



# Fast Linear Solvers for Incompressible CFD Simulations with Compatible Discrete Operator Schemes

Yongseok Jang, Jerome Bonelle, Carola Kruse, Frank Hülsemann, Ulrich  
Ruede

## ► To cite this version:

Yongseok Jang, Jerome Bonelle, Carola Kruse, Frank Hülsemann, Ulrich Ruede. Fast Linear Solvers for Incompressible CFD Simulations with Compatible Discrete Operator Schemes. 2023. hal-04087358v2

**HAL Id: hal-04087358**

**<https://hal.science/hal-04087358v2>**

Preprint submitted on 13 May 2024

**HAL** is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

# Fast linear solvers for incompressible CFD simulations with compatible discrete operator schemes

Yongseok Jang<sup>1\*</sup>, Jérôme Bonelle<sup>2</sup>, Carola Kruse<sup>3</sup>, Frank Hülsemann<sup>4</sup>, Ulrich Ruede<sup>3,5</sup>

<sup>1</sup>ONERA, Université Paris-Saclay, Châtillon, France.

<sup>2</sup>EDF Lab Chatou, Chatou, France.

<sup>3</sup>Cerfacs, Toulouse, France.

<sup>4</sup>EDF Lab Paris-Saclay, Palaiseau, France.

<sup>5</sup>Department of Computer Science, FAU Erlangen-Nürnberg, Erlangen, Germany.

\*Corresponding author(s). E-mail(s): [yongseok.jang@onera.fr](mailto:yongseok.jang@onera.fr);

Contributing authors: [jerome.bonelle@edf.fr](mailto:jerome.bonelle@edf.fr); [carola.kruse@cerfacs.fr](mailto:carola.kruse@cerfacs.fr);  
[frank.hulsemann@edf.fr](mailto:frank.hulsemann@edf.fr); [ulrich.ruede@fau.de](mailto:ulrich.ruede@fau.de);

## Abstract

Finding a robust and efficient solver for (non-)symmetric systems that arise in incompressible Computational Fluid Dynamics (CFD) is of great interest to both academia and industry. We consider the Compatible Discrete Operator (CDO) discretization that has recently been devised for CFD simulations in the context of incompressible Stokes and Navier–Stokes flows. The discrete problems resulting from CDO schemes yield large saddle-point systems that require relevant numerical methods suitable to deal with large indefinite and poorly conditioned linear systems. In this paper, we focus on two segregated methods: the augmented Lagrangian Uzawa method and the generalized Golub-Kahan bidiagonalization, as well as a monolithic method based on an algebraic transformation by change of variables. We also employ algebraic multigrid (AMG) preconditioned Krylov solvers such as the Flexible Conjugate Gradient (FCG) method, and the Flexible Generalized Minimal Residual (FGMRES) method, to solve the linear systems. Using the CFD software code `saturne`, we compare the numerical performance with respect to the choice of linear solvers and numerical strategies for the saddle-point problem. In the numerical experiments, the AMG preconditioned Krylov methods show robustness in test cases of Stokes and Navier–Stokes problems.

**Keywords:** Algebraic multigrid method, Compatible discrete operator, Incompressible Navier–Stokes, Saddle-point problems

**MSC Classification:** 65F08 , 65F10 , 65M22 , 76D05

# 1 Introduction

Problems in Computational Fluid Dynamics (CFD) arise in many academic and industrial fields, e.g., aerospace, petroleum and nuclear engineering. In this paper, we focus on the steady Stokes and Navier–Stokes equations in the case of incompressible flows. Let  $\Omega \subset \mathbb{R}^d, d = 2, 3$  be an open bounded connected polytopal domain and  $\partial\Omega$  be its boundary. The velocity is a vector-valued field denoted by  $\vec{u}$  and the pressure is a scalar-valued field denoted by  $p$  such that:

$$-\nu\Delta\vec{u} + \chi((\vec{u} \cdot \nabla)\vec{u}) + \nabla p = \vec{f} \quad \text{in } \Omega, \quad (1.1)$$

$$\nabla \cdot \vec{u} = 0 \quad \text{in } \Omega. \quad (1.2)$$

Equation (1.1) refers to the conservation of the momentum and (1.2) to the incompressibility constraint, ensuring that the velocity field conserves mass. Here we assume a constant mass density. The parameter  $\nu > 0$  denotes the fluid viscosity and  $\vec{f}$  is the volumetric forcing term.  $-\nu\Delta\vec{u}$  is the viscous term and  $(\vec{u} \cdot \nabla)\vec{u}$  is the convection term. The choice  $\chi = 0$  corresponds to the Stokes equations whereas the choice  $\chi = 1$  to the Navier–Stokes equations. Dirichlet boundary conditions are enforced on  $\partial\Omega$ . The pressure is uniquely defined by enforcing  $\int_{\Omega} p = 0$ . To manage the nonlinearity of the Navier–Stokes equation in the convection term, we introduce *Picard's iteration* yielding the following linearized Navier–Stokes equation (also known as *Oseen problem*) such that

$$-\nu\Delta\vec{u}^{(k)} + (\vec{u}^{(k-1)} \cdot \nabla)\vec{u}^{(k)} + \nabla p^{(k)} = \vec{f} \quad \text{in } \Omega, \quad (1.3)$$

$$\nabla \cdot \vec{u}^{(k)} = 0 \quad \text{in } \Omega, \quad (1.4)$$

for each iteration  $k$ , starting from an arbitrary initial guess  $(\vec{u}^{(0)}, p^{(0)})$ .

The discretization of equations (1.1) and (1.2) has been extensively studied in the literature. Depending on the choice of the velocity-pressure coupling (segregated or coupled), the definition and location of the degrees of freedom (DoF), the resulting linear system(s) can have quite different structure and different features. In what follows, we consider the Compatible Discrete Operator (CDO) schemes introduced in [1] for the spatial discretization. CDO schemes belong to a class of space discretization schemes called *mimetic*, *structure-preserving* or *compatible*. These schemes have been inspired by the seminal works of Bossavit [2] and of Hyman & Scovel [3]. They have shed a new light on the way to devise the discretization of partial differential equations (PDE) thanks to some concepts of differential geometry and algebraic topology. During the last two decades, several other discretization schemes belonging to this latter class have emerged, e.g., the Discrete Exterior Calculus (DEC) schemes [4], the Discrete Geometric Approach (DGA) [5], the Mimetic Spectral Element method [6], the Mimetic Finite Difference schemes [7], the Hybrid Mixed Mimetic (HHM) framework [8], the Finite Element Exterior Calculus (FEEC) schemes [9] or for De Rham complexes [10]. More recently, extensions to higher order discretizations have been devised in Virtual Element Methods (VEM) [11], Hybrid High Order (HHO) schemes [12] or the Discrete De Rham framework [13].

In this paper, we focus on CDO face-based schemes with a full velocity/pressure coupling. As explained in [14], this choice leads to a low-order approximation of the Navier–Stokes equations. The stability and the approximation properties are fulfilled on a wide range of meshes: from Cartesian meshes to polyhedral, non-matching and/or distorted meshes. The CDO face-based discretization is a stable method (cf. Section 2). The resulting linear system is a saddle-point problem formulated as  $\mathcal{A}\mathbf{x} = \mathbf{b}$ , where

$$\mathcal{A} = \begin{bmatrix} A & B^T \\ B & O \end{bmatrix}, \quad \mathbf{x} = \begin{bmatrix} \mathbf{u} \\ \mathbf{p} \end{bmatrix} \quad \text{and} \quad \mathbf{b} = \begin{bmatrix} \mathbf{f} \\ \mathbf{g} \end{bmatrix}. \quad (1.5)$$

Here,  $A \in \mathbb{R}^{n \times n}$ ,  $B \in \mathbb{R}^{m \times n}$ ,  $O \in \mathbb{R}^{m \times m}$  is the null matrix,  $\mathbf{u} \in \mathbb{R}^n$ ,  $\mathbf{f} \in \mathbb{R}^n$ ,  $\mathbf{p} \in \mathbb{R}^m$  and  $\mathbf{g} \in \mathbb{R}^m$  for  $n \geq m$ . The blocks  $A$ ,  $B^T$ ,  $B$  and  $O$  are commonly referred to as the (1,1)-, (1,2)-, (2,1)- and (2,2)-blocks, respectively. In the case of the Stokes equations,  $A$  is a symmetric positive definite (SPD) matrix corresponding to the viscous (diffusion) term, while  $B$  is associated with the divergence operator for the velocity field and  $B^T$  with the pressure gradient operator. In the case of the (linearized) Navier–Stokes equations,  $A$  is a non-symmetric matrix due to the addition of a convective term to the viscous term.

Different types of saddle-point problems occur in many applications of applied mathematics and engineering. To solve the corresponding linear systems, numerous techniques have been proposed and developed during the previous decades; please refer to the comprehensive introduction of Benzi, Golub and Liesen [15] on the resolution of saddle-point problems. In what follows, one splits these techniques to solve (1.5) into two categories: *segregated* techniques working iteratively on a subset of blocks and *monolithic* techniques working on the full system (all blocks at once). For these two categories, linear systems have to be solved either with a *direct* method or an iterative method.

Direct methods are robust with respect to the properties of the linear system (symmetry and conditioning for instance) but their computational cost in terms of CPU and mainly memory usage can become prohibitive on large scale systems. In this work, we consider parallel multi-frontal algorithms available in MUMPS [16] as direct solver.

Krylov subspace methods [17] are our choice of iterative methods: for instance, GMRES [18] on non-symmetric systems and CG [19] on symmetric positive definite systems. Krylov subspace methods can become very competitive solvers when combined with efficient preconditioning techniques. To employ variable preconditioning, i.e., the preconditioner can be modify at each iteration (e.g. inner-outer Krylov methods), flexible variants of those methods have been developed such as FGMRES [20], GCR [21, 22] or FCG [23]. For more details for non-symmetric problem solvers, we refer to [24, 25] and the references therein. The efficiency of the iterative solver relies mainly on the choice of the preconditioner. In this paper, one considers multilevel preconditioning technique based on algebraic multigrid (AMG) methods. These methods have been proven to work efficiently on large scale linear systems stemming from unstructured grids without needing geometric grid information; see [26] for a comparison of multigrid preconditioners. The main ingredients composing the design of AMG

methods are the choice of the cycle (V-cycle for instance), that of the smoother (a symmetric Gauss-Seidel for instance) and the coarsening algorithm. There are two main classes of coarsening strategies: independent-set based AMG, such as *classical AMG* [27–30] and aggregation-based AMG, such as *smoothed aggregation AMG* [31] and *pairwise aggregation AMG* [22, 32]. Please refer to [33] for more references and details about AMG methods. More recently, Notay [34] then Bacq and Notay [35] investigated aggregation-based AMG for the Stokes problems and (linearized) Navier–Stokes problems, respectively, including a two-grid analysis.

In the manner of segregated techniques, one option is to utilize stationary iterations and Uzawa-like algorithms [15, 36], including the variant known as the *Augmented-Lagrangian Uzawa* (ALU) algorithm [37]. Another approach involves the *Golub–Kahan Bidiagonalization* (GKB) algorithm [38], which has recently emerged as a method for solving symmetric saddle-point systems.

Regarding monolithic techniques, apart from direct methods applied to the full system, other techniques vary in their approach to defining an efficient preconditioner for a Krylov subspace method. *ILU* type preconditioning or null space methods [15] are possible choices. An approximated *block LU* factorization has also been introduced in [39]. More generally, block preconditioning are among the most explored techniques of this category. In this case, the approximation of the Schur complement [40] (in our case, equal to  $-BA^{-1}B^T$ ) is often required. In this situation, the quality of this approximation is a key ingredient to get an efficient solver. More specifically in the context of the incompressible Navier–Stokes problems *Semi-Implicit Pressure Linked Equation* (SIMPLE) type preconditioning [41] have been investigated. Other approaches, such as augmented Lagrangian preconditioning, which is applicable for high Reynolds number [42, 43] cases, and a vector penalty projection method [44] are other techniques of interest. Additionally, an algebraic transformation of the saddle-point system relying on a change of variables has been devised in [34] in the context of the Stokes problem. This transformation allows one to use multigrid methods on the (monolithic) transformed system.

Our main contribution is to present a first comparative study of algorithms used to solve saddle-point systems that arise from the CDO face-based discretization of the Stokes and Navier–Stokes equations. This discretization differs from the standard Finite Elements schemes since one performs a *static condensation* of the (1,1)-block (cf. Section 2). We focus on three algorithms: two segregated techniques, the *Augmented-Lagrangian Uzawa* (ALU) algorithm [37] and the *Golub–Kahan Bidiagonalization* (GKB) algorithm [38] and, one monolithic technique, the Notay’s algebraic transformation [34]. To our knowledge, this is the first application of the CDO scheme with the GKB algorithm to solve the Stokes problem, as well as the first use of the algebraic transformation for both the Stokes and Navier–Stokes problems. Additionally, several Krylov solvers preconditioned with an AMG are compared either on the resolution of the (1,1)-block or on the resolution of the transformed saddle-point problem. These comparative studies are performed on the key ingredients that underpin the definition of an AMG, such as the type of cycle, the type of smoothers, and the coarsening strategy. Specifically, we consider an in-house implementation of the K-cycle algorithm

based on Notay’s work [22, 45], BoomerAMG from the HYPRE library [46, 47], and GAMG/HMG from the PETSc library [48].

