. 5(a) as a function of the number of triangles. The curves correspond to different values of the (purely imaginary) shift. Clearly, the iteration count is independent of the mesh refinement, regardless of the shift. Second, a fixed mesh refinement is chosen (14,828 triangles) and the linear system is solved for different values of the Reynolds number in the range [10; 500] for different shift values. As reported in [46] when computing steady solutions, the optimal value of the augmentation parameter that minimizes the number of iterations depends on the Reynolds number. The optimal value of γ has been first determined for each value of $\mathcal{R}e$. For $\mathcal{R}e = 10$, the optimal value is $\gamma \simeq 1.2$ and it decreases to $\gamma \simeq 0.4$ for $\mathcal{R}e = 500$. These optimal values of γ have been determined for $s = 0$ but are used in the following regardless of the values of s. The number of iterations is depicted in Fig. 5(b) as a function of the Reynolds number. The mAL preconditioner shows a mild degradation of its performance as $\mathcal{R}e$ increases, independently of the values of the shift. The increase of the number of inner iterations is proportional to $\mathcal{R}e^{0.5}$ in this numerical experiment.

As a conclusion, the modified augmented Lagrangian preconditioner exhibits interesting properties for performing efficiently a linear stability analysis using a shift-and-invert strategy: robustness with respect to a complex-valued shift, mesh independence, and a mild deterioration as $\mathcal{R}e$ increases.

5.3. *Comparison with other block preconditioners*

The mAL preconditioner is one of many other preconditioners developed to solve the steady incompressible Navier–Stokes equations. Among them, we select the Pressure Convection–Diffusion (PCD) [41] and the SIMPLE [40] preconditioners, widely used and easily implemented, and compare their performance with those of the mAL preconditioner. In addition, we also test the unsteady Stokes preconditioner [31] which has gained in popularity in the hydrodynamic stability community [68], as it can be easily implemented using existing time-steppers so as to compute base flows and leading eigenvalues [15]. In our implementation however, the Stokes preconditioner is itself applied using a nested Krylov subspace method instead of an existing time-stepper. More

details on those preconditioners are given in Appendix B but it is worth recalling here that they are designed for classical Galerkin discretization of the Navier–Stokes equations. Therefore, their application to the iterative solution of Eq. (12) is meant for $\gamma = 0$, i.e. without grad–div stabilization terms.

The numerical test case consists in solving iteratively Eq. (12) with a random right-hand side vector, as detailed before, for a large relative tolerance equal to 10^{-3} to limit the number of iterations. The shift is fixed to $s = 0$ since it was shown in Section 5.2.3 that the mAL preconditioner depends very weakly on the shift parameter when the mesh is refined or the Reynolds number is increased. All innermost block solutions are performed using exact LU factorizations. The PCD and SIMPLE preconditioners are parameter-free, unlike the mAL and Stokes preconditioners. For the latter, the optimal values of the parameter (γ for mAL and a time-step like parameter for Stokes) are determined for each values of the Reynolds number.

The effect of the mesh refinement and Reynolds number on the number of inner iterations, studied in the previous paragraph for the mAL preconditioner, is assessed here for all the other preconditioners. Results are compared in Fig. 6. This number of iterations is a good measure to compare the efficiency of the different preconditioners, in a first approximation, because for each inner iteration, the application of all preconditioners requires the solution of subproblems with similar complexities.[4] Therefore, the computational time of one inner iteration is roughly similar for all preconditioners.

All the preconditioners are independent of the mesh refinement, as shown in Fig. 6(a), except for the SIMPLE preconditioner for which the number of inner iterations slightly increases with the number of triangles. Interestingly, the number of iterations is significantly less for mAL and PCD (around 50) than for Stokes and SIMPLE (around 1000). Note that for the two preconditioners depending on a parameter (mAL and Stokes), their optimal value was found to be independent of the mesh refinement.

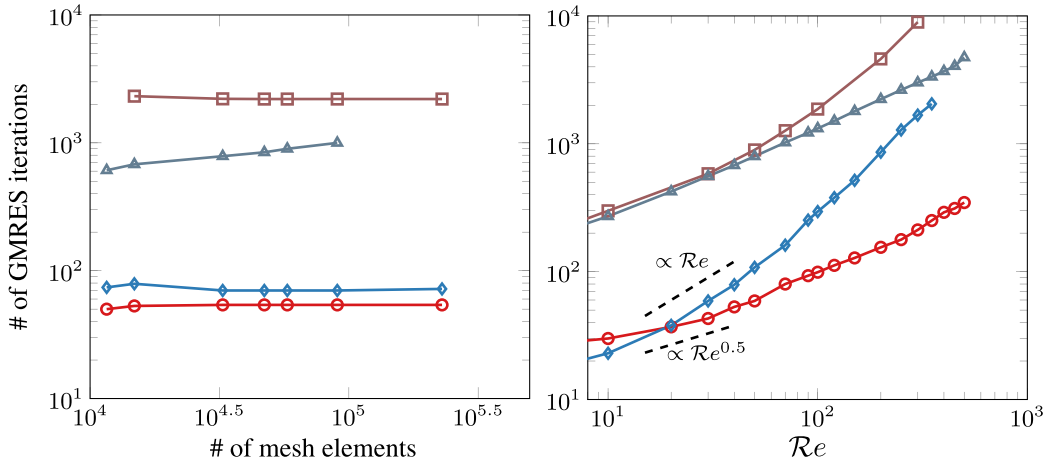
The effect of the Reynolds number is reported in Fig. 6(b). For all tested preconditioners, the number of iterations increases with the Reynolds number, but with different slopes. The mAL preconditioner exhibits the best performance for all Reynolds numbers, except for low Reynolds number ($\mathcal{R}e < 20$) where PCD is more efficient. However, the number of iterations obtained with the PCD preconditioner increases strongly for larger values of the Reynolds number ($\mathcal{R}e > 80$). The mAL and SIMPLE preconditioners exhibit a similar trend: the number of iterations scales with the Reynolds number as $\mathcal{R}e^{0.5}$. However, it is significantly larger with SIMPLE than with mAL, regardless of the Reynolds number. At low $\mathcal{R}e$, the Stokes preconditioner behaves similarly to the SIMPLE preconditioner, but for larger $\mathcal{R}e$, it degrades significantly and exhibits the same trend as the PCD preconditioner. Finally, when considering the number of iterations, the mAL preconditioner is undoubtedly the best preconditioner. We note that, contrary to the mesh dependence study, the optimal parameters of mAL and Stokes showed some variations with respect to $\mathcal{R}e$.

Let us now compare the computational time for applying the four preconditioners. To that aim, the direct LU factorizations used until now for the innermost velocity blocks solvers are replaced by GMRES right-preconditioned with an $ILU(2)$ method (as implemented in PETSc). The choice of an innermost iterative solution allows for a more comprehensive interpretation of the computational time, since it accounts for the various complexities in solving iteratively the velocity blocks involved in the different preconditioners. Moreover, such an innermost iterative solution is necessary when considering large-scale three-dimensional problems, as shown in the next section. The relative tolerance of the inner (resp. innermost) GMRES is fixed to 10^{-3} (resp. 10^{-2}). The computational times obtained with the four preconditioners are reported in Fig. 7(a) for $\mathcal{R}e = 40$ (left) and $\mathcal{R}e = 100$ (right). The total time is split into the time spent in computing matrix–vector products, in applying the global preconditioner, and in constructing the global Krylov subspace. The inner iteration counts are given between parenthesis. Note that it may be slightly higher than what is presented in Fig. 6 since the velocity blocks are now solved only approximately. All computations are run on a standard laptop computer.