The remainder of the paper is structured as follows. After the introduction, we describe the main features and ingredients of the CDO face-based discretization in Section 2, along with the resulting saddle-point problem. In Sections 3 and 4, we provide detailed explanations of the different algorithms used in our comparative studies, as well as the various strategies and configurations of the multilevel algorithms used as a preconditioner for a Krylov solver. Numerical experiments to compare their numerical performance are reported in Section 5. At the end, conclusion and prospect for future improvements are presented in Section 6. We note that all the algorithms and discretizations described in this paper are freely available in the latest version of `code_saturne`<sup>1</sup>.

## 2 The Compatible Discrete Operator (CDO) framework

The discretization of the Stokes and Navier–Stokes equations under consideration rely on the CDO framework, as developed during Bonelle’s PhD [49, 50]. It encompasses several discretizations according to the location of the degrees of freedom (DoF): vertex-based, edge-based, face-based and cell-based schemes.

In this work, we focus on the CDO face-based schemes for the discretization of the steady Stokes and Navier–Stokes equations. The velocity DoFs are hybrid in the sense that they are located in the cells and at the faces (see Figure 1; Left part). They are defined as the mean-value of the velocity vector-field in a cell and on a face for each component. The main ingredients underpinning the discretization (the velocity gradient reconstruction operator and the velocity divergence operator) are recalled and we refer to [14, 51] for more details. The resulting CDO face-based scheme is analyzed in [51] and references therein (stability, consistency, *a priori* error estimates, etc.). In particular, the stability results imply that  $\mathcal{A}$ , defined in (1.5), is invertible. Namely, the solvability conditions [15, Theorem 3.4]

$$\ker(H) \cap \ker(B) = \{\mathbf{0}\} \quad \text{and} \quad \ker(B^T) = \{\mathbf{0}\}, \quad (2.1)$$

where  $H$  is the symmetric part of  $A$  are fulfilled since  $A$  is non-singular and  $B$  is a full rank operator (*LBB inf-sup* condition).

### Notation

Let  $C$  (resp.  $F$ ) be the set of the mesh cells (resp. mesh faces). The number of elements in a set  $X$  is denoted by  $\#X$  (e.g.  $\#C$  is the number of mesh cells). The set of faces associated to a cell  $c \in C$  is denoted by  $F_c$ . The set of faces is split into two subsets: the set of boundary faces  $F^\partial := \{f \in F \mid f \subset \partial\Omega\}$  and the set of the interior faces  $F^\circ := F \setminus F^\partial$ . For a face  $f$ ,  $|f|$  corresponds to its surface and  $\vec{x}_f$  to its barycenter. One chooses an arbitrary orientation  $\vec{n}_f$  to each face and one denotes  $\vec{n}_{f,c}$  its outward

---

<sup>1</sup><https://www.code-saturne.org>

normal such that  $\vec{n}_{f,c} = \iota_{f,c} \vec{n}_f$  with  $\iota_{f,c} = \pm 1$  according to the arbitrary choice. For a cell  $c$ ,  $|c|$  and  $\vec{x}_c$  denote its volume and barycenter, respectively.

The vector-valued piecewise constant polynomial space in a cell  $c \in C$  (resp. on a face  $f \in F$ ) is denoted by  $\mathbb{P}_0^d(c)$  (resp.  $\mathbb{P}_0^d(f)$ ) with  $d$  the dimension of  $\Omega$  (here  $d = 3$ ). The scalar-valued piecewise constant polynomial space in a cell  $c \in C$  is denoted by  $\mathbb{P}_0(c)$ . The local space of velocity DoFs (i.e. associated to a cell  $c \in C$ ) is denoted by  $\mathcal{U}(c)$  and the local space of pressure DoFs by  $\mathcal{P}(c)$  such that

$$\mathcal{U}(c) := \left( \bigotimes_{f \in F_c} \mathbb{P}_0^d(f) \right) \times \mathbb{P}_0^d(c) \quad \text{and} \quad \mathcal{P}(c) := \mathbb{P}_0(c).$$

For  $c \in C$ , an element  $\mathbf{u}_c \in \mathcal{U}(c)$  is such that  $\mathbf{u}_c := ((\vec{u}_f)_{f \in F_c}, \vec{u}_c) \in \mathbb{R}^{d(\#F_c+1)}$  and  $p_c \in \mathcal{P}(c)$  is simply a constant value inside the cell  $c$ .  $\mathcal{U}(\Omega)$  denotes the global space of velocity DoFs and  $\mathcal{P}(\Omega)$  the global space of pressure DoFs. These spaces are defined by

$$\mathcal{U}(\Omega) := \left( \bigotimes_{f \in F} \mathbb{P}_0^d(f) \right) \times \left( \bigotimes_{c \in C} \mathbb{P}_0^d(c) \right) \quad \text{and} \quad \mathcal{P}(\Omega) := \bigotimes_{c \in C} \mathbb{P}_0(c).$$

An element  $\mathbf{u} \in \mathcal{U}(\Omega)$  is such that  $\mathbf{u} := ((\vec{u}_f)_{f \in F}, (\vec{u}_c)_{c \in C}) \in \mathbb{R}^{d(\#F+\#C)}$ . An element  $p \in \mathcal{P}(\Omega)$  is such that  $p := (p_c)_{c \in C} \in \mathbb{R}^{\#C}$ . The space of pressure DoFs is denoted by  $\mathcal{P}^*(\Omega)$  when a zero mean-value is prescribed on the pressure DoFs. The space of velocity DoFs is denoted by  $\mathcal{U}^\circ(\Omega)$  when homogeneous Dirichlet boundary conditions are enforced on  $\partial\Omega$ . For the sake of clarity, we only focus on this kind of boundary conditions. These two spaces are defined as follows:

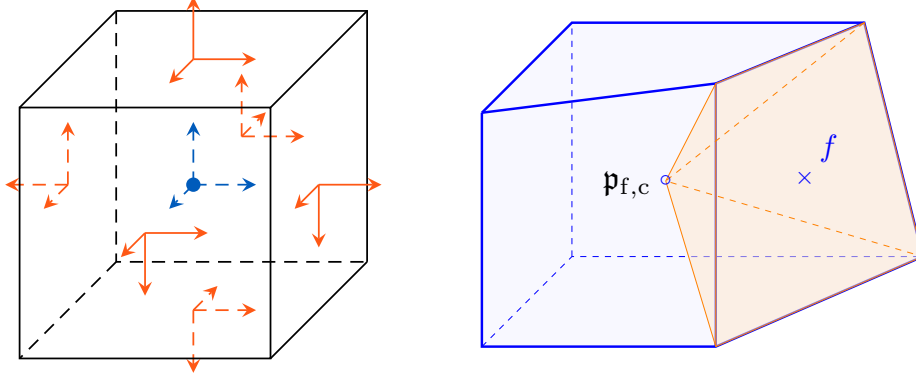
$$\mathcal{P}^*(\Omega) := \{p \in \mathcal{P}(\Omega) \mid \sum_{c \in C} p_c |c| = 0\} \quad \text{and} \quad \mathcal{U}^\circ(\Omega) = \{\mathbf{u} \in \mathcal{U}(\Omega) \mid \vec{u}_f = 0 \, \forall f \in F^\partial\}.$$

### Main operators

The two main operators used to discretize the Stokes problem (1.1)- (1.2) are the velocity gradient reconstruction operator and the velocity divergence operator. The pressure gradient is simply obtained as the adjoint operator of the velocity divergence. For a cell  $c \in C$ , the local reconstruction operator for the velocity gradient is denoted by  $\mathbf{G}_c$  and is defined by a piecewise constant tensor in each pyramid  $\mathbf{p}_{f,c}$  associated to a face  $f \in F_c$  (see Figure 1). Let  $\mathfrak{P}_{F_c} := \{\mathbf{p}_{f,c}\}_{f \in F_c}$  be the set of all pyramids in the cell  $c$ .  $\mathbf{G}_c : \mathcal{U}(c) \mapsto \mathbb{P}_0^{d \times d}(\mathfrak{P}_{F_c})$  such that

$$\mathbf{G}_c(\mathbf{u}_c)|_{\mathbf{p}_{f,c}} = \mathbf{G}_{0,c}(\mathbf{u}_c) + \eta \frac{|f|}{|\mathbf{p}_{f,c}|} ((\vec{u}_f - \vec{u}_c) - \mathbf{G}_{0,c}(\mathbf{u}_c)(\vec{x}_f - \vec{x}_c)) \otimes \vec{n}_{f,c}$$

where  $\eta > 0$  is a scaling coefficient related to the stability of the reconstruction. Depending on the value of this coefficient, one can recover the Generalization of the Crouzeix–Raviart framework [52] with  $\eta = 1$ , the Discrete Geometric Approach [5] with  $\eta = 1/d$  or the Hybrid Finite Volume method [8] with  $\eta = 1/\sqrt{d}$ .



**Fig. 1** Left: Example in a hexahedral cell of the locations of the velocity DoFs (three components on each face in orange and three components in the cell in blue), and the pressure DoF (the mean-value in the cell – the blue bullet). Right: Example of a pyramid of basis the face  $f$ . The velocity gradient is constant inside this volume.

$G_{0,c}(\mathbf{u}_c)$  is a piecewise constant tensor in the cell  $c$  corresponding to the  $\mathbb{P}_0$ -consistent gradient reconstruction defined as

$$G_{0,c} : \mathcal{U}(c) \mapsto \mathbb{P}_0^{d \times d}(c) \quad \text{s.t.} \quad G_{0,c}(\mathbf{u}_c) := \frac{1}{|c|} \sum_{f \in F_c} |f| (\vec{u}_f - \vec{u}_c) \otimes \vec{n}_{f,c}. \quad (2.2)$$

The global velocity gradient reconstruction operator is simply defined by collecting the local velocity gradient reconstruction operators. The definition of the velocity divergence relies on the identity  $\nabla \cdot (\vec{u}) = \text{trace}(\nabla \vec{u})$ . For each cell  $c \in C$ , the local velocity divergence operator  $D_c$  is defined as

$$D_c : \mathcal{U}(c) \mapsto \mathcal{P}(c) \quad \text{s.t.} \quad D_c(\mathbf{u}_c) := \frac{1}{|c|} \sum_{f \in F_c} |f| \vec{u}_f \cdot \vec{n}_{f,c}. \quad (2.3)$$

In the case of the Navier–Stokes equations, one also introduces a convection operator, see [51] for more details.

#### **Weak formulation of the discrete Stokes problem**

The discrete weak formulation for the Stokes problem stated in (1.1)–(1.2) with  $\chi = 0$  relies on the two previous operators. With homogeneous Dirichlet boundary conditions on the velocity field, this yields: find  $(\mathbf{u}, p) \in \mathcal{U}^\circ(\Omega) \times \mathcal{P}^*(\Omega)$  such that  $\forall \mathbf{w} \in \mathcal{U}^\circ(\Omega)$  and  $\forall q \in \mathcal{P}^*(\Omega)$ ,

$$\sum_{c \in C} \int_c \nu G_c(\mathbf{u}_c) : G_c(\mathbf{w}_c) - \sum_{c \in C} \int_c D_c(\mathbf{w}_c) p_c = \sum_{c \in C} \int_c \vec{f} \cdot \vec{w}_c \quad (2.4)$$

$$- \sum_{c \in C} \int_c D_c(\mathbf{u}_c) q_c = 0. \quad (2.5)$$



### Algebraic viewpoint

Switching to the algebraic viewpoint, the different summands in equations (2.4)–(2.5) correspond to the different blocks of the local saddle-point systems  $\hat{\mathcal{A}}_c \hat{\mathbf{x}}_c = \hat{\mathbf{b}}_c$  associated to each cell  $c \in C$  with  $\hat{\mathbf{x}}_c$  and  $\hat{\mathbf{b}}_c$ , two arrays restricted to all the face and cell DoFs associated to a cell. Elements which will be modified by the static condensation are written with a hat symbol. More specifically, the system corresponds to

$$\hat{\mathcal{A}}_c := \left[ \begin{array}{c|c} \hat{A}(c) & B^T(c) \\ \hline B(c) & 0 \end{array} \right] \quad \text{and} \quad \hat{\mathbf{b}}_c := \left[ \begin{array}{c} \hat{\mathbf{f}}(c) \\ \hline \mathbf{g}(c) \end{array} \right] \quad (2.6)$$

where  $\hat{A}(c)$  is a square matrix of size  $d(\#F_c + 1)$  associated to  $\int_c \nu \mathbf{G}_c(\mathbf{u}_c) : \mathbf{G}_c(\mathbf{w}_c)$  and  $B(c)$  is a rectangular matrix of size  $d \times d(\#F_c + 1)$  associated to  $\int_c \mathbf{D}_c(\mathbf{u}_c) \mathbf{q}_c$ . The static condensation modifies the velocity block  $\hat{A}(c)$  and its right-hand side  $\hat{\mathbf{f}}(c)$ . To detail this operation, one splits the system (2.6) to make appear the contribution stemming from the face and cell velocity DoFs.

$$\hat{\mathcal{A}}_c := \left[ \begin{array}{c|c|c} \hat{A}_{FF}(c) & A_{FC}(c) & B_F^T(c) \\ \hline A_{CF}(c) & A_{CC}(c) & 0 \\ \hline B_F(c) & 0 & 0 \end{array} \right] \quad \text{and} \quad \hat{\mathbf{b}}_c := \left[ \begin{array}{c} \hat{\mathbf{f}}_F(c) \\ \hline \hat{\mathbf{f}}_C(c) \\ \hline \mathbf{g}(c) \end{array} \right] \quad (2.7)$$

Since the block  $A_{CC}(c)$  is a diagonal square matrix of size  $d$ , the static condensation technique [53] allows one to reduce easily the size of the local system by removing the cell DoFs related to the velocity as follows:

$$\mathcal{A}_c := \left[ \begin{array}{c|c} A_c & B_F^T(c) \\ \hline B_F(c) & 0 \end{array} \right] \quad \text{and} \quad \mathbf{b}_c := \left[ \begin{array}{c} \mathbf{f}(c) \\ \hline \mathbf{g}(c) \end{array} \right] \quad (2.8)$$

where the velocity block  $A_c := \hat{A}_{FF}(c) - A_{FC}(c) \cdot A_{CC}^{-1}(c) \cdot A_{CF}(c)$  is a square matrix of size  $d\#F_c$  and  $\mathbf{f}(c) := \hat{\mathbf{f}}_F(c) - A_{FC}(c) \cdot A_{CC}^{-1}(c) \hat{\mathbf{f}}_C(c)$ . The resulting system (1.5) with  $n = d\#F$  and  $m = \#C$  stems from the cell-wise assembly process of the local systems detailed in (2.8).

## 3 Some strategies to solve saddle-point problems

Here we introduce three algorithms for solving saddle-point problems obtained in stable discretizations of the Stokes or Navier–Stokes equations. In general, a direct solver is a robust and precise option to solve the system (1.5), especially when the (1,1)-block is non-symmetric and ill-conditioned. However, when the systems become too large or less sparse as in typical 3D problems, then a direct solver is no longer applicable. For the special case of the Stokes problem, i.e.  $\chi = 0$  in (1.1)–(1.2), the system (1.5) is symmetric. We therefore first introduce the Craig’s variant of the Golub–Kahan bidiagonalization (GKB) for symmetric saddle-point systems. It shows a good performance in terms of iteration count and solution time, however, it is only applicable for symmetric matrices and thus its usage in `code_saturne` is limited.

As explained earlier, we solve the Navier–Stokes problem with Picard (fixed-point) iterations, where each iteration corresponds to an Oseen problem (1.3)–(1.4). The saddle-point matrix is non-symmetric due to the presence of the advective field defined as  $\vec{u}^{(k-1)}$ . The second algorithm that we introduce is the Augmented Lagrangian Uzawa (ALU) method suited for symmetric or non-symmetric indefinite matrices. In the practical cases of this paper and with some well-chosen parameter, it solves the problem in an acceptably small number of iterations. As a third alternative, we use an algorithm that we call *Notay’s algebraic transformation*. It can be applied to symmetric or non-symmetric problems and the obtained transformed matrices can be solved with an efficient AMG solver or with a Krylov subspace method preconditioned with an AMG.

### 3.1 Golub-Kahan bidiagonalization

We briefly summarize the generalized GKB method that has been introduced in [38]. The algorithm solves symmetric saddle-point systems, which implies that the GKB method can only be applied to the Stokes equations in our context. As a first step, we need to transform the saddle-point system to obtain a zero vector in the upper part of the right-hand side vector. We furthermore use a common regularization technique [54], known as the augmented Lagrangian approach. Let therefore  $\gamma \geq 0$  be a scaling factor and  $W \in \mathbb{R}^{m \times m}$  be an SPD matrix. By recalling the solvability condition (2.1), we can replace (1.5) by the equivalent augmented system  $\mathcal{A}_\gamma \mathbf{x}_\gamma^* = \mathbf{b}_\gamma^*$  with

$$\mathcal{A}_\gamma = \begin{bmatrix} A_\gamma & B^T \\ B & O \end{bmatrix}, \quad \mathbf{x}_\gamma^* = \begin{bmatrix} \mathbf{u}_\gamma^* \\ \mathbf{p} \end{bmatrix}, \quad \mathbf{b}_\gamma^* = \begin{bmatrix} 0 \\ \mathbf{g}_\gamma^* \end{bmatrix}, \quad (3.1)$$

where  $A_\gamma = A + \gamma B^T W^{-1} B$ ,  $\mathbf{u}_\gamma^* = \mathbf{u} - A_\gamma^{-1}(\mathbf{f} + \gamma B^T W^{-1} \mathbf{g})$  and  $\mathbf{g}_\gamma^* = \mathbf{g} - B A_\gamma^{-1} \mathbf{f}$ . Then the GKB method follows as: Find  $V \in \mathbb{R}^{n \times n}$ ,  $Q \in \mathbb{R}^{m \times m}$  and  $D \in \mathbb{R}^{n \times m}$  such that

$$\begin{cases} B^T Q = A_\gamma V \begin{bmatrix} D \\ O \end{bmatrix}, & V^T A_\gamma V = I_n, \\ BV = WQ \begin{bmatrix} D^T & O \end{bmatrix}, & Q^T W Q = I_m, \end{cases} \quad (3.2)$$

with  $D$  being an upper bidiagonal matrix

$$D = \begin{bmatrix} \alpha_1 & \beta_1 & 0 & \cdots & 0 \\ 0 & \alpha_2 & \beta_2 & \cdots & 0 \\ \vdots & \ddots & \ddots & \ddots & \vdots \\ 0 & \cdots & 0 & \alpha_{n-1} & \beta_{n-1} \\ 0 & \cdots & 0 & 0 & \alpha_n \end{bmatrix},$$

with entries  $\alpha_i$  and  $\beta_i$  computed successively as given in Algorithm 1. By multiplying the augmented system  $\mathcal{A}_\gamma \mathbf{x}_\gamma^* = \mathbf{b}_\gamma^*$  with the block-diagonal matrix with diagonal elements  $V^T$  and  $Q^T$  from the left, and using the change of variables  $\mathbf{u}_\gamma^* = V\mathbf{y}$  and  $\mathbf{p} = Q\mathbf{z}$ , we obtain with (3.2)

$$\begin{bmatrix} I_n & D \\ D^T & O \end{bmatrix} \begin{bmatrix} \mathbf{y} \\ \mathbf{z} \end{bmatrix} = \begin{bmatrix} \mathbf{0} \\ Q^T \mathbf{g}_\gamma^* \end{bmatrix}. \quad (3.3)$$

Hence, once we derive  $V$  and  $Q$ , we can also get the solution of the augmented system by solving (3.3). Craig's variant algorithm [38, Algorithm 3.1] used to solve the augmented system is presented in Algorithm 1. In each iteration of the algorithm, we compute column by column the matrices  $V$  and  $Q$  and obtain the next update of the solution  $\mathbf{u}_\gamma^*$  (and thus  $\mathbf{u}$ ) and  $\mathbf{p}$  from the system (3.3). This algorithm is a three-term recurrence, hence it is not necessary to store all the basis vectors of  $V$  and  $Q$ , but only the previous ones are needed. Please note that in each Golub-Kahan iteration, we have to solve a linear system defined by the matrix  $A_\gamma$ . We call this solution step the *inner iteration*. Depending on the size of the system, a direct solver or another iterative solver may be applied.

---

**Algorithm 1** Craig's variant algorithm.

---

**Require:**  $A_\gamma, B, W, \mathbf{b}_\gamma^*$ , maxit.

**Ensure:**  $\mathbf{u}_\gamma^*, \mathbf{p}$ .

```

1:  $\beta_1 = \|\mathbf{b}_\gamma^*\|_{W^{-1}}; \mathbf{q}_1 = W^{-1}\mathbf{b}_\gamma^*/\beta_1; \mathbf{r} = A_\gamma^{-1}B^T\mathbf{q}_1; \alpha_1 = \|\mathbf{r}\|_{A_\gamma}; \mathbf{v}_1 = \mathbf{r}/\alpha_1.$ 
2:  $\zeta_1 = \beta_1/\alpha_1; \mathbf{d}_1 = \mathbf{q}_1/\alpha_1; \mathbf{u}_\gamma^{(1)} = \zeta_1\mathbf{v}_1; \mathbf{p}^{(1)} = -\zeta_1\mathbf{d}_1.$ 
3:  $k = 0; \text{convergence} = \text{false}.$ 
4: while  $\text{convergence} = \text{false} \ \& \ k < \text{maxit}$  do
5:    $k = k + 1.$ 
6:    $\mathbf{s} = W^{-1}(B\mathbf{v}_k - \alpha_k W\mathbf{q}_k).$ 
7:    $\beta_{k+1} = \|\mathbf{s}\|_W; \mathbf{q}_{k+1} = \mathbf{s}/\beta_{k+1}.$ 
8:    $\mathbf{r} = A_\gamma^{-1}(B^T\mathbf{q}_{k+1} - \beta_{k+1}A_\gamma\mathbf{v}_k); \alpha_{k+1} = \|\mathbf{r}\|_{A_\gamma}; \mathbf{v}_{k+1} = \mathbf{r}/\alpha_{k+1}.$ 
9:    $\zeta_{k+1} = -(\beta_{k+1}/\alpha_{k+1})\zeta_k; \mathbf{d}_{k+1} = (\mathbf{q}_{k+1} - \beta_{k+1}\mathbf{d}_k)/\alpha_{k+1}.$ 
10:   $\mathbf{u}_\gamma^{(k+1)} = \mathbf{u}_\gamma^{(k)} + \zeta_{k+1}\mathbf{v}_{k+1}; \mathbf{p}^{(k+1)} = \mathbf{p}^{(k)} - \zeta_{k+1}\mathbf{d}_{k+1}.$ 
11:   $\text{convergence} \leftarrow \text{CHECK}()$ 
12: end while
13:  $\mathbf{u}_\gamma^* \leftarrow \mathbf{u}_\gamma^{(k+1)}; \mathbf{p} \leftarrow \mathbf{p}^{(k+1)}.$ 
14: return  $\mathbf{u}_\gamma^*$  and  $\mathbf{p}$ 

```

---

In Algorithm 1, the stopping criterion  $\text{CHECK}()$  is yet undefined. In the following we will use a lower bound estimate of the energy error as in [38, 55]. The  $A_\gamma$  orthogonality of  $V$  implies

$$\|\mathbf{e}^{(k)}\|_{A_\gamma}^2 = \sum_{j=k+1}^n \zeta_j^2 > \xi_{k,l}^2 := \sum_{j=k+1}^{k+l+1} \zeta_j^2,$$

with  $\mathbf{e}^{(k)} = \mathbf{u}_\gamma^* - \mathbf{u}_\gamma^{(k)}$  being the error,  $\zeta_j$  defined in Algorithm 1 and  $l > 1$  an integer. The quantity  $\xi_{k,l}$  is a lower bound for the error at step  $k-l$ . To obtain a lower bound estimate for  $\mathbf{e}^{(k)}$ , the algorithm thus needs to run  $l$  more iterations. With a stopping tolerance  $\epsilon < 1$ , the stopping criterion is then defined as

$$\text{if } \xi/\bar{\xi} \leq \epsilon, \text{ then } \text{convergence}=\text{true}, \text{ where } \xi = \sum_{j=k-l+1}^k \zeta_j^2 \text{ and } \bar{\xi}^2 = \sum_{j=1}^k \zeta_j^2.$$

We set  $l = 5$  as default in our experiments. For more details on bounds of  $\|\mathbf{e}^{(k)}\|_{A_\gamma}^2$ , we refer to [38, 55].

We will conclude with a remark about the choice of the matrix  $W$ . The matrix  $W$  can generally be any SPD matrix, and it can play two important roles. In the first case, we can relax the condition on the definiteness of  $A$ , such that it may only be symmetric positive semi-definite. With (2.1), the (1,1)-block  $A_\gamma$  is then SPD and the GKB algorithm can be applied. If  $A$  is however already SPD, this manipulation is not the purpose of the augmentation. The goal is then to obtain a linear system that is easier to solve, which translates into an improvement in convergence. For many matrices, the simple choice of  $W$  being the identity is enough [54]. In [55], the authors show that for the augmentation parameter  $\gamma$  being big enough, the algorithm converges in only a few iterations and mesh-independent convergence can be achieved. We emphasize, however, that while the iteration count for the outer Golub-Kahan method decreases, the matrix  $A_\gamma$  becomes more and more ill-conditioned. When an inner iterative solver is used, the number of inner iterations thus increases. It is important to find a good balance, and when the (1,1)-block exhibits favorable properties for certain solvers, it might be more efficient in terms of computation time to discard the augmentation. For the Stokes equation, the symmetric (1,1)-block corresponds to the discretization of the Laplace operator, for which efficient multigrid solvers exist. After experimentation, (see, e.g., [56, 57]) we decided not to use the augmentation of the (1,1)-block in Section 5 and set the parameter  $\gamma = 0$ . The transformation to obtain a zero vector in the upper right-hand side is still to be kept by using the matrix  $A$  in the transformation.