For the low Reynolds number $\mathcal{R}e = 40$, the mAL and PCD preconditioners are about ten times faster than the SIMPLE and Stokes preconditioners.[5] For this Reynolds number, the PCD preconditioner is comparable with the mAL preconditioner, as it is only 40% slower. However, when the Reynolds number is increased to $\mathcal{R}e = 100$,

[4] Two scalar velocity solves and one pressure Schur complement solve for mAL and Stokes; one vectorial velocity solve and one pressure Schur complement solve for PCD and SIMPLE.

[5] Note that the time for building the inner Krylov subspace is very small for the Stokes preconditioner at $\mathcal{R}e = 40$, despite a large number of iterations. This is due to the fact that, in the implementation detailed in Appendix B, the Stokes preconditioner is itself applied with a nested iterative method. Therefore, no Krylov subspace of dimension 947 is actually built.



(a) Mesh dependence ($\mathcal{R}e = 40$) (b) Reynolds dependence (14,828 triangles)

Fig. 6. Influence of (a) the number of mesh elements and (b) the Reynolds number $\mathcal{R}e$ on the number of inner iterations required to solve Eq. (12) with the mAL (—o—), PCD (—♦—), SIMPLE (—▲—), and Stokes (—□—) preconditioners. The relative tolerance is set to 10^{-3} and the shift to zero.

the performance of PCD degrades significantly with respect to mAL, as it is now about five times slower. The computational times are not given for the SIMPLE and Stokes preconditioners because they largely exceed 2100 seconds. The deterioration in the computational time of the PCD preconditioner when the Reynolds number is increased, is in agreement with the growth in the number of iterations observed before. For even higher Reynolds numbers $\mathcal{R}e > 100$, the mAL preconditioner is expected to be increasingly more interesting than its competitors.

As a conclusion, this benchmark shows that, compared to other widely used preconditioners, mAL provides a more efficient approach for solving Eq. (12) on a configuration typical of two-dimensional external flows around bluff bodies. In particular, among the alternatives tested here, it is the only preconditioner combining a mesh-independent iteration count and a mild degradation with $\mathcal{R}e^{0.5}$, making it the most efficient preconditioner for $\mathcal{R}e \geqslant 20$.

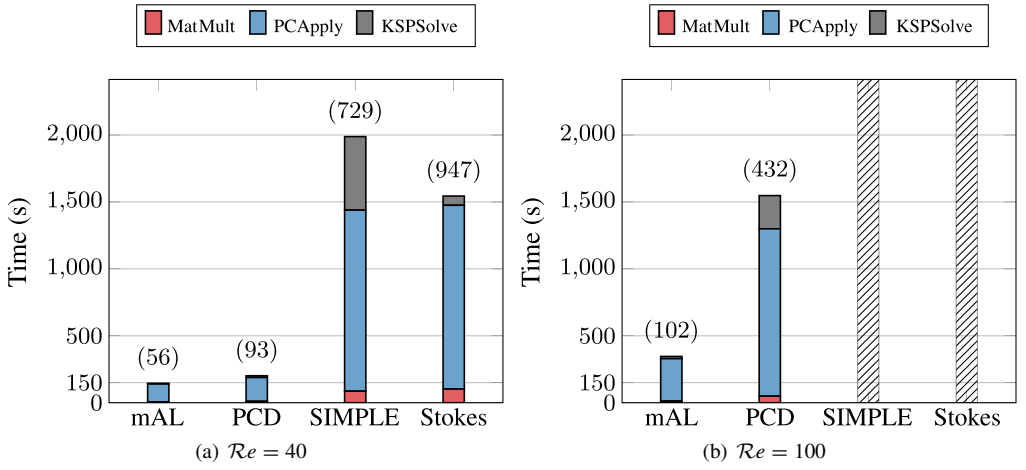
5.4. Performance of the parallel implementation

In this section, the performance of the parallel implementation detailed in Section 4 is tested on the three-dimensional configuration presented in Section 5.1. First, a coarse mesh is used, in order to be able to compare our approach with the direct parallel solver MUMPS. Then, the full size 3D configuration presented before is considered to test the parallel performance of our approach on a problem that a direct solver could not handle at a reasonable memory cost.

5.4.1. Comparison with a direct solver on a small-scale 3D configuration

Despite its large memory requirements, some authors have used the "matrix-based" approach, combined with direct solvers for the arising linear systems, to perform the stability analysis of three-dimensional flows [27,69]. In this section, we aim at comparing the performance of this approach to ours. To that end, the three-dimensional test case is considered using a coarse mesh of 1.1 million tetrahedra (4.8 million unknowns), in order to keep the memory consumption of the direct solver reasonable. The computations are performed on Sator, an ONERA cluster composed of 620 nodes with two fourteen-core Intel Broadwell clocked at 2.4 GHz. The direct solver we compare ourselves to is MUMPS.

5.4.1.1. Nonlinear solver. In this section, the Newton nonlinear tolerance and the GMRES relative tolerance are both set to 10^{-6}. In Fig. 8(a) we report the average wall-clock time per Newton iteration for the mAL and MUMPS approaches, as a function of the number of processes. On the top x-axis, we report the amount of available memory



(a) $\mathcal{R}e = 40$ (b) $\mathcal{R}e = 100$

Fig. 7. Comparison of the performance of various preconditioners on the two-dimensional test case (14,828 triangles). The total time is split between matrix–vector products, applying the preconditioner, and building the Krylov subspace. The number of global GMRES iterations is given between parenthesis. System Eq. (12) is solved to a relative tolerance of 10^{-3}. For $\mathcal{R}e = 100$, the hatched bars correspond to preconditioners for which the total time largely exceeded 2100 seconds and is not reported in detail. The velocity blocks in the preconditioners are solved iteratively to a relative tolerance of 10^{-2} using an innermost GMRES, right-preconditioned with $ILU(2)$. The pressure blocks are solved exactly with MUMPS. These times will depend on the particular preconditioners used for solving the diagonal blocks. Therefore, those results should be considered qualitatively.

corresponding to each number of processes. First, as expected, the memory requirements of our approach are lower than with MUMPS: we observe that MUMPS cannot be run on less than 224 processes, which corresponds to an available memory of 1024 GB, whereas the mAL approach can be run on 28 processes (128 GB). Note that the memory requirements of the mAL approach could be even lower by using iterative methods for the subdomain solvers of the innermost ASM-preconditioned GMRES iterations. Moreover, thanks to good scalability properties and the absence of a full LU factorization at each Newton iteration, the mAL approach is clearly faster than MUMPS (about ten times with 448 processes).

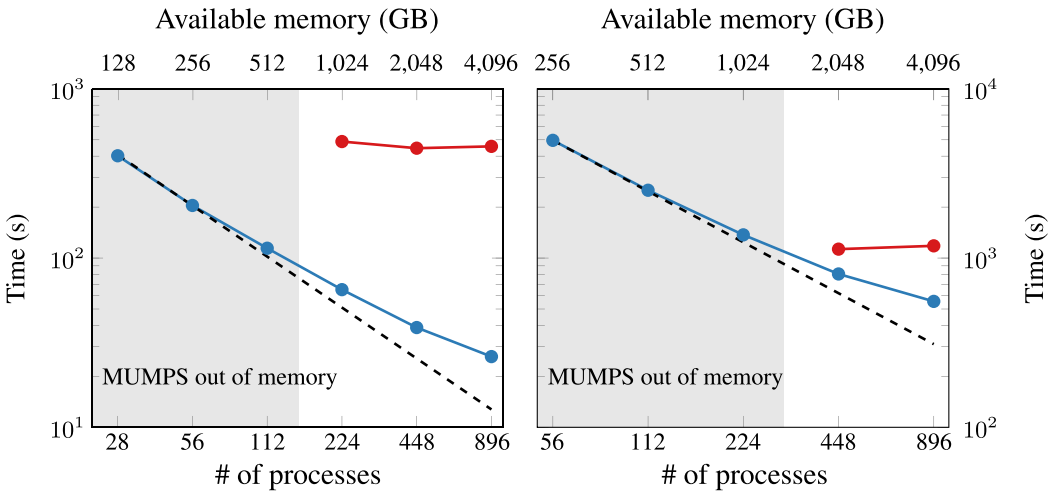
5.4.1.2. Eigensolver. For the eigensolver, the Krylov–Schur tolerance is 10^{-6} whereas the inner relative tolerance is 10^{-3}. Note that we use a larger tolerance for the inner linear solve than for the outer Krylov–Schur solver. Indeed, contrary to what is often recommended in the literature (e.g., [70, §3.4.1]), we observed that it was not necessary to use a smaller tolerance for the inner solution of Eq. (12) in order to keep a satisfying accuracy on the computed eigenvalues. More details on that aspect may be found in Appendix C.

We show in Fig. 8(b) the total wall-clock time for computing 5 eigenvalues closest to the shift $s = 0.6\mathbf{i}$, using mAL and MUMPS as inner solvers, as a function of the number of processes. The available memory is again reported on the top x-axis. Similar conclusions as for the nonlinear solver can be made for memory consumption with a multiplication factor of two, due to the use of complex instead of real algebra. From a wall-clock time point of view, we observe, as for the nonlinear solver, that the mAL approach possesses much better scalability than MUMPS, which leads to a faster computation. We note however that MUMPS is harder to beat with an iterative approach when used in the eigensolver than in the nonlinear solver. The reason is that, in the Krylov–Schur method, the very high cost of forming the full LU factorization is greatly amortized by the many inner solves realized with it, whereas in the Newton method, each inner solve requires to build the factorization again.

As a conclusion, the mAL approach presents the double advantage of being much less memory-intensive than the direct solver and also faster, even for the unfavorable case of eigenvalue computations.

5.4.2. Parallel performance on a large-scale 3D configuration

In this section, the 3D plate configuration is used, with a fine mesh, resulting in 75 million unknowns. The parallel performance of our implementation is investigated for the nonlinear base flow solver and eigenvalue solver.



(a) Nonlinear solver (average time per Newton iteration) (b) Eigensolver (overall time for 5 eigenvalues, $s = 0.6\mathbf{i}$)

Fig. 8. Comparison of mAL (—●—) and MUMPS (—●—) as inner solvers on (a) the nonlinear solver and (b) the eigensolver. The computation is performed on the three-dimensional test case using a coarse mesh (4.8 million unknowns) and $\gamma = 0.3$. The gray area indicates when the LU factorization is not feasible due to too large memory requirements. The dashed line represents ideal scalability.

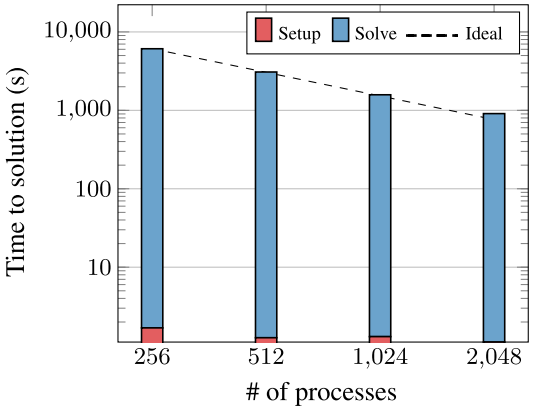
Results were obtained on Curie, a system composed of 5040 nodes with two eight-core Intel Sandy Bridge clocked at 2.7 GHz. The interconnect is an InfiniBand QDR full fat tree and the MPI implementation exploited was bullxMPI version 1.2.9.2. All binaries and shared libraries were compiled with Intel compilers and Math Kernel Library support (for dense linear algebra computations) version 18.0.1.163. Recent releases of FreeFEM and PETSc/SLEPc were used (version 3.61 and 3.9.3 respectively). In all following plots and tables, the time spent in the finite element kernel is never accounted for because we are mostly interested in the performance of the preconditioner. Only the time spent in PETSc or SLEPc is reported.

5.4.2.1. Nonlinear solver. In this paragraph, we investigate the parallel performance of the nonlinear steady-state solver. The inner Krylov solver is the flexible GMRES algorithm [61], which is stopped when the relative unpreconditioned residual norm is lower than 10^{-1}. The Newton method is stopped when the l^2 norm of the residual is lower than 10^{-6}. As an initial guess for the computation at $\mathcal{R}e = 100$, a solution at a lower Reynolds number $\mathcal{R}e = 50$ is first computed using a higher nonlinear outer tolerance of 10^{-4}. Also note that in this preprocessing step, all the domain decomposition information obtained from ParMETIS partitioning is dumped and will be used in successive runs for the nonlinear and generalized eigenvalue solvers. In Table 2, the numerical performance of the nonlinear solver is reported. One may notice that even if a high relative tolerance is used to stop the flexible GMRES, very few Newton iterations (second column) are needed for the solver to converge, independently of the number of subdomains (first column). The number of mAL-preconditioned inner iterations needed to reach convergence, averaged over all Newton iterations, is reported in the third column. It is seen not to depend on the number of processes. Eventually, in the last three columns, we show the average number of ASM-preconditioned innermost iterations needed for each velocity block of $\mathbf{P}_{\text{mAL}}$ to reach the desired convergence tolerance of 10^{-1} (see 4.3). A slight increase is observed with the number of processes. This is an expected feature of simple one-level domain decomposition methods, like the additive Schwarz method, that are known to not scale numerically [71].