### 3.2 Augmented Lagrangian Uzawa method

The *Augmented Lagrangian–Uzawa* (ALU) method is an efficient variant of the classical Uzawa method [36]. As explained in [15, Section 8.2], it has the same solution as the saddle-point system (1.5). As before, let  $\gamma \geq 0$  be a scaling factor for the augmentation term  $B^T W^{-1} B$ . The ALU method corresponds to the Uzawa method applied to the system  $\mathcal{A}_\gamma \mathbf{x} = \mathbf{b}_\gamma$  where  $\mathcal{A}_\gamma$  is defined in (3.1) and  $\mathbf{b}_\gamma = [\mathbf{f}_\gamma, \mathbf{g}]^T$  with  $\mathbf{f}_\gamma = \mathbf{f} + \gamma B^T W^{-1} \mathbf{g}$ . In `code_saturne`, an incremental formulation of the ALU method as detailed in Algorithm 2 is considered. The previous discussion about the choice of the matrix  $W$  remains also true for ALU. We choose a diagonal matrix with entries equal to the volume of each mesh cell for  $W$ . An initial guess on the pressure (resp. velocity) field  $\mathbf{p}_0$  (resp.  $\mathbf{u}_0$ ) is needed in this method. It is either a null array in the steady case or the latest known pressure (resp. velocity) field in an unsteady case.

The efficiency and convergence of the ALU method relies on the value of the parameter  $\gamma$ . As mentioned in [15], convergence holds for  $\gamma \in (0, 2/\rho)$  where  $\rho$  is the largest eigenvalue of the Schur complement  $-BA_\gamma^{-1}B^T$ . A too large value for the parameter  $\gamma$  induces an ill-conditioned system which slows down the convergence of the inner linear system defined by  $A_\gamma$ . Once again, a trade-off between the iteration count and the cost of the inner resolution has to be found. In the experiments in Section 5, we choose  $\gamma = 100$ .

---

**Algorithm 2** Augmented Lagrangian–Uzawa algorithm (incremental form).

---

**Require:**  $A, B, W, \mathbf{f}, \mathbf{g}, \mathbf{u}_0, \mathbf{p}_0, \gamma, \epsilon, k_{\max}$ .

**Ensure:**  $\mathbf{u}, \mathbf{p}$ .

- 1: Initialise  $A_\gamma = A + \gamma B^T W^{-1} B$ ;  $\mathbf{f}_\gamma = \mathbf{f} + \gamma B^T W^{-1} \mathbf{g}$  and  $k = 1$ .
  - 2: Velocity solve:  $A_\gamma \mathbf{u}^{(1)} = \mathbf{f}_\gamma - B^T \mathbf{p}_0$ ;  $\delta \mathbf{u}^{(1)} = \mathbf{u}^{(1)} - \mathbf{u}_0$ .
  - 3: Pressure update:  $\delta \mathbf{p}^{(1)} = \gamma W^{-1} (B \mathbf{u}^{(1)} - \mathbf{g})$ ;  $\mathbf{p}^{(1)} = \mathbf{p}_0 + \delta \mathbf{p}^{(1)}$ .
  - 4: **while**  $\|\delta \mathbf{u}^{(k)}\| < \epsilon \|\delta \mathbf{u}^{(1)}\|$  &  $k < k_{\max}$  **do**
  - 5:     Solve the velocity increment:  $A_\gamma \delta \mathbf{u}^{(k+1)} = -B^T \delta \mathbf{p}^{(k)}$
  - 6:     Velocity update:  $\mathbf{u}^{(k+1)} = \delta \mathbf{u}^{(k+1)} + \mathbf{u}^{(k)}$ .
  - 7:     Pressure update:  $\delta \mathbf{p}^{(k+1)} = \gamma W^{-1} (B \mathbf{u}^{(k+1)} - \mathbf{g})$ ;  $\mathbf{p}^{(k+1)} = \mathbf{p}^{(k)} + \delta \mathbf{p}^{(k+1)}$ .
  - 8:      $k = k + 1$ .
  - 9: **end while**
  - 10:  $\mathbf{u} \leftarrow \mathbf{u}^{(k)}$  and  $\mathbf{p} \leftarrow \mathbf{p}^{(k)}$ .
  - 11: **return**  $\mathbf{u}$  and  $\mathbf{p}$ .
- 

### 3.3 Notay’s algebraic transformation

The following approach of algebraically transforming the linear system in (1.5) by a change of variables was initially introduced by Y. Notay in [34]. To keep our presentation self-contained, we briefly present the main idea. Notay’s right-hand side transformation introduces the change of variables

$$\begin{pmatrix} \mathbf{u} \\ \mathbf{p} \end{pmatrix} = \begin{pmatrix} I & -\alpha D_A^{-1} B^T \\ O & I \end{pmatrix} \begin{pmatrix} \tilde{\mathbf{u}} \\ \tilde{\mathbf{p}} \end{pmatrix} \quad (3.4)$$

which, when combined with a change of sign of the divergence constraint, yields the system  $\tilde{\mathcal{A}} \tilde{\mathbf{x}} = \tilde{\mathbf{b}}$  where

$$\tilde{\mathcal{A}} := \begin{pmatrix} A & (I - \alpha A D_A^{-1}) B^T \\ -B & \alpha B D_A^{-1} B^T \end{pmatrix}, \quad \tilde{\mathbf{x}} = \begin{bmatrix} \tilde{\mathbf{u}} \\ \tilde{\mathbf{p}} \end{bmatrix} \quad \text{and} \quad \tilde{\mathbf{b}} = \begin{bmatrix} \mathbf{f} \\ -\mathbf{g} \end{bmatrix} \quad (3.5)$$

with a positive constant  $\alpha$  and  $D_A$  being the diagonal matrix built from  $A$ . Hereafter, we assume  $\alpha = 1$ . Other transformations, e.g. the two-sided variant introduced in [58], are not considered in this work since they appear less efficient [34].

When we consider the vector  $\mathbf{v}$  of the null space of  $\tilde{\mathcal{A}}$ , we can decompose it with respect to the velocity and the pressure. It follows

$$\tilde{\mathcal{A}}\mathbf{v} = \mathbf{0} \iff \begin{cases} A\mathbf{v}_u + (I - \alpha AD_A^{-1})B^T\mathbf{v}_p = \mathbf{0} \\ -B\mathbf{v}_u + \alpha BD_A^{-1}B^T\mathbf{v}_p = \mathbf{0} \end{cases},$$

and since  $A$  is invertible, we have  $\mathbf{v}_u = -A^{-1}(I - \alpha AD_A^{-1})B^T\mathbf{v}_p$ . By substitution of  $\mathbf{v}_u$  into  $-B\mathbf{v}_u + \alpha BD_A^{-1}B^T\mathbf{v}_p = \mathbf{0}$ , we can derive  $BA^{-1}B^T\mathbf{v}_p = \mathbf{0}$  so that

$$\mathbf{v}_p^T BA^{-1}B^T\mathbf{v}_p = 0 \iff \mathbf{v}_p \in \ker(BA^{-1}B^T) = \ker(B^T).$$

By recalling the fact that the CDO discretization satisfies the solvability conditions (2.1) (e.g.  $\ker(B^T) = \mathbf{0}$ ), it implies that the kernel of  $\tilde{\mathcal{A}}$  is trivial, therefore the transformed matrix is invertible.

The (2,2)-block  $BD_A^{-1}B^T$  of (3.5) is the product of a discrete divergence with a discrete gradient which is similar to a discrete Laplacian [58]. If we replace  $D_A^{-1}$  with  $A^{-1}$ , so that the (2,2)-block is the exact Schur complement of  $A$ ,  $\tilde{\mathcal{A}}$  becomes a lower triangular block system which can be easily solved. However, the computation of  $A^{-1}$  in a large problem is often too expensive so that we may want to use an approximation of the Schur complement, instead. We also refer to [59] for sparse approximations of the Schur complement.

## 4 Algebraic multigrid preconditioning for Krylov solvers

Multigrid methods have shown to be efficient preconditioners in different settings, in particular for scalar diffusion and convection-diffusion problems [22, 30, 60–62]. Let us recall that the suboptimal performance of plain aggregation AMG schemes for diffusion operators is well known and documented in the literature [30] which has motivated the research on approaches like smoothed aggregation and the K-cycle, for instance.

The methods considered in this article expose a considerable number of options to the user. So, these AMG methods can be extensively tuned. Nevertheless, an exhaustive search of the combinatorial parameter space is beyond the scope of our current work. Their performance on a given system depends on a certain number of parameters, such as the choice of the smoother, the number of smoother iterations, the coarsening strategy (pairwise or N-times pairwise for the aggregation schemes or different aggressive coarsening techniques for the C/F schemes) and the number of levels. While it is easy to handle unstructured meshes, finding an appropriate or even optimal set of coarsening parameters is non-trivial. For example, one has to strike a balance between coarsening factors and computational costs. A too large coarsening factor (which translates into a small number of levels and in the case of plain aggregation AMG in general also implies low operator complexities) tends to increase the number of iterations to solve linear systems while its memory requirement is reduced. On the other hand, a too small coarsening factor requires a larger number of levels to reduce the size of the coarsest system below a given threshold. Most common choices

of coarsening factors and complexities of operators are given by

$$2 \leq \tau \leq 4 \quad \text{and} \quad 1.3 \leq \mathcal{C} \leq 2,$$

where  $\tau$  is the coarsening factor and  $\mathcal{C}$  denotes the operator complexity, defined by

$$\tau = \frac{1}{L} \sum_{\ell=1}^L \frac{n_\ell}{n_{\ell-1}} \quad \text{and} \quad \mathcal{C} = \frac{1}{nnz(A_L)} \sum_{\ell=0}^L nnz(A_\ell),$$

on a hierarchy of matrices  $A_\ell$ ,  $0 \leq \ell \leq L$ ,  $A_0$  being the matrix on the coarsest and  $A_L$  the matrix on the finest level with  $n_\ell$  and  $nnz(A_\ell)$  denoting the number of unknowns and the number of nonzero elements of  $A_\ell$  on each level, respectively. Following the recent work of Lin, Shadid and Tsuji [63], we have included Krylov smoothers in our comparison.

## 5 Numerical experiments

In this section, we present numerical results for the solution of the saddle-point linear system (1.5) arising from a CDO face-based discretization with the algorithms described previously. We will use the GKB method for the solution of the Stokes equations and ALU for the Navier–Stokes equations and will then compare both of them to Notay’s algebraic transformation method.

The test problems were run with the open source CFD software `code_saturne`, which in addition to in-house implementations of several Krylov solvers and preconditioners, provides interfaces to external solvers, such as MUMPS [16], Hypre [47] and PETSc [64] (e.g., please see Appendix A for the usage). In the following experiments, we employ Hypre through the PETSc interface.

In our comparison, we have representatives of different AMG approaches:

- BoomerAMG of the hypre library [46, 47, 65] relies on C/F or Ruge–Stüben coarsening [29],
- GAMG of PETSc [48] provides plain and smoothed aggregation schemes
- the (in-house) K-cycle of `code_saturne` is inspired by the algorithm described in [22] and combines therefore multiple passes pairwise aggregation coarsening with Krylov-acceleration on the intermediate levels.

Table 1 sums up the main characteristics of the AMG methods used in the computations.

Multigrid methods are among the most promising methods to solve large-scale systems and they can be used as a solver or preconditioner. In what follows, multigrid methods are always used as preconditioners. Via the interfaces to PETSc and Hypre, we also have the possibility to use the highly efficient *Hypre BoomerAMG* library, as well as the multigrid frameworks *GAMG* and *HMG* as preconditioners. Hypre BoomerAMG follows classical AMG coarsening [29, 30] with/without aggressive coarsening, while in GAMG, we use the smoothed aggregation option [31]. HMG is a hybrid version of preconditioning and one of the latest developments in the PETSc

**Table 1** Settings for the algebraic multigrid preconditioners.

	K-cycle	GAMG	BoomerAMG
Applied through	<code>code_saturne</code>	PETSc	PETSc
Type of cycle	K-cycle	V-cycle	V-cycle
Type of coarsening	pairwise aggregation [22]	smoothed aggregation [31]	classical interpolation [66]
aggreg. limit / threshold	mesh dependent, but factors satisfying $\tau \approx 4$ or $C \approx 1.3$		
max. levels	10 (i.e. $L \leq 10$ )		
Smoother			
Pre-/post-smoothing	Default: one step each; otherwise stated explicitly		
symm. Gauss-Seidel	yes	yes	yes
Krylov smoother	no	GMRES(10)	GMRES(10) (through HMG)
Stop. criterion Kryl. sm.	N/A	max 10 iter / tol = $10^{-3}$	max 10 iter / tol = $10^{-3}$

preconditioner package. It is able to employ hybrid preconditioning such as Hypre BoomerAMG, GAMG, or other multilevel methods for the construction of coarser level matrices and interpolation. More precisely, it allows us to choose any available PETSc preconditioner as smoother and the coarse solver. For example, in our test cases, we combine the context of Hypre BoomerAMG for coarsening and the context of GAMG for smoothing. We refer to [64, 67] for more details of the usage of HMG. Furthermore, we provide Appendix B for the example of PETSc options to use AMG preconditioning in our numerical experiments. `code_saturne` is not able to perform matrix-matrix multiplications and computations with non-square matrices up to now. When using Notay’s algebraic transformation, these operations have to be done in an external environment. For practical reasons, we will then employ the linear solvers given in the PETSc library without the use of the in-house K-cycle multigrid implementation in `code_saturne`.