In Fig. 9, the scalability of our implementation is shown, using the run with 256 processes as the reference, and going up to 2048 processes. The parallel efficiency of this approach remains above 83%. The fact that one additional Newton iteration is needed with 256 processes has to be highlighted, since it does improve the efficiency. Other than that, because exact LU factorizations are used as subdomain solvers in the additive Schwarz method used for each velocity field, the setup phase scales superlinearly (see the second column of the table in Fig. 9). Moreover,

Table 2
Numerical performance of the 3D nonlinear solver with respect to the number of processes ($\mathcal{R}e = 100$). The second column represents the number of Newton outer iterations, the third is the number of mAL-preconditioned inner GMRES iterations per Newton step. The last three columns correspond to the average number of ASM-preconditioned innermost GMRES iterations for each velocity block per inner iteration.

P	# of Newton iterations	# of iterations per Newton it.	# of iterations per field (x, y, z)		
256	6	83	30	12	19
512	5	81	31	13	20
1024	5	84	35	15	21
2048	5	84	44	17	27



P	Setup (s)	Solve (s)	Speedup
256	364.6	5,739.9	–
512	88.9	2,994.9	2.0
1024	56.6	1,523.6	3.9
2048	13.9	895.1	6.7

Fig. 9. Scalability of the 3D nonlinear solver with respect to the number of processes.

because the number of iterations needed for the corresponding solvers only grows slightly, as shown in the three last columns from Table 2, the solution phase also scales appropriately.

5.4.2.2. Eigensolver. We now evaluate the parallel performance of the eigensolver. The tolerance on the Krylov–Schur algorithm is 10^{-6} whereas the relative tolerance for the inner linear solver is set[6] to 10^{-4}. The main difference with the Newton method is that the multiple inner linear solves involve the very same shifted operator $(\mathbf{J}+s\mathbf{M})$ and preconditioner $\mathbf{P}_{\text{mAL}}$. To improve the performance of the eigensolver, let us show first the effect of using a recycled Krylov method for solving these systems, instead of the standard GMRES. This can be done by switching from the KSP objects of PETSc to the iterative methods of the HPDDM library [72] which handle subspace recycling. In particular, the flexible GCRO-DR method is used, with a recycled subspace between each linear solves of dimension five. The following lines allow to switch between PETSc and HPDDM Krylov methods from within FreeFEM:

```
int recycle = getARGV("-recycle", 0); // use GMRES by default
int restart = getARGV("-restart", 200); // default to 200
real tolInner = getARGV("-inner_tol", 1.0e-4); // default to 10^-4
string paramsKrylov = (recycle == 0 ? "-st_ksp_type fgmres " +
    "-st_ksp_monitor -st_ksp_rtol " + tolInner +
    " -st_ksp_gmres_restart " + restart + " -st_ksp_max_it 1000"
    :
    "-st_ksp_type hpddm -hpddm_st_krylov_method gcrodr " +
    "-hpddm_st_recycle " + recycle + " -hpddm_st_max_it 1000" +
    " -hpddm_st_verbosity 4 -hpddm_st_gmres_restart " + restart +
    " -hpddm_st_tol " + tolInner + " -hpddm_st_variant flexible");
```

[6] For the same reasons explained in 5.4.1.2, we use a larger tolerance for the inner linear solves than for the outer Krylov–Schur algorithm (see also Appendix C).

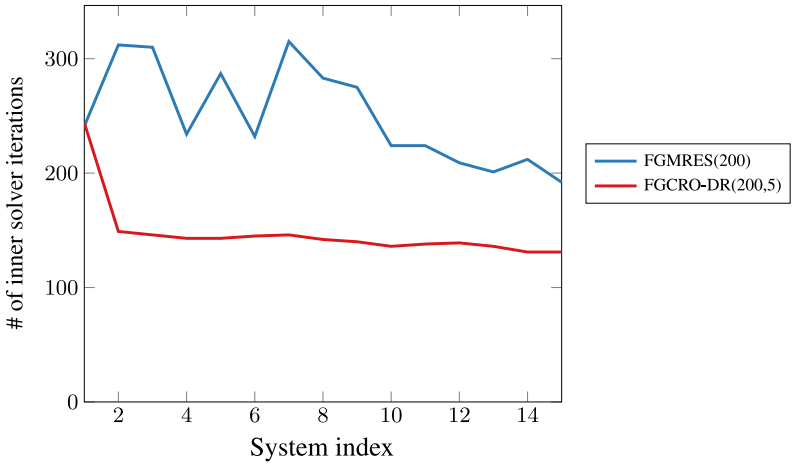


Fig. 10. Effect of a recycled Krylov method on the performance of the eigensolver. The number of mAL-preconditioned inner iterations for the flexible GMRES and GCRO-DR algorithms is compared for each linear solve of a Krylov–Schur iteration.

Table 3
Numerical performance of the 3D eigensolver with respect to the number of processes. The second and third columns represent respectively the number of Krylov–Schur iterations and the number of linear solves (outer iterations) per Krylov–Schur iteration. The fourth column is the number of mAL-preconditioned FGCRO-DR inner iterations per linear solve. The last three columns correspond to the average number of innermost ASM-preconditioned GMRES iterations for each velocity sub-block per inner iteration.

P	# of eigensolver iterations	# of linear solves	# of iterations per linear solve	# of iterations per field (x, y, z)		
512	7	7	120	7	10	11
1024	7	8	127	7	10	13
2048	7	8	119	10	13	17

In Fig. 10, the number of mAL-preconditioned inner linear iterations needed for each iterative method (FGMRES or FGCRO-DR) to solve the sequence of linear systems of the first iteration of the Krylov–Schur algorithm is reported. For this particular Krylov–Schur iteration, fifteen systems have to be solved. When using FGMRES, it corresponds to a total of 3.751 inner linear iterations. When using FGCRO-DR, it corresponds to only 2.209 inner linear iterations. Even though the solutions of all fifteen systems are not rigorously equal when switching from FGMRES to FGCRO-DR, after the first iteration of the eigensolver, convergence is reached for the two eigenpairs closest to the shift: $-1.03.10^{-2} + 0.57\mathbf{i}$ and $-7.81.10^{-2} + 0.57\mathbf{i}$.

In all the following runs, FGCRO-DR is used in order to reduce the number of inner iterations. The number of Krylov–Schur iterations needed to retrieve the requested eigenpairs is reported in the second column of Table 3. In the third column is the average number of solved linear systems per eigensolver iteration. The average number of mAL-preconditioned FGCRO-DR iterations (inner iterations) per linear solve (outer iteration) is presented in the fourth column whereas the last three columns contain the average number of innermost iterations for each velocity field, per application of $\mathbf{P}_{\text{mAL}}$. It was not possible to have the code run on 256 processes due to memory requirements significantly higher than for the nonlinear solver. Indeed, we switch from a real-valued to a complex-valued problem and the additional operators $\mathbf{M}$ and $\mathbf{L}_p$ are assembled explicitly. In Table 4, the scalability of our implementation is shown, using the run with 512 processes as the reference, and going up to 2048 processes. The parallel efficiency of this approach is approximately the same as for the nonlinear solver, though on a narrower range of process counts, remaining above 82%.

Table 4
Scalability of the 3D eigensolver with respect to the number of processes.

P	Setup (s)	Solve (s)	Speedup
512	55.3	39,160.7	–
1024	25.7	24,508.3	1.6
2048	27.1	11,849.9	3.3

6. Conclusion

The stationary base flow as well as the eigenvalue computations involved in hydrodynamic linear stability analysis require multiple solutions of linear systems based on the (shifted) Jacobian operator of the incompressible steady Navier–Stokes equations. To solve such systems on large-scale configurations involving hundreds of millions of unknowns, we proposed to use a Krylov subspace linear solver like the flexible GMRES algorithm, preconditioned by the modified augmented Lagrangian (mAL) preconditioner [45]. On a two-dimensional bluff-body flow, we studied numerically the performance of the mAL preconditioner for linear stability analysis purposes. We showed in particular that this approach handles efficiently complex-valued shifts and thus is well-suited for the computations of eigenvalues with possibly large frequencies, using the shift-and-invert spectral transformation. Then, the mAL preconditioner was tested against some other widely used steady-state (PCD, SIMPLE) and time-stepping-based (Stokes) preconditioners, all of them used in a sequential version. The mAL preconditioner was shown to require lower numbers of GMRES iterations and to be faster than all its competitors.

In order to perform large-scale three-dimensional stability analysis computations, a parallel implementation of the mAL preconditioner was presented and is made available online: https://github.com/prj-/moulin2019al. The FreeFEM finite element language was used as a discretization kernel whereas PETSc and SLEPc were used as distributed linear algebra backends. First, a comparison with the parallel direct solver MUMPS was presented on a three-dimensional bluff-body flow configuration, using a coarse enough mesh to make the LU factorization possible. The mAL approach required about one tenth as much memory and had better strong scaling properties. Despite the attractiveness of a direct linear solver – when it can be afforded – for the eigenvalue computations (the factorization is done once and re-used multiple times), the mAL approach turned out to be a faster alternative than the direct approach, thanks to its much better parallel performance. Finally, the implementation was used on a fine mesh resulting in 75 million unknowns and showed satisfying strong scaling properties up to 2048 processes, for both the base flow and eigenvalue computations. The role of subspace recycling between the multiple consecutive linear solves, inside the Krylov–Schur eigensolver, was tested and allowed significant performance gains.

Acknowledgments

This project has received funding from the European Research Council (ERC) under the European Union Horizon 2020 research and innovation program (grant agreement 638307). Moreover, this work was granted access to the HPC resources of TGCC@CEA under the allocation A0030607519 made by GENCI.

Appendix A. Reproducibility

In addition to the few extracts of the code used in the paper, the interested reader can find the complete FreeFEM code in the following repository: https://github.com/prj-/moulin2019al.

Appendix B. Definition of other block preconditioners

Here are defined the classical block preconditioners for incompressible Navier–Stokes that are compared to the modified augmented Lagrangian approach in Section 5.3. Contrary to mAL, those preconditioners do not require an augmentation. Thus, they are used without grad–div stabilization ($\gamma = 0$). Versions that incorporate a complex shift s are proposed here.

Pressure convection–diffusion preconditioner The pressure convection–diffusion (PCD) preconditioner was proposed by [41]:

$$\mathbf{P}_{\text{PCD}} = \begin{pmatrix} \mathbf{A} + s\mathbf{M_u} & \mathbf{B}^{\text{T}} \\ 0 & \mathbf{S}_p \end{pmatrix}, \tag{17}$$

Table 5
Effect of linear solver tolerance on eigenvalue computation ($\mathcal{R}e = 40$, $\gamma = 0.7$, $\varepsilon_{\text{eig}} = 10^{-6}$, $s = 0.7\mathbf{i}$).

ε_{lin}	# of GMRES iterations per linear solve	# of linear solves	λ	ω	$\left\|\sigma\mathbf{M}\hat{\mathbf{q}} + \mathbf{J}(\mathbf{q}_b)\hat{\mathbf{q}}\right\|_2$
10^{-1}	15	12	$4.48 \cdot 10^{-2}$	$6.90 \cdot 10^{-1}$	$1.5 \cdot 10^{-4}$
10^{-2}	28	10	$3.77 \cdot 10^{-2}$	$7.01 \cdot 10^{-1}$	$2.6 \cdot 10^{-6}$
10^{-3}	36	8	$3.75 \cdot 10^{-2}$	$7.01 \cdot 10^{-1}$	$9.5 \cdot 10^{-7}$
10^{-4}	45	8	$3.75 \cdot 10^{-2}$	$7.01 \cdot 10^{-1}$	$9.2 \cdot 10^{-7}$
10^{-5}	53	8	$3.75 \cdot 10^{-2}$	$7.01 \cdot 10^{-1}$	$9.2 \cdot 10^{-7}$
10^{-6}	62	8	$3.75 \cdot 10^{-2}$	$7.01 \cdot 10^{-1}$	$9.2 \cdot 10^{-7}$
10^{-7}	71	8	$3.75 \cdot 10^{-2}$	$7.01 \cdot 10^{-1}$	$9.2 \cdot 10^{-7}$
10^{-8}	80	8	$3.75 \cdot 10^{-2}$	$7.01 \cdot 10^{-1}$	$9.2 \cdot 10^{-7}$

with $\mathbf{S}_p{}^{-1} = -\mathbf{M}_p{}^{-1}(\mathbf{F}_p + s\mathbf{M}_p)\mathbf{L}_p{}^{-1}$, where $\mathbf{F}_p$ is a convection–diffusion operator built on the pressure space. Compared to the classical PCD preconditioner for steady-state Navier–Stokes equations, the shift contribution $s\mathbf{M}_p$ is added to the pressure Schur complement approximation.

SIMPLE preconditioner The SIMPLE preconditioner was proposed as a solver by [40]. We use its preconditioner version [73]:

$$\mathbf{P}_{\text{SIMPLE}} = \begin{pmatrix} \mathbf{A} + s\mathbf{M}_{\mathbf{u}} & 0 \\ \mathbf{B} & \mathbf{S}_p \end{pmatrix} \begin{pmatrix} \mathbf{I} & \text{diag}(\mathbf{A} + s\mathbf{M}_{\mathbf{u}})^{-1}\mathbf{B}^{\mathrm{T}} \\ 0 & \mathbf{I} \end{pmatrix}, \tag{18}$$

and $\mathbf{S}_p{}^{-1} = -\left[\mathbf{B}\text{diag}(\mathbf{A} + s\mathbf{M}_{\mathbf{u}})^{-1}\mathbf{B}^{\mathrm{T}}\right]^{-1}$.

Stokes preconditioner The Stokes preconditioning approach was popularized by [35] in the hydrodynamic stability community. Tuckerman's idea is two-fold:

(1) preconditioning the linearized Navier–Stokes problem Eq. (12) by the Stokes problem, i.e.,

$$\mathbf{P}_{\text{Stokes}} = \begin{pmatrix} \mathbf{A}_{\text{Stokes}} + s\mathbf{M}_{\mathbf{u}} & \mathbf{B}^{\mathrm{T}} \\ \mathbf{B} & 0 \end{pmatrix},$$

with $\mathbf{A}_{\text{Stokes}} = \mathbf{D} + \Delta t^{-1}\mathbf{M}_{\mathbf{u}}$ and $\mathbf{D}$ contains only the diffusion terms. Note that the time-step contribution Δt has no physical meaning here: it only represents some numerical parameter of the preconditioner. A case-dependent optimal value may exist, as reported in [39]. We determined this optimal value numerically.