For the non-symmetric system matrix  $\tilde{\mathcal{A}}$  of Notay’s algebraic transformation, we combine adequate Krylov methods with GAMG or Hypre (BoomerAMG) preconditioners. While we apply coarsening schemes only to  $A_\gamma$  in GKB and ALU, the construction of coarser level operators for the whole transformed  $\mathcal{A}$  will be required for the algebraic transformation.

### *General setting for outer solvers*

We compare and discuss combinations of solvers and preconditioners for the three methods GKB, ALU and Notay’s algebraic transformation. The choice of settings for the GKB and ALU method with respect to the inner and outer solvers and a symmetric or non-symmetric inner system can be found in Table 2. For the non-symmetric matrix in the algebraic transformation, we use FGMRES. We choose  $10^{-8}$  as the stopping tolerance and set the maximum number of iterations to 10 000. As preconditioners for this system, as well as for the inner systems in the ALU and GKB methods, we use algebraic multigrid methods. The setups and different combinations are given in Table 1. We want to emphasize that the choice of parameters, i.e. to define a threshold, for the construction of aggregation or prolongation is a significantly important matter. In particular, in case of GAMG for Stokes systems, this parameter must not be low



and to provide relevant coarser levels, we found that  $\mathcal{C} \approx 1.3$ . Further particular configurations are discussed during the description of the numerical experiments.

### *Computational setup for the test cases*

We finish this section with giving the configurations for the test cases:

- Hardware: Intel(R) Core(TM) i9-10885H CPU @ 2.40GHz, 32GB
- The version of `code_saturne`: `code_saturne7.1-alpha`
- The version of PETSc: 3.15.0
- No parallel computing (neither MPI nor openMP), i.e. running sequentially

**Table 2** Settings for the GKB and ALU algorithms.

	GKB	ALU
Outer tolerance	$10^{-8}$	$10^{-7}$
Augment. parameter $\gamma$	0	100
Inner solver settings		
Inner tolerance	$10^{-8}$	$10^{-7}$
symm. inner solver	FCG	-
non-symm. inner solver	-	FGMRES
Preconditionner (PETSc)	Hypre, GAMG	
Preconditionner ( <code>code_saturne</code> )	K-cycle	

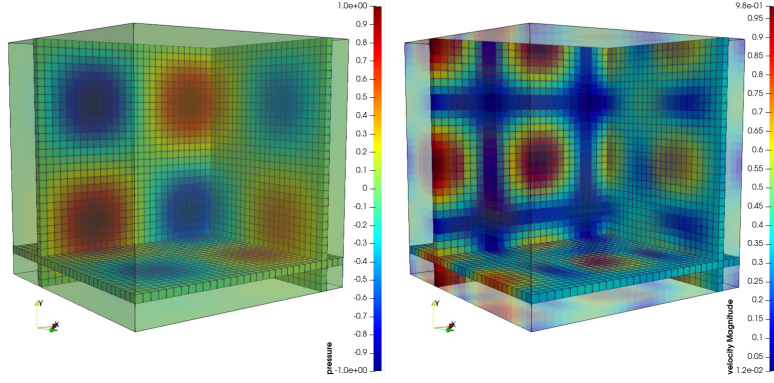
## 5.1 Stokes problem (3D TGV problem)

The first test case is an adaptation of the Taylor–Green Vortices (TGV) problem with an analytic solution consisting of the product of sine and cosine functions. Its strong solution satisfies the steady Stokes equations, i.e. the equations (1.1) and (1.2) with  $\chi = 0$  and  $\nu = 1$ . The strong solution is given by

$$\vec{u} = \begin{pmatrix} 0.5 \sin(2\pi x) \cos(2\pi y) \cos(2\pi z) \\ 0.5 \cos(2\pi x) \sin(2\pi y) \cos(2\pi z) \\ -\cos(2\pi x) \cos(2\pi y) \sin(2\pi z) \end{pmatrix} \quad \text{and} \quad p = \sin(2\pi x) \sin(2\pi y) \sin(2\pi z),$$

where  $(x, y, z) \in [0, 1]^3$  and  $\vec{f}$  is given to fulfil (1.1). The solution is presented graphically in Figure 2.

For the discretization, we will use two types of meshes. The first one is Cartesian and the second one is tetrahedral. In the abbreviations used in the following, we denote the Cartesian meshes by  $H$  and the tetrahedral ones by  $T$ . These letters are followed by the number of elements. The resulting numbers of degrees of freedom for the velocity and pressure components are given in Table 3. Since this 3D TGV problem leads to a symmetric matrix, we may use the GKB algorithm. We will compare its performance to Notay’s algebraic transformation approach.



**Fig. 2** 3D TGV problem: pressure field (left) and velocity field (right).

**Table 3** 3D TGV problem: the degrees of freedom with respect to meshes.

Mesh	Degrees of freedom		
	Velocity	Pressure	Total unknowns
H32	304 128	32 768	336 896
H64	2 396 160	262 144	2 658 304
T5	188 361	30 480	218 841
T6	374 964	61 052	436 016

### 5.1.1 Solution by GKB

The GKB algorithm is a segregated method for matrices of type (1.5) with a symmetric (1,1)-block and we need, as explained earlier, an inner iterative solver for the (1,1)-block. Here, we choose a preconditioned flexible conjugate gradient method. In general, we can employ the GKB algorithm via PETSc or directly in `code_saturne`, but in the following experiments we choose the in-house implementation. Since the system of inner iterations is symmetric, we are able to use GAMG with smoothed aggregation. Table 4 illustrates the elapsed times and the numbers of iterations, where each column corresponds to the respective preconditioner of FCG. The numbers of iterations are sufficiently good for those three preconditioners. However, in terms of setup time, the in-house K-cycle aggregation step shows a much better performance than PETSc. To summarise the result of Table 4, we have

Setup time: K-cycle < Hypre BoomerAMG < GAMG  
 Solution time: GAMG < K-cycle < Hypre BoomerAMG  
 Total time for H32: K-cycle < GAMG < Hypre BoomerAMG  
 Total time for H64: GAMG < K-cycle < Hypre BoomerAMG  
 The number of iterations: Hypre BoomerAMG < GAMG < K-cycle

These results show that a higher setup time might be recovered by a faster solution process, and also that a low number of iterations might not necessarily mean that the

method performs best in practice. We furthermore see that the performance of the solvers is mesh-independent. The winner in the total time to solution is GAMG for H64, whereas for H32, it is the in-house K-cycle (although, admittedly, not by far).

**Table 4** 3D Stokes: performance of numerical methods for Stokes system with GKB and AMG preconditioned FCG as inner solver.

Strategy	GKB					
Mesh	H32			H64		
Preconditioner	K-cycle	Hypre	GAMG	K-cycle	Hypre	GAMG
Setup time (s)	0.96	2.84	3.77	7.97	25.6	31.8
Solution time (s)	18.1	21.6	15.9	173.6	187.4	140.5
Total time (s)	19.1	24.4	19.7	181.6	213.0	172.3
# iterations (inner/outer/total)	13/21/283	5/21/105	7/21/147	13/23/299	5/22/110	7/23/161

*Remark 5.1* In our numerical experiments, the GKB algorithm is used without augmentation, i.e., we set  $\gamma = 0$ . On the other hand, the ALU algorithm requires a rather large augmentation parameter to show a good performance even in the symmetric case. The downside is the resulting, more ill-conditioned matrix  $A_\gamma$  in the inner solution step. As pointed out in [51], due to its favorable performance caused mainly by an efficient inner solution, the GKB algorithm is preferred to the ALU algorithm in Stokes problems. Therefore, we do not provide any further discussion for the ALU method in the symmetric case.

### 5.1.2 Solution by algebraic transformation

As second approach, we apply Notay’s algebraic transformation and solve the resulting system with FGMRES preconditioned by multigrid. For BoomerAMG, we keep the settings of Table 1, whereas we follow the official PETSc manual (please see [64, Section Algebraic Multigrid (AMG) Preconditioners]) for recommended settings of GAMG parameters for non-symmetric problems (e.g., see Appendix B.2). Results are presented in Table 5. Although Hypre requires a higher time for setup, its overall solution time is smaller than the one of GAMG. This effect is especially visible for the mesh H64.

For comparing these results to the GKB method, the measure of numbers of iterations can not be used, since the transformed matrix is of another size than  $\mathcal{A}_\gamma$  and thus the cost of one iteration is different. Comparing the time to solution in Tables 4 and 5, we observe that Hypre on the transformed system is faster than the any of the tested multigrid preconditioners in the GKB formulation.

We have carried out further test cases to study the influence of different cycle-types for the multigrid method. As seen in Table 6, using multigrid cycles other than V(1,1) in BoomerAMG does not show any improvement in terms of iteration numbers and solution times, since the interpolation operator of BoomerAMG is good enough. We observe that increasing the number of relaxation steps in GAMG reduces iteration

**Table 5** 3D Stokes: algebraic transformation with AMG preconditioned FGMRES.

Strategy	Algebraic transformation			
	H32		H64	
Mesh				
Preconditioner	Hypre	GAMG	Hypre	GAMG
Setup time (s)	6.344	4.029	54.707	32.965
Solution time (s)	12.243	16.589	105.760	315.444
Total time to solution	18.587	20.618	160.467	348.409
# iterations	36	64	36	168

counts, see Table 6. For the non-symmetric system under consideration, BoomerAMG outperforms GAMG. We have empirically tried to figure out optimal settings for the usage of GAMG, but no choice worked satisfactorily. Thus, hereafter, only Hypre BoomerAMG is used in combination with symmetric Gauss-Seidel smoothing for the preconditioning step in the algebraic transformation strategy.

**Table 6** 3D Stokes: algebraic transformation with respect to multigrid cycles on H32.

Iterative solver	FGMRES					
	Hypre BoomerAMG			GAMG		
Preconditioner						
Multigrid cycle	V(1,1)	V(2,2)	V(3,3)	V(1,1)	V(2,2)	V(3,3)
Total time (s)	11.540	14.911	18.832	17.317	18.174	19.407
# iterations	26	28	30	123	77	59

In a second trial, we solved the system arising in the change of variable approach on the tetrahedral meshes T5 and T6 with Hyper BoomerAMG. Table 7 indicates that the algebraic transformation of the matrix works also well for tetrahedral meshes. While increasing the number of smoothing steps does not improve iteration numbers on the Cartesian grid (Table 6), more pre- and post-smoothing steps do reduce the number of iterations for tetrahedral meshes, see Table 7, since the tetrahedral meshes are more complex. In terms of the elapsed time, V(3,3) is however slower than V(1,1).

**Table 7** 3D Stokes: V(1,1) and V(3,3) on tetrahedral meshes with Hypre BoomerAMG.

Iterative solver	FGMRES			
	T5		T6	
Mesh				
Multigrid cycle	V(1,1)	V(3,3)	V(1,1)	V(3,3)
Total time (s)	21.608	27.648	58.962	71.328
# iterations	37	26	37	27

Stationary and relatively basic algorithms such as the (weighted) Jacobi or Gauss-Seidel iterations are frequently used as smoothers in multigrid cycles. However, according to [63, 68] Krylov methods can be relevant as smoothers in multigrid settings as well.

In the following, we report on FGMRES preconditioned by AMG (here, GAMG) with Krylov-smoothing for the non-symmetric problems. For the Krylov smoother, we allow up to 10 GMRES iterations combined with an inner tolerance of  $10^{-3}$ , which stops when either one or the other criterion is reached. We may now think in a nested way and precondition the smoother itself, e.g., the Gauss-Seidel method as preconditioner for the smoother GMRES(10). At the coarsest level, we solve the system with a direct solver. Table 8 shows that Krylov smoothing reduces in general the number of iterations. However, comparing these results to Table 5, the elapsed time is greater than for Gauss-Seidel smoothing. Furthermore, we see a dependence of the solver performances on the meshes, i.e., the iteration numbers are unstable.

**Table 8** 3D Stokes: GAMG with Krylov smoothers  
FGMRES[30](GAMG(GMRES(10)+X)).

Mesh	Smoother	# it	Time (s)
H32	GMRES	29	175.226
	GMRES+SGS	3	39.534
H64	GMRES	48	2426.144
	GMRES+SGS	29	2980.572
T6	GMRES	84	715.284
	GMRES+SGS	11	187.401

Although Krylov smoothing cannot decrease the elapsed time in GAMG, it may lead to smaller iteration numbers for combinations of meshes and solvers. The HMG context of PETSc allows us to combine Krylov smoothers with classical AMG coarsening in Hypr. In Table 9, it is shown that HMG with symmetric Gauss-Seidel smoothing shows similar numerical results as Hypr BoomerAMG. On the other hand, using Krylov smoothers sometimes reduces the number of iterations, whereas more time is spent in the solver. Similarly, as for Krylov smoothing in GAMG, the performances depend on the mesh and the experiments failed for fine meshes. Thus, we do not observe any advantage from using HMG with Krylov smoothing in Stokes systems for our particular test cases.

### 5.1.3 Summary for the Stokes test case

Our study compared the GKB and algebraic transformation methods in the Stokes test case. We found that AMG preconditioned GMRES solvers were robust in solving the inner system of the GKB method. The three distinct coarsening strategies yielded comparable results and are independent of mesh sizes. In the algebraic transformation, only the classical AMG via Hypr BoomerAMG showed robustness. We attempted to improve the numerical performance of GAMG by employing Krylov smoothers, but this approach required much more computational cost and the results depended

**Table 9** 3D Stokes: HMG preconditioning FGMRES[30](HMG), Boomer classical coarsening with max it=200 (if # it  $\geq$  200, the test fails).

Mesh	Smoother	# it	Time (s)
H32	SGS	28	13.483
	GMRES+SGS	135	338.316
H64	SGS	29	119.388
	GMRES+SGS		Failed
T6	SGS	32	52.344
	GMRES+SGS		Failed

on the meshes. Furthermore, with the proper use of AMG preconditioning, the algebraic transformation approach was slightly faster than the GKB method. Therefore, we recommend using the algebraic transformation with BoomerAMG employing the symmetric Gauss-Seidel smoother in practice, as it is a monolithic method that yields better numerical performance.

## 5.2 Navier–Stokes problem (2D Burggraf problem)

As second test case, we consider the steady incompressible Navier–Stokes problem given by equations (1.1) and (1.2) with  $\chi = 1$  and  $\nu = 1/Re$ . To resolve the non-linearity, we apply Picard’s iteration and solve the Oseen’s problem (1.3)-(1.4). As for the Stokes problem, we are interested in solving the saddle-point systems arising from a CDO discretization scheme. Due to the convection terms of the velocity fields, the (1,1)-block becomes non-symmetric. The test problem in the following is a 2D Burggraf flow [69], which is a 2D analytic polynomial solution to the Navier–Stokes equation. The exact solution is given by

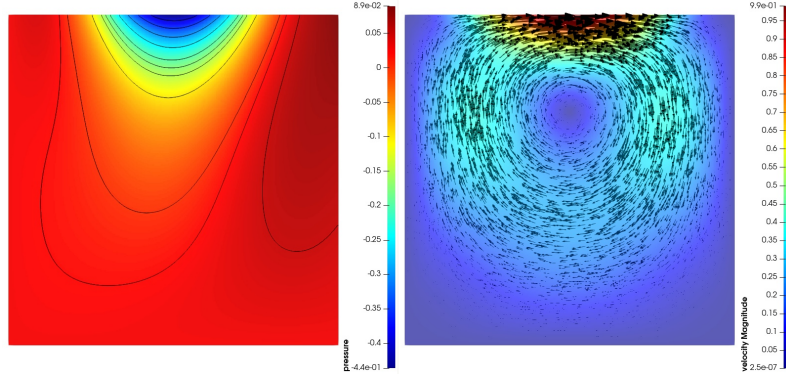
$$\begin{aligned} \vec{u} &= \begin{pmatrix} 16x^2(x^2 - 2x + 1)y(2y^2 - 1) \\ -16x(2x^2 - 3x + 1)y^2(y^2 - 1) \end{pmatrix}, \\ p &= \frac{8}{Re} \left( 24x^3(0.2x^2 - 0.5x + 1/3) + 4x(2x^2 - 3x + 1)(12y^2 - 2) + (24x - 12)(y^2(y^2 - 1)) \right. \\ &\quad + 64 \left( 0.5(x^2(x^2 - 2x + 1))^2(24y^3(y^2 - 1) - 2y(12y^2 - 2)(2y^2 - 1)) \right. \\ &\quad \left. \left. - 2y^3(y^2 - 1)(2y^2 - 1)(x^2(x^2 - 2x + 1)(12x^2 - 12x + 2) - 4x^2(2x^2 - 3x + 1)^2) \right) \right). \end{aligned}$$

For more details about the exact solution, we refer to [51, 69]. As a default parameter, we assume the Reynolds number to be  $Re = 100$ . The solution is graphically presented in Figure 3. For the 2D Burggraf model, we only consider the Cartesian meshes H128 and H256. The numbers of degrees of freedom for  $u$  and  $p$  on these two meshes are given in Table 10. Furthermore we use the stopping criterion

$$\|\vec{u}^k - \vec{u}^{k-1}\| \leq 10^{-7} \quad \text{or} \quad \max k = 50.$$

for the Picard’s iteration scheme.

Due to the non-symmetry of the (1,1)-block, the GKB method is no longer applicable but see also [70]. Notably, the algebraic transform approach remains a viable



**Fig. 3** 2D Burggraf flow with  $Re = 100$ : pressure field (left) and velocity field (right).

**Table 10** 2D Burggraf problem: the degrees of freedom with respect to meshes.

Mesh	Degrees of freedom		
	Velocity	Pressure	Total unknowns
H128	197 376	16 384	213 760
H256	787 968	65 536	853 504

option in this scenario. For comparison, we also employ the ALU method, as described in Section 3.2.

### 5.2.1 Solution by ALU method

As first test, we use the ALU method with an augmentation parameter  $\gamma = 100$ . For the inner solver, we compare the LU factorization of MUMPS with GMRES preconditioned by Hypr BoomerAMG. The results in Table 11 show that with both inner solvers, BoomerAMG or MUMPS, ALU converges in only three iterations and the Picard's iterations need three to five steps. In terms of computation time, the usage of an inner iterative solver is not competitive. Here we see the influence of the parameter  $\gamma$ . With  $\gamma = 100$ , the outer ALU method converges in only 3 iterations, and this for a direct or an iterative inexact solver inside. However the inner matrix  $A_\gamma$  becomes ill-conditioned and thus GMRES needs a high number of iterations to converge, also to the required, rather relaxed, tolerance of  $10^{-8}$ .

### 5.2.2 Solution by algebraic transformation

As next experiment, we first transform the matrix along Notay's change of variable approach and then solve the resulting matrix either with MUMPS or GMRES preconditioned by BoomerAMG with symmetric Gauss-Seidel smoothing. The results are given in Table 12. Also here, we see that for these medium size 2D Navier-Stokes test cases, the direct solver MUMPS leads to a smaller computation time than GMRES. Compared to ALU, the latter converges faster. This is explained by the size of the

**Table 11** 2D Navier–Stokes: ALU ( $\gamma = 100$ ) at  $Re = 100$ .

Strategy	ALU			
	H128		H256	
Mesh				
Inner Solver	MUMPS	GMRES	MUMPS	GMRES
Precond.	-	Hypre	-	Hypre
Time(s)	7.330	752.140	23.150	6,071.680
# Picard It	4	5	3	4
# ALU It	3	3	3	3
# inner It (average)	1	4 121	1	9 446

matrices to be solved. While in the algebraic transformation approach, the whole system is solved by Mumps, the linear system in the ALU method is only of the size of the velocity block. However, when we use a Krylov iterative solvers in both approaches, then the algebraic transformation wins in terms of computation time.

**Table 12** 2D Navier–Stokes: algebraic transformation at  $Re = 100$ .

Strategy	algebraic transformation			
	H128		H256	
Mesh				
Solver	MUMPS	GMRES	MUMPS	GMRES
Precond.	-	Hypre	-	Hypre
Time(s)	13.798	58.235	46.540	304.380
# Picard It	4	4	3	3
# It	1	139	1	190

As earlier for the Stokes system, we have carried out numerical simulations using AMG combined with Krylov smoothing. In particular, the smoother consists of GMRES(10) with the ILU(0) preconditioner. The results in Table 13 show that comparing the different solver combinations by the number of required iterations might be misleading. While GAMG with Krylov smoothers on the H128 mesh divides the iteration count of BoomerAMG with the simple symmetric Gauss-Seidel smoother by a factor of 5, the total computation time is however about three-times more than the HMG case with symmetric Gauss-Seidel smoothing. On the finer mesh H256, HMG with the Krylov smoother even fail to converge. As for BoomerAMG, HMG on H128 with symmetric Gauss-Seidel smoothing is faster than its counterpart with Krylov smoothing. For both of them, it is interesting to note that the iteration count increases only slightly from H128 to H256. Among these two solvers, HMG with a symmetric Gauss-Seidel smoother has shown the fastest convergence, is however still beaten by the ALU method with MUMPS as inner solver.

### 5.2.3 Reynolds numbers

Lastly, we investigate how the variation of the Reynolds number influences the performance in the algebraic transformation. As the Reynolds number increases, the



**Table 13** 2D Navier–Stokes: Hypre BoomerAMG, GAMG and HMG in the algebraic transformation with max it =200.

Mesh	AMG	Smoother	# iterations			Time (s)	
			Picard	inner	total	setup	solve
H128	Hypre	SGS	4	139	557	6.155	58.235
	GAMG	GMRES+ILU	4	27	109	7.264	113.461
	HMG	SGS	4	111	447	6.749	36.720
		GMRES+ILU	4	83	333	6.778	466.188
H256	Hypre	SGS	3	190	570	20.552	304.380
	GAMG	GMRES+ILU	3	109	328	21.862	1191.965
	HMG	SGS	3	158	475	23.700	178.412
		GMRES+ILU		Failed			N/A

nonlinear term in the Navier–Stokes equation becomes increasingly important, which makes the system of equations more difficult to solve. This is because the nonlinear term causes the solution to depend on itself, which leads to an iteration cycle that can be difficult to resolve. In particular, the (1,1)-block  $A$  is ill-conditioned and dominated by the convective term. Although the Picard iteration can be an effective approach for solving nonlinear systems of equations, linearizing the equation can lead to convergence issues, especially for highly nonlinear problems. Careful attention must be paid to the initial guess and other aspects of the method to ensure convergence to a solution.

Previously, we observed that the ALU method was not well-suited to iterative solvers due to the augmentation parameter yielding the ill-conditioned  $A_\gamma$ , while the algebraic transformation worked well with Krylov solvers. To investigate the robustness of AMG preconditioned Krylov solvers, we should consider the algebraic transformation approach in this experiment. We focus on the numerical performance of the algebraic transformation varying with Reynolds numbers. We therefore first transform the system algebraically and apply the best solver combination up to date, i.e. GMRES preconditioned by HMG with symmetric Gauss-Seidel smoothing.

Table 14 indicates that the HMG framework works well for low Reynolds numbers, but the performance deteriorates for higher ones. For  $Re = 10\,000$ , the solution even diverged. Indeed, low Reynolds numbers imply that the velocity block is dominated by the diffusion terms, this means that it has relatively good spectral properties. In contrast, high Reynolds numbers make the transformed matrix difficult to solve with iterative methods. Therefore, it is necessary to find another numerical remedy. This might already begin with exchanging the standard way of Picard iterations with another method, as they might not be sufficiently suitable for numerical simulations of flows with high Reynolds numbers. Hence, another numerical treatment might be needed such as introducing artificial terms, e.g. the augmented Lagrangian formula [42, 43]. Moreover, we refer to [71] and [44, 72] for a rotation form and a vector projection method, respectively.

**Table 14** 2D Navier–Stokes: GMRES[30](HMG(SGS)) in the algebraic transformation with respect to  $Re$  on H128.

Mesh	$Re$	# iterations			Time (s)	
		Picard	inner	total	setup	solve
H128	1	2	23	46	3.009	5.751
	10	2	25	51	3.021	6.190
	100	4	111	447	6.749	36.720
	1 000	9	403	5 492	23.400	534.129

#### 5.2.4 Summary for the Navier–Stokes test case

As in the Stokes problem, we could solve the linearized Navier–Stokes operator by AMG preconditioned FGMRES with the algebraic transformation strategy. For the fixed Reynolds number of 100 and the moderately sized matrices used, the ALU method with an inner direct solver outperformed other methods. However, these results may not remain valid as the system size increases beyond a critical point, making it impractical to use a direct solver. In such cases, iterative solvers become necessary. Encouraged by the found best choice, we also compared its numerical performance for varying Reynolds numbers. It is observed that the numerical performance is similar to the case of Stokes when the Reynolds number is low. However, if a high Reynolds number is imposed, our method shows poor results or fails. Therefore, our recommendation is valid with low Reynolds numbers and further investigation is required for the case of higher ones.

## 6 Conclusion and discussion

This paper addresses the steady-state incompressible Stokes and Navier–Stokes problems using CDO face-based discretization schemes. We compare three different numerical approaches to solve saddle-point problems. Overall, the algebraic transformation exhibits the benefits of the monolithic scheme by the absence of additional outer iterations and shows better numerical performance.

In the Stokes case, the GKB strategy is found to be robust when using any AMG preconditioned GMRES solver, whereas the algebraic transformation requires Hypr BoomerAMG for robustness. For the Navier–Stokes problem, the ALU method shows reasonable performance only with a direct solver due to the ill-conditioned inner systems imposed by the augmented terms. However, the algebraic transformation remains well-suited to AMG preconditioned Krylov solvers for low Reynolds number problems. When good coarsening operators are not available, Krylov smoothers improve convergence but at the cost of increased computational expenses and varying numerical performance depending on the problem.