(2) applying the preconditioner by adapting a pre-existing time-stepping code, which significantly reduces the development costs. In this work however, we prefer to apply $\mathbf{P}_{\text{Stokes}}{}^{-1}$ by using a few inner iterations of GMRES, preconditioned by:

$$\mathbf{P}_{\text{Stokes, inner}} = \begin{pmatrix} \mathbf{A}_{\text{Stokes}} + s\mathbf{M}_{\mathbf{u}} & \mathbf{B}^{\mathrm{T}} \\ 0 & \mathbf{S}_p \end{pmatrix},$$

with $\mathbf{S}_p{}^{-1} = -\mathcal{R}e^{-1}\mathbf{M}_p{}^{-1} - (\Delta t + s)\mathbf{L}_p{}^{-1}$ [74]. A large relative tolerance of 10^{-2} is set for the inner iterations, as we observed that further convergence of the inner iterations did not improve the convergence of the outer GMRES iterations. Obviously, $\mathbf{A}_{\text{Stokes}}$ being block diagonal, applying $\mathbf{P}_{\text{Stokes, inner}}{}^{-1}$ naturally requires two scalar velocity solves (in 2D) and one pressure solve.

Note that, in Section 5.3, the GMRES iteration count reported for the Stokes preconditioner corresponds to the total number of applications of $\mathbf{P}_{\text{Stokes, inner}}$ necessary to converge Eq. (12) to the desired tolerance.

Appendix C. Linear solver tolerance and eigenvalue convergence criterion

In cases where an iterative linear solver is used, the action of $(\mathbf{J} + s\mathbf{M})^{-1}$, required when applying the spectrally transformed operator $\mathbf{T}$ in the Krylov–Schur algorithm (see Section 2.4), is approximated using some user-defined tolerance. As a consequence, matrix $\mathbf{T}$ in Eq. (11) is replaced by some approximation $\widetilde{\mathbf{T}}$. It is usually recommended to set the tolerance for the linear solver lower than the one prescribed to the eigensolver, so that the imprecision of

the linear solver does not pollute the eigensolver accuracy (see e.g. [70, §3.4.1]). Here, we re-evaluate this statement numerically on the two-dimensional test case presented in Section 5.1.

The following numerical experiment is performed. We solve the eigenproblem (11) using a mAL-preconditioned GMRES algorithm to apply $(\mathbf{J} + s\mathbf{M})^{-1}$. The relative tolerance of the GMRES algorithm ε_{lin} is varied between 10^{-8} and 10^{-1} while the tolerance of the Krylov–Schur algorithm ε_{eig} is kept constant to 10^{-6}. Only one eigenvalue, closest to the shift $s = 0.7\mathbf{i}$, is demanded. In the second column of Table 5, the number of GMRES iterations required to apply $(\mathbf{J} + s\mathbf{M})^{-1}$ is shown. The value is averaged over all applications of $(\mathbf{J} + s\mathbf{M})^{-1}$ to compute the demanded eigenvalue. In the third column, the total number of applications of $\widetilde{\mathbf{T}}$ is shown. In the last three columns, we monitor the eigenvalue and the discrete l^2 norm of the eigenproblem residual. It is observed that one can in practice increase ε_{lin} well above $\varepsilon_{\text{eig}} = 10^{-6}$, without compromising significantly the accuracy of the computed eigenvalue. At least up to $\varepsilon_{\text{lin}} = 10^{-3}$, the computed eigenvalue is converged to satisfying accuracy, for a cost divided by two with respect to the "safe choice" $\varepsilon_{\text{lin}} = 10^{-6}$. Increasing ε_{lin} may thus allow some performance improvement.

Finally, the interpretation of the last column of Table 5 deserves some further explanation. Indeed, one can observe that, for $\varepsilon_{\text{lin}} = 10^{-1}$ and $\varepsilon_{\text{lin}} = 10^{-2}$, despite the fact that we kept $\varepsilon_{\text{eig}} = 10^{-6}$, the Krylov–Schur algorithm considered it had converged to an appropriate eigenvalue, while the residual was still above ε_{eig}. The reason to that observation is that most Arnoldi-based eigensolver packages, such as SLEPc, use convergence criteria based on the residual of the transformed problem Eq. (11), not the original one Eq. (6). As a consequence, the effect of using a large linear tolerance ε_{lin} is to apply an approximate operator $\widetilde{\mathbf{T}}$ far from the exact one. But the eigenvalues of $\widetilde{\mathbf{T}}$ can be computed to any prescribed tolerance by the Krylov–Schur algorithm. Note that, in SLEPc, a workaround is to use the option `-eps_true_residual` which forces the computation of the residual on the original problem Eq. (6) and thus is free from any approximation linked to the underlying linear solver. In this case, the effect of using a large linear tolerance ε_{lin} would be to make the convergence of the Krylov–Schur algorithm increasingly slow (or even impossible). This option being more costly, it should however be avoided for large-scale computations.

References

[1] P.G. Drazin, W.H. Reid, Hydrodynamic Stability, Cambridge University Press, 2004.

[2] P. Huerre, P.A. Monkewitz, Local and global instabilities in spatially developing flows, Annu. Rev. Fluid Mech. 22 (1) (1990) 473–537.

[3] V. Theofilis, Advances in global linear instability analysis of nonparallel and three-dimensional flows, Prog. Aerosp. Sci. 39 (4) (2003) 249–315.

[4] J.-M. Chomaz, Global instabilities in spatially developing flows: Non-normality and nonlinearity, Annu. Rev. Fluid Mech. 37 (1) (2005) 357–392.

[5] S. Bagheri, E. Åkervik, L. Brandt, D.S. Henningson, Matrix-free methods for the stability and control of boundary layers, AIAA J. 47 (5) (2009) 1057–1068.

[6] D. Sipp, O. Marquet, P. Meliga, A. Barbagallo, Dynamics and control of global instabilities in open-flows: a linearized approach, Appl. Mech. Rev. 63 (3) (2010).

[7] V. Theofilis, Global linear instability, Annu. Rev. Fluid Mech. 43 (1) (2011) 319–352.

[8] H.A. Dijkstra, F.W. Wubs, A.K. Cliffe, E. Doedel, I.F. Dragomirescu, B. Eckhardt, A.Y. Gelfgat, A.L. Hazel, V. Lucarini, A.G. Salinger, E.T. Phipps, S.U. Juan, H. Schuttelaars, L.S. Tuckerman, U. Thiele, Numerical bifurcation methods and their application to fluid dynamics: Analysis beyond simulation, Commun. Comput. Phys. 15 (1) (2014) 1–45.

[9] J.-C. Loiseau, M.A. Bucci, S. Cherubini, J.-C. Robinet, Time-stepping and Krylov methods for large-scale instability problems, in: Computational Modelling of Bifurcations and Instabilities in Fluid Dynamics, Springer International Publishing, 2019, pp. 33–73.