## Statements and Declarations

**Conflict of interest.** The authors declare that they have no conflict of interest.

**Acknowledgments.** The authors would like to acknowledge the support from EDF R&D.

**Author contributions.**

- Y.Jang: Writing- Original draft preparation, Writing - Review & Editing, Investigation, Software, Visualization.
- J.Bonelle: Writing - Review & Editing, Methodology, Software, Resources.
- C.Kruse: Writing - Review & Editing, Methodology.
- F.Hülsemann: Writing - Review & Editing, Conceptualization.
- U.Rüde: Writing - Review & Editing, Supervision.

**Data availability.** The data that support the findings of this study are available from the corresponding author, Y. Jang, upon reasonable request. The CFD tool `code_saturne` is freely available and we refer to Appendix and an official document in <https://www.code-saturne.org>.

**ORCID.**

Yongseok Jang  <https://orcid.org/0000-0002-2036-558X>  
Jérôme Bonelle  <https://orcid.org/0000-0002-8164-4646>  
Carola Kruse  <https://orcid.org/0000-0002-4142-7356>  
Frank Hülsemann  <https://orcid.org/0000-0001-5736-4532>  
Ulrich Rüde  <https://orcid.org/0000-0001-8796-8599>

## References

- [1] Bonelle, J., Ern, A.: Analysis of compatible discrete operator schemes for elliptic problems on polyhedral meshes. *Esaim. Math. Model. Numer. Anal* **48**(2), 553–581 (2014) <https://doi.org/10.1051/m2an/2013104>
- [2] Bossavit, A.: On the geometry of electromagnetism. *J. Japan Soc. Appl. Electromagn. & Mech.* **6**, 17–281 (1998)
- [3] Hyman, J., Scovel, J.: Deriving mimetic difference approximations to differential operators using algebraic topology. Los Alamos National Laboratory (1988)
- [4] Desbrun, M., Hirani, A.N., Leok, M., Marsden, J.E.: Discrete Exterior Calculus (2005). <http://arxiv.org/abs/math/0508341>
- [5] Codecasa, L., Specogna, R., Trevisan, F.: A new set of basis functions for the discrete geometric approach. *J. Comput. Phys.* **229**(19), 7401–7410 (2010) <https://doi.org/10.1016/j.jcp.2010.06.023>
- [6] Palha, A., Pinto Rebelo, P., Hiemstra, R., Kreeft, J., Gerritsma, M.: Physics-compatible discretization techniques on single and dual grids, with application to the Poisson equation of volume forms. *J. Comput. Phys.* **257**, 1394–1422 (2014) <https://doi.org/10.1016/j.jcp.2013.08.005>

- [7] Brezzi, F., Lipnikov, K., Shashkov, M.: Convergence of the Mimetic Finite Difference method for diffusion problems on polyhedral meshes. *SIAM J. Numer. Anal.* **43**(5), 1872–1896 (2005) <https://doi.org/10.1137/040613950>
- [8] Droniou, J., Eymard, R., Gallouët, T., Herbin, R.: A unified approach to Mimetic Finite Difference, Hybrid Finite Volume and Mixed Finite Volume methods. *Math. Model. Methods Appl. Sci.* **20**(2), 265–295 (2010) <https://doi.org/10.1142/S0218202510004222>
- [9] Arnold, D.N., Falk, R.S., Winther, R.: Finite element exterior calculus: from Hodge theory to numerical stability. *Bull. Amer. Math. Soc. (N.S.)* **47**, 281–354 (2010) <https://doi.org/10.1090/S0273-0979-10-01278-4>
- [10] Lashuk, I.V., Vassilevski, P.S.: The construction of the coarse de Rham complexes with improved approximation properties. *Computational Methods in Applied Mathematics* **14**(2), 257–303 (2014) <https://doi.org/10.1515/cmam-2014-0004>
- [11] Beirão da Veiga, L., Brezzi, F., Cangiani, A., Manzini, G., Marini, L.D., Russo, A.: Basic principles of Virtual Element Methods. *Math. Model. Methods Appl. Sci.* **23**(1), 199–214 (2013) <https://doi.org/10.1142/S0218202512500492>
- [12] Di Pietro, D.A., Ern, A., Lemaire, S.: An arbitrary-order and compact-stencil discretization of diffusion on general meshes based on local reconstruction operators. *Comput. Methods Appl. Math.* **14**(4), 461–472 (2014) <https://doi.org/10.1515/cmam-2014-0018>
- [13] Di Pietro, D.A., Droniou, J.: An Arbitrary-Order Discrete de Rham Complex on Polyhedral Meshes: Exactness, Poincaré Inequalities, and Consistency. *Found. Comput. Math.* **23**(1), 85–164 (2023) <https://doi.org/10.1007/s10208-021-09542-8>
- [14] Bonelle, J., Ern, A., Milani, R.: Compatible discrete operator schemes for the steady incompressible Stokes and Navier–Stokes equations. In: Klöforn, R., Keilegavlen, E., Radu, F.A., Fuhrmann, J. (eds.) *International Conference on Finite Volumes for Complex Applications*, pp. 93–101 (2020). [https://doi.org/10.1007/978-3-030-43651-3\\_6](https://doi.org/10.1007/978-3-030-43651-3_6). Springer
- [15] Benzi, M., Golub, G.H., Liesen, J.: Numerical solution of saddle point problems. *Acta numer.* **14**, 1–137 (2005) <https://doi.org/10.1017/S0962492904000212>
- [16] Amestoy, P., Duff, I., L’Excellent, J.-Y.: A fully asynchronous multifrontal solver using distributed dynamic scheduling. *SIAM J. Matrix Anal. Appl.* **23**(1), 15–41 (2020) <https://doi.org/10.1137/S0895479899358194>
- [17] Saad, Y.: *Iterative Methods for Sparse Linear Systems*. SIAM, Philadelphia (2003). <https://doi.org/10.1137/1.9780898718003>

- [18] Saad, Y., Schultz, M.H.: GMRES: A generalized minimal residual algorithm for solving nonsymmetric linear systems. *SIAM J. Sci. Stat. Comp.* **7**(3), 856–869 (1986) <https://doi.org/10.1137/0907058>
- [19] Hestenes, M.R., Stiefel, E.: Methods of conjugate gradients for solving linear systems. *Journal of research of the National Bureau of Standards* **49**, 409–435 (1952) <https://doi.org/10.6028/jres.049.044>
- [20] Saad, Y.: A flexible inner-outer preconditioned GMRES algorithm. *SIAM J. Sci. Comput.* **14**(2), 461–469 (1993) <https://doi.org/10.1137/0914028>
- [21] Eisenstat, S.C., Elman, H.C., Schultz, M.H.: Variational iterative methods for nonsymmetric systems of linear equations. *SIAM J. Numer. Anal.* **20**(2), 345–357 (1983) <https://doi.org/10.1137/0720023>
- [22] Notay, Y.: An aggregation-based algebraic multigrid method. *Electron. T. Numer. Ana.* **37**(6), 123–146 (2010)
- [23] Notay, Y.: Flexible conjugate gradients. *SIAM J. Sci. Comput.* **22**(4), 1444–1460 (2000) <https://doi.org/10.1137/S1064827599362314>
- [24] Farahbakhsh, I.: *Krylov Subspace Methods with Application in Incompressible Fluid Flow Solvers*. Wiley, Hoboken, NJ (2020). <https://doi.org/10.1002/9781119618737>
- [25] Meurant, G., Tebbens, J.D.: *Krylov Methods for Nonsymmetric Linear Systems: From Theory to Computations*. Spr. S. Comp. Math. Springer International Publishing, Cham (2020). <https://doi.org/10.1007/978-3-030-55251-0>
- [26] Ahmed, N., Bartsch, C., John, V., Wilbrandt, U.: An assessment of some solvers for saddle point problems emerging from the incompressible Navier-Stokes equations. *Comput. Methods in Appl. Mech. Eng.* **331**, 492–513 (2018) <https://doi.org/10.1016/j.cma.2017.12.004>
- [27] Brandt, A., McCormick, S., Hufe, J.: Algebraic multigrid (AMG) for sparse matrix equations. *Sparsity and its Applications* **257** (1985)
- [28] Brandt, A.: Algebraic multigrid theory: The symmetric case. *Appl. Math. Comput.* **19**(1-4), 23–56 (1986)
- [29] Ruge, J.W., Stüben, K.: 4. Algebraic Multigrid, pp. 73–130. <https://doi.org/10.1137/1.9781611971057.ch4> . <https://epubs.siam.org/doi/abs/10.1137/1.9781611971057.ch4>
- [30] Stüben, K.: A review of algebraic multigrid. *J. Comput. Appl. Math.* **128**(1), 281–309 (2001) [https://doi.org/10.1016/S0377-0427\(00\)00516-1](https://doi.org/10.1016/S0377-0427(00)00516-1) . Numerical Analysis 2000. Vol. VII: Partial Differential Equations

- [31] Vaněk, P., Mandel, J., Brezina, M.: Algebraic multigrid by smoothed aggregation for second and fourth order elliptic problems. *Computing* **56**(3), 179–196 (1996) <https://doi.org/10.1007/BF02238511>
- [32] Notay, Y.: Aggregation-based algebraic multigrid for convection-diffusion equations. *SIAM J. Sci. Comput.* **34**(4), 2288–2316 (2012) <https://doi.org/10.1137/110835347>
- [33] Xu, J., Zikatanov, L.: Algebraic multigrid methods. *Acta Numer.* **26**, 591 (2017) <https://doi.org/10.1017/S0962492917000083>
- [34] Notay, Y.: Algebraic multigrid for Stokes equations. *SIAM J. Sci. Comput.* **39**(5), 88–111 (2017) <https://doi.org/10.1137/16M1071419>
- [35] Bacq, P.-L., Notay, Y.: A new semialgebraic two-grid method for Oseen problems. *SIAM J. Sci. Comput.* **0**(0), 226–253 (2022) <https://doi.org/10.1137/21M1429011>
- [36] Arrow, K.J., Hurwicz, L., Uzawa, H.: *Studies in Linear and Nonlinear Programming*. Stanford University Press, Stanford (1958)
- [37] Fortin, M., Glowinski, R.: *Augmented Lagrangian Methods: Applications to the Numerical Solution of Boundary-Value Problems* vol. 15 in *Studies in Mathematics and its Applications*. North-Holland Publishing Co., Amsterdam (1983)
- [38] Arioli, M.: Generalized Golub–Kahan Bidiagonalization and Stopping Criteria. *SIAM J. Matrix Anal. Appl.* **34**(2), 571–592 (2013) <https://doi.org/10.1137/120866543>
- [39] Elman, H., Silvester, D., Wathen, A.: *Finite Elements and Fast Iterative Solvers: With Applications in Incompressible Fluid Dynamics*, 2nd edn. Oxford University Press, Oxford, New York (2014). <https://doi.org/10.1093/acprof:oso/9780199678792.001.0001>
- [40] Ur Rehman, M., Geenen, T., Vuik, C., Segal, G., MacLachlan, S.: On iterative methods for the incompressible Stokes problem. *Int. J. Numer. Methods Fluids* **65**(10), 1180–1200 (2011) <https://doi.org/10.1002/FLD.2235>
- [41] Segal, A., Ur Rehman, M., Vuik, C.: Preconditioners for incompressible Navier–Stokes solvers. *Numer. Math. Theor. Meth. Appl.* **3**(3), 245–275 (2010) <https://doi.org/10.4208/nmtma.2010.33.1>
- [42] Benzi, M., Olshanskii, M.A., Wang, Z.: Modified augmented Lagrangian preconditioners for the incompressible Navier–Stokes equations. *Int. J. Numer. Methods Fluids* **66**(4), 486–508 (2011) <https://doi.org/10.1002/fld.2267>