[10] G.M. Shroff, H.B. Keller, Stabilization of unstable procedures: the recursive projection method, SIAM J. Numer. Anal. 30 (4) (1993) 1099–1120.

[11] E. Åkervik, L. Brandt, D.S. Henningson, J. Hœpffner, O. Marxen, P. Schlatter, Steady solutions of the Navier–Stokes equations by selective frequency damping, Phys. Fluids 18 (6) (2006) 1–5.

[12] V. Citro, P. Luchini, F. Giannetti, F. Auteri, Efficient stabilization and acceleration of numerical simulation of fluid flows by residual recombination, J. Comput. Phys. 344 (April) (2017) 234–246.

[13] G.E. Karniadakis, M. Israeli, S.A. Orszag, High-order splitting methods for the incompressible Navier–Stokes equations, J. Comput. Phys. 97 (2) (1991) 414–443.

[14] J. Kim, P. Moin, Application of a fractional-step method to incompressible Navier–Stokes equations, J. Comput. Phys. 59 (2) (1985) 308–323.

[15] L.S. Tuckerman, D. Barkley, Bifurcation analysis for timesteppers, in: Numerical Methods for Bifurcation Problems and Large-Scale Dynamical Systems, Springer, New York, NY, 2000, pp. 453–466.

[16] M. Caliari, P. Kandolf, A. Ostermann, S. Rainer, Comparison of software for computing the action of the matrix exponential, BIT Numer. Math. 54 (1) (2014) 113–128.

[17] M.W. Rostami, F. Xue, Robust linear stability analysis and a new method for computing the action of the matrix exponential, SIAM J. Sci. Comput. 40 (5) (2018) A3344–A3370.
[18] P. Amestoy, I. Duff, J.-Y. L'Excellent, J. Koster, A fully asynchronous multifrontal solver using distributed dynamic scheduling, SIAM J. Matrix Anal. Appl. 23 (1) (2001) 15–41.
[19] X.S. Li, An overview of SuperLU: Algorithms, implementation, and user interface, ACM Trans. Math. Softw. 31 (3) (2005) 302–325.
[20] K.N. Christodoulou, L.E. Scriven, Finding leading modes of a viscous free surface flow: An asymmetric generalized eigenproblem, J. Sci. Comput. 3 (4) (1988) 355–406.
[21] U. Ehrenstein, F. Gallaire, On two-dimensional temporal modes in spatially evolving open flows: The flat-plate boundary layer, J. Fluid Mech. 536 (2005) 209–218.
[22] F. Sartor, C. Mettot, D. Sipp, Stability, receptivity, and sensitivity analyses of buffeting transonic flow over a profile, AIAA J. 53 (7) (2015) 1980–1993.
[23] K.A. Cliffe, T.J. Garratt, A. Spence, Eigenvalues of the discretized Navier–Stokes equation with application to the detection of hopf bifurcations, Adv. Comput. Math. 1 (3) (1993) 337–356.
[24] K. Meerbergen, D. Roose, Matrix transformations for computing rightmost eigenvalues of large sparse non-symmetric eigenvalue problems, IMA J. Numer. Anal. 16 (3) (1996) 297–346.
[25] C.J. Mack, P.J. Schmid, A preconditioned Krylov technique for global hydrodynamic stability analysis of large-scale compressible flows, J. Comput. Phys. 229 (3) (2010) 541–560.
[26] Y. Saad, Variations on Arnoldi's method for computing eigenelements of large unsymmetric matrices, Linear Algebra Appl. 34 (1980) 269–295.
[27] O. Marquet, M. Larsson, Global wake instabilities of low aspect-ratio flat-plates, Eur. J. Mech. B/Fluids 49 (2015) 400–412.
[28] S. Bagheri, P. Schlatter, P.J. Schmid, D.S. Henningson, Global stability of a jet in crossflow, J. Fluid Mech. 624 (2009) 33–44.
[29] J.-C. Loiseau, J.-C. Robinet, S. Cherubini, E. Leriche, Investigation of the roughness-induced transition: Global stability analyses and direct numerical simulations, J. Fluid Mech. 760 (2014) 175–211.
[30] V. Citro, F. Giannetti, P. Luchini, F. Auteri, Global stability and sensitivity analysis of boundary-layer flows past a hemispherical roughness element, Phys. Fluids 27 (8) (2015) 084110.
[31] L.S. Tuckerman, Steady-state solving via Stokes preconditioning; Recursion relations for elliptic operators, in: D.L. Dwoyer, M.Y. Hussaini, R.G. Voigt (Eds.), 11th International Conference on Numerical Methods in Fluid Dynamics, Springer Berlin Heidelberg, Berlin, Heidelberg, 1989, pp. 573–577.
[32] C.K. Mamun, L.S. Tuckerman, Asymmetry and hopf bifurcation in spherical couette flow, Phys. Fluids 7 (1) (1994) 80–91.
[33] D. Barkley, L.S. Tuckerman, Stokes Preconditioning for the inverse power method, in: P. Kutler, J. Flores, J.-J. Chattot (Eds.), Lecture Notes in Physics: Proceedings of the Fifteenth International Conference on Numerical Methods in Fluid Dynamics, Springer, New York, 1997, pp. 75–76.
[34] A. Bergeon, D. Henry, H. Benhadid, L.S. Tuckerman, Marangoni convection in binary mixtures with soret effect, J. Fluid Mech. 375 (1998) 143–177.
[35] L.S. Tuckerman, F. Bertagnolio, O. Daube, P. Le Quéré, D. Barkley, Stokes preconditioning for the inverse Arnoldi method, in: D. Henry, A. Bergeon (Eds.), Continuation Methods for Fluid Dynamics, Aussois, 2000, pp. 241–255.
[36] I. Mercader, O. Batiste, A. Alonso, Continuation of travelling-wave solutions of the Navier–Stokes equations, Internat. J. Numer. Methods Fluids 52 (7) (2006) 707–721.
[37] L.S. Tuckerman, Laplacian Preconditioning for the inverse Arnoldi method, Commun. Comput. Phys. 18 (05) (2015) 1336–1351.
[38] M. Brynjell-Rahkola, L.S. Tuckerman, P. Schlatter, D.S. Henningson, Computing optimal forcing using Laplace preconditioning, Commun. Comput. Phys. 22 (05) (2017) 1508–1532.
[39] C. Beaume, Adaptive Stokes preconditioning for steady incompressible flows, Commun. Comput. Phys. 22 (02) (2017) 494–516.
[40] S.V. Patankar, D.B. Spalding, A calculation procedure for heat, mass and momentum transfer in three-dimensional parabolic flows, Numer. Predict. Flow Heat Transf. Turbul. Combust. (1983) 54–73.
[41] D. Kay, D. Loghin, A. Wathen, A preconditioner for the steady-state Navier–Stokes equations, SIAM J. Sci. Comput. 24 (1) (2002) 237–256.
[42] M. Benzi, M.A. Olshanskii, An augmented Lagrangian-based approach to the oseen problem, SIAM J. Sci. Comput. 28 (6) (2006) 2095–2113.
[43] M. Benzi, M.A. Olshanskii, Field-of-values convergence analysis of augmented Lagrangian preconditioners for the linearized Navier–Stokes problem, SIAM J. Numer. Anal. 49 (2) (2011) 770–788.
[44] T. Heister, G. Rapin, Efficient augmented Lagrangian-type preconditioning for the Oseen problem using grad–div stabilization, Internat. J. Numer. Methods Fluids 71 (1) (2013) 118–134.
[45] M. Benzi, M.A. Olshanskii, Z. Wang, Modified augmented Lagrangian preconditioners for the incompressible Navier–Stokes equations, Internat. J. Numer. Methods Fluids 66 (4) (2011) 486–508.
[46] M. Benzi, Z. Wang, Analysis of augmented Lagrangian-based preconditioners for the steady incompressible Navier–Stokes equations, SIAM J. Sci. Comput. 33 (5) (2011) 2761–2784.
[47] M. Benzi, Z. Wang, A parallel implementation of the modified augmented Lagrangian preconditioner for the incompressible Navier–Stokes equations, Numer. Algorithms 64 (1) (2013) 73–84.
[48] A. Segal, M. Ur Rehman, C. Vuik, Preconditioners for incompressible Navier–Stokes solvers, Numer. Math. 3 (3) (2010) 245–275.
[49] X. He, C. Vuik, Comparison of some preconditioners for the incompressible Navier–Stokes equations, Numer. Math. 9 (2) (2016) 239–261.
[50] P.E. Farrell, L. Mitchell, F. Wechsung, An augmented Lagrangian preconditioner for the 3D stationary incompressible Navier–Stokes equations at high Reynolds number, 2018, arXiv:1810.03315.

[51] M.A. Olshanskii, M. Benzi, An augmented Lagrangian approach to linearized problems in hydrodynamic stability, SIAM J. Sci. Comput. 30 (3) (2008) 1459–1473.
[52] F. Hecht, New development in FreeFem++, J. Numer. Math. 20 (3–4) (2012) 251–266.
[53] S. Balay, S. Abhyankar, M.F. Adams, J. Brown, P. Brune, K. Buschelman, L. Dalcin, V. Eijkhout, W.D. Gropp, D. Kaushik, M.G. Knepley, L.C. McInnes, K. Rupp, B.F. Smith, S. Zampini, H. Zhang, PETSc web page, 2019, http://www.mcs.anl.gov/petsc.
[54] V. Hernandez, J.E. Roman, V. Vidal, SLEPC: A scalable and flexible toolkit for the solution of eigenvalue problems, ACM Trans. Math. Software 31 (3) (2005) 351–362.
[55] M. Olshanskii, G. Lube, T. Heister, J. Löwe, Grad–div stabilization and subgrid pressure models for the incompressible Navier–Stokes equations, Comput. Methods Appl. Mech. Engrg. 198 (49) (2009) 3975–3988.
[56] F. Brezzi, M. Fortin, Mixed and Hybrid Finite Element Methods, first ed., in: Springer Series in Computational Mathematics, vol. 15, Springer-Verlag, New York, 1991.
[57] M.A. Olshanskii, A low order Galerkin finite element method for the Navier–Stokes equations of steady incompressible flow: a stabilization issue and iterative methods, Comput. Methods Appl. Mech. Engrg. 191 (47) (2002) 5515–5536.
[58] M.A. Olshanskii, A. Reusken, Grad–div stabilization for Stokes equations, Math. Comp. 73 (248) (2004) 1699–1718.
[59] A. Linke, L.G. Rebholz, N.E. Wilson, On the convergence rate of grad–div stabilized Taylor–Hood to Scott–Vogelius solutions for incompressible flow problems, J. Math. Anal. Appl. 381 (2) (2011) 612–626.
[60] G.W. Stewart, A Krylov–Schur algorithm for large eigenproblems, SIAM J. Matrix Anal. Appl. 23 (3) (2002) 601–614.
[61] Y. Saad, A flexible inner-outer preconditioned GMRES algorithm, SIAM J. Sci. Comput. 14 (2) (1993) 461–469.
[62] P. Jolivet, V. Dolean, F. Hecht, F. Nataf, C. Prud'homme, N. Spillane, High performance domain decomposition methods on massively parallel architectures with FreeFem++, J. Numer. Math. 20 (3–4) (2012) 287–302.
[63] T.F. Chan, H.A. Van Der Vorst, Approximate and incomplete factorizations, in: Parallel Numerical Algorithms, Springer, 1997, pp. 167–202.
[64] M. Adams, H. Bayraktar, T. Keaveny, P. Papadopoulos, Ultrascalable implicit finite element analyses in solid mechanics with over a half a billion degrees of freedom, in: Proceedings of the 2004 ACM/IEEE Conference on Supercomputing, in: SC04, IEEE Computer Society, 2004, pp. 34:1–34:15.
[65] C. Geuzaine, J.-F. Remacle, Gmsh: A 3-D finite element mesh generator with built-in pre- and post-processing facilities, Internat. J. Numer. Methods Engrg. 79 (11) (2009) 1309–1331, http://geuz.org/gmsh.
[66] G. Karypis, V. Kumar, A parallel algorithm for multilevel graph partitioning and sparse matrix ordering, J. Parallel Distrib. Comput. 48 (1) (1998) 71–95.
[67] M. Benzi, D. Bertaccini, Block preconditioning of real-valued iterative algorithms for complex linear systems, IMA J. Numer. Anal. 28 (3) (2008) 598–618.
[68] L.S. Tuckerman, J. Langham, A. Willis, Order-of-magnitude speedup for steady states and traveling waves via Stokes preconditioning in channelflow and openpipeflow, Comput. Methods Appl. Sci. 50 (2019) 3–31.
[69] M.C. Iorio, L.M. González, E. Ferrer, Direct and adjoint global stability analysis of turbulent transonic flows over a NACA0012 profile, Internat. J. Numer. Methods Fluids 76 (3) (2014) 147–168.
[70] J.E. Roman, C. Campos, E. Romero, A. Tomás, SLEPc Users manual scalable library for eigenvalue problem computations, 2019, http://slepc.upv.es/documentation/slepc.pdf.
[71] V. Dolean, P. Jolivet, F. Nataf, An Introduction to Domain Decomposition Methods: Algorithms, Theory and Parallel Implementation, SIAM, 2015.
[72] P. Jolivet, P.-H. Tournier, Block iterative methods and recycling for improved scalability of linear solvers, in: Proceedings of the 2016 International Conference for High Performance Computing, Networking, Storage and Analysis, SC16, IEEE, 2016.
[73] H.C. Elman, V. Howle, J. Shahid, R. Shuttleworth, R.S. Tuminaro, A taxonomy and comparison of parallel block multi-level preconditioners for the incompressible Navier–Stokes equations, J. Comput. Phys. 227 (3) (2008) 1790–1808.
[74] J. Cahouet, J.-P. Chabard, Some fast 3D finite element solvers for the generalized Stokes problem, Internat. J. Numer. Methods Fluids 8 (1988) 869–895.