- [43] Farrell, P.E., Mitchell, L., Wechsung, F.: An Augmented Lagrangian Preconditioner for the 3D Stationary Incompressible Navier–Stokes Equations at High Reynolds Number. *SIAM J. Sci. Comp.* **41**(5), 3073–3096 (2019) <https://doi.org/10.1137/18M1219370>
- [44] Angot, P., Caltagirone, J.-P., Fabrie, P.: A new fast method to compute saddle-points in constrained optimization and applications. *Appl. Math. Lett.* **25**(3), 245–251 (2012) <https://doi.org/10.1016/j.aml.2011.08.015>
- [45] Notay, Y., Vassilevski, P.S.: Recursive Krylov-based multigrid cycles. *Numer. Linear Algebra Appl.* **15**(5), 473–487 (2008) <https://doi.org/10.1002/nla.542>
- [46] *hypre: High Performance Preconditioners*. <https://llnl.gov/casc/hypre>, <https://github.com/hypre-space/hypre>
- [47] Falgout, R.D., Yang, U.M.: hypre: A library of high performance preconditioners. In: Sloot, P.M.A., Hoekstra, A.G., Tan, C.J.K., Dongarra, J.J. (eds.) *Computational Science — ICCS 2002, Lec. Notes Comp. Sc.*, pp. 632–641. Springer, Berlin, Heidelberg (2002). [https://doi.org/10.1007/3-540-47789-6\\_66](https://doi.org/10.1007/3-540-47789-6_66)
- [48] Balay, S., Abhyankar, S., Adams, M.F., Benson, S., Brown, J., Brune, P., Buschelman, K., Constantinescu, E.M., Dalcin, L., Dener, A., Eijkhout, V., Faibussowitsch, J., Gropp, W.D., Hapla, V., Isaac, T., Jolivet, P., Karpeev, D., Kaushik, D., Knepley, M.G., Kong, F., Kruger, S., May, D.A., McInnes, L.C., Mills, R.T., Mitchell, L., Munson, T., Roman, J.E., Rupp, K., Sanan, P., Sarich, J., Smith, B.F., Zampini, S., Zhang, H., Zhang, H., Zhang, J.: PETSc Web page. <https://petsc.org/> (2022). <https://petsc.org/>
- [49] Bonelle, J.: Compatible discrete operator schemes on polyhedral meshes for elliptic and Stokes equations. PhD thesis, Université Paris-Est – École des Ponts (2014)
- [50] Bonelle, J., Ern, A.: Analysis of compatible discrete operator schemes for the Stokes equations on polyhedral meshes. *IMA J. Numer. Anal.* **35**(4), 1672–1697 (2015) <https://doi.org/10.1093/imanum/dru051>
- [51] Milani, R.: Compatible Discrete Operator schemes for the unsteady, incompressible Navier-Stokes equations. PhD Thesis, Université Paris Est (2020)
- [52] Di Pietro, D.A., Lemaire, S.: An extension of the Crouzeix-Raviart space to general meshes with application to quasi-incompressible linear elasticity and Stokes flow. *Math. Comput.* **84**(291), 1–31 (2015) <https://doi.org/10.1090/S0025-5718-2014-02861-5>
- [53] Brezzi, F., Fortin, M.: *Mixed and Hybrid Finite Element Methods*. Springer series in computational mathematics. Springer, New York (1991). <https://doi.org/10.1007/978-1-4612-3172-1>

- [54] Golub, G.H., Greif, C.: On solving block-structured indefinite linear systems. *SIAM J. Sci. Comput.* **24**(6), 2076–2092 (2003) <https://doi.org/10.1137/S1064827500375096>
- [55] Kruse, C., Darrigrand, V., Tardieu, N., Arioli, M., Rüde, U.: Application of an iterative Golub-Kahan algorithm to structural mechanics problems with multi-point constraints. *Adv. Model. Simul. Eng. Sci.* **7**(1), 45 (2020) <https://doi.org/10.1186/s40323-020-00181-2>
- [56] Kruse, C., Sosonkina, M., Arioli, M., Tardieu, N., Rüde, U.: Parallel performance of an iterative solver based on the Golub-Kahan Bidiagonalization. In: Wyrzykowski, R., Deelman, E., Dongarra, J., Karczewski, K. (eds.) *Parallel Processing and Applied Mathematics. Lecture Notes in Computer Science*, pp. 104–116. Springer International Publishing, Cham (2020). [https://doi.org/10.1007/978-3-030-43229-4\\_10](https://doi.org/10.1007/978-3-030-43229-4_10)
- [57] Kruse, C., Sosonkina, M., Arioli, M., Tardieu, N., Rüde, U.: Parallel solution of saddle point systems with nested iterative solvers based on the Golub-Kahan bidiagonalization. *Concur. Comp.-Pract. E.* **33**(11), 5914 (2021) <https://doi.org/10.1002/cpe.5914>
- [58] Notay, Y.: A new multigrid approach for Stokes problems. *Numer. Math.* **132**, 51–84 (2016) <https://doi.org/10.1007/s00211-015-0710-0>
- [59] Giraud, L., Haidar, A., Saad, Y.: Sparse approximations of the Schur complement for parallel algebraic hybrid linear solvers in 3d. *Numer. Math. Theor. Meth. Appl.* **3**, 276–294 (2010) <https://doi.org/10.4208/nmtma.2010.33.2>
- [60] Brandt, A., Livne, O.E.: *Multigrid Techniques: 1984 Guide with Applications to Fluid Dynamics*, Revised Edition. SIAM, Philadelphia (2011). <https://doi.org/10.1137/1.9781611970753>
- [61] Briggs, W.L., Henson, V.E., McCormick, S.F.: *A Multigrid Tutorial*. SIAM, Philadelphia (2000). <https://doi.org/10.1137/1.9780898719505>
- [62] Trottenberg, U., Oosterlee, C.W., Schüller, A.: *Multigrid*. Elsevier, San Diego (2000)
- [63] Lin, P.T., Shadid, J.N., Tsuji, P.H.: In: Brummelen, H., Corsini, A., Perotto, S., Rozza, G. (eds.) *Krylov Smoothing for Fully-Coupled AMG Preconditioners for VMS Resistive MHD*, pp. 277–286. Springer, Cham (2020). [https://doi.org/10.1007/978-3-030-30705-9\\_24](https://doi.org/10.1007/978-3-030-30705-9_24)
- [64] Balay, S., Abhyankar, S., Adams, M.F., Brown, J., Brune, P., Buschelman, K., Dalcin, L., Dener, A., Eijkhout, V., Gropp, W.D., Kaushik, D., Knepley, M.G., May, D.A., McInnes, L.C., Mills, R.T., Munson, T., Rupp, K., Sanan, P., Smith, B.F., Zampini, S., Zhang, H., Zhang, H.: *PETSc users manual*. Technical Report



- [65] Falgout, R.D., Jones, J.E., Yang, U.M.: The design and implementation of hypre, a library of parallel high performance preconditioners. In: Bruaset, A.M., Tveito, A. (eds.) *Numerical Solution of Partial Differential Equations on Parallel Computers*, pp. 267–294. Springer, Berlin, Heidelberg (2006). [https://doi.org/10.1007/3-540-31619-1\\_8](https://doi.org/10.1007/3-540-31619-1_8)
- [66] De Sterck, H., Yang, U.M., Heys, J.J.: Reducing complexity in parallel algebraic multigrid preconditioners. *SIAM J. Matrix Anal. Appl.* **27**(4), 1019–1039 (2006) <https://doi.org/10.1137/040615729>
- [67] Kong, F., Wang, Y., Gaston, D.R., Permann, C.J., Slaughter, A.E., Lindsay, A.D., DeHart, M.D., Martineau, R.C.: A highly parallel multilevel Newton–Krylov–Schwarz method with subspace-based coarsening and partition-based balancing for the multigroup neutron transport equation on three-dimensional unstructured meshes. *SIAM J. Sci. Comput.* **42**(5), 193–220 (2020) <https://doi.org/10.1137/19M1249060>
- [68] Bank, R.E., Douglas, C.C.: Sharp estimates for multigrid rates of convergence with general smoothing and acceleration. *SIAM Journal on Numerical Analysis* **22**(4), 617–633 (1985) <https://doi.org/10.1137/0722038>
- [69] Burggraf, O.R.: Analytical and numerical studies of the structure of steady separated flows. *J. Fluid Mech.* **24**(1), 113–151 (1966) <https://doi.org/10.1017/S0022112066000545>
- [70] Dumitras, A., Kruse, C., Ruede, U.: Generalized Golub-Kahan bidiagonalization for nonsymmetric saddle point systems (2023). <https://doi.org/10.48550/arXiv.2310.06952>
- [71] Benzi, M.: A generalization of the Hermitian and skew-Hermitian splitting iteration. *SIAM J. Matrix Anal. Appl.* **31**(2), 360–374 (2009) <https://doi.org/10.1137/080723181>
- [72] Angot, P., Caltagirone, J.-P., Fabrie, P.: A fast vector penalty-projection method for incompressible non-homogeneous or multiphase Navier–Stokes problems. *Appl. Math. Lett.* **25**(11), 1681–1688 (2012) <https://doi.org/10.1016/j.aml.2012.01.037>

## Appendix A `code_saturne` options

Here, we provide parameter options to use a saddle-point problems solvers and K-cycle AMG in the context of `code_saturne`. Note that for the K-cycle, we rely on the

Option	Value	Result
CS_NSKEY_SLES_STRATEGY	"gkb" or "gkb_saturne" "gkb_petsc" "uzawa_al" "notay"	GKB method via <code>code_saturne</code> in house GKB method via PETSc ALU method Algebraic transformation
CS_NSKEY_GD_SCALE_COEF	" $\gamma$ "	Set an augmented parameter $\gamma \geq 0$
CS_NSKEY_NL_ALGO_RTOL	"tol"	Set the relative tolerance for Picard's iterations to $\text{tol} > 0$
CS_NSKEY_MAX_NL_ALGO_ITER	"max_it"	Set the maximal number of Picard's iterations to $\text{max\_it} \in \mathbb{N}$
CS_NSKEY_IL_ALGO_RTOL	"tol"	Set the relative tolerance for outer iterations to $\text{tol} > 0$
CS_NSKEY_MAX_IL_ALGO_ITER	"max_it"	Set the maximal number of outer iterations to $\text{max\_it} \in \mathbb{N}$
CS_EQKEY_SOLVER_FAMILY	"cs" "petsc"	Use a linear solver from <code>code_saturne</code> in house Use a linear solver from PETSc library
CS_EQKEY_ITSOL	"fcg" "fgmres" "mumps"	FCG solver FGMRES solver Direct solver (through MUMPS or PETSc)
CS_EQKEY_ITSOL_EPS	"tol"	Set the tolerance for iterative solvers to $\text{tol} > 0$
CS_EQKEY_ITSOL_MAX_ITER	"max_it"	Set the maximal number of iterative solvers to $\text{max\_it} \in \mathbb{N}$

implementation in `code_saturne`. Then,

`cs_multigrid_set_solver_options()` and `cs_multigrid_set_coarsening_options()`

allow us to set multigrid parameters for smoothing and coarsening, respectively. For more details of the usage of `code_saturne`, we refer to the official document<sup>2</sup>

## Appendix B PETSc options

In this appendix, we present the example sets of PETSc options that we used for our numerical tests.

### B.1 Hypre BoomerAMG

Option	Value	Result
-pc_type	"hypre"	HypreBoomerAMG
-pc_hypre_type	"boomeramg"	as a preconditioner
-pc_hypre_boomeramg_coarsen_type	"Falgout" ("HMIS" or "PMIS")	The classical coarsening (with lower complexity)
-pc_hypre_boomeramg_agg_n1	"0"	No aggressive coarsening
-pc_hypre_boomeramg_relax_type_down -pc_hypre_boomeramg_relax_type_up	"symmetric-SOR/Jacobi"	SGS smoothing

<sup>2</sup><https://www.code-saturne.org/documentation/7.1/doxygen/src/index.html>

For more details of PETSc options and the full list of options, we refer to the main Hydre website<sup>3</sup> and MOOSE website<sup>4</sup>.

## B.2 GAMG

Option	Value	Result
<code>-pc.type</code>	"gamg"	GAMG
<code>-pc_gamg.type</code>	"agg"	as a preconditioner
<code>-pc_gamg_agg_nsmooths</code>	"1"	One smoothing step to use with smoothed aggregation
<code>-pc_gamg_sym_graph</code>	"true"(if a matrix is non-symmetric)	Symmetrize the graph
<code>-mg_levels_ksp.type</code>	"richardson"	SGS smoothing
<code>-mg_levels_pc.type</code>	"sor"	

Note that GAMG is basically built with the smoothed aggregation scheme so that it is more applicable for symmetric systems. In practice, by turning on `-pc_gamg_sym_graph`, it is required to symmetrize the graph for non-symmetric systems. Moreover, to consider unsmoothed aggregation coarsening, we should set `-pc_gamg_agg_nsmooths=0`.

With GAMG, it is able to use Krylov smoothers. For example, we can employ the preconditioned GMRES for each smoothing step as following.

Option	Value	Result
<code>-mg_levels_ksp.type</code>	"richardson"	Krylov smoothing by GMRES
<code>-mg_levels_pc.type</code>	"bjacobi"	
<code>-mg_coarse_pc.type</code>	"tfs"	
<code>-mg_levels_pc_bjacobi_blocks</code>	"1"	
<code>-mg_levels_sub_ksp.type</code>	"gmres"	
<code>-mg_levels_sub_ksp_max_it</code>	"10"	Maximum 10 GMRES smoothing steps
<code>-mg_levels_sub_ksp_rtol</code>	"1e-3"	Set a tolerance of the GMRES smoother to $10^{-3}$
<code>-mg_levels_sub_pc.type</code>	"sor"	SGS preconditioning for the GMRES smoother
<code>-mg_levels_sub_pc.type</code>	"ilu"	ILU(0) preconditioning for the GMRES smoother
<code>-mg_levels_sub_pc_factor_levels</code>	"0"	

## B.3 HMG

Using HMG, we can combine the coarsening schemes from Hydre BoomerAMG and the smoothing operators from GAMG in PETSc as following.

Option	Value	Result
<code>-pc.type</code>	"hmg"	HMG preconditioner with the classical coarsening
<code>-hmg_inner_pc.type</code>	"hydre"	

To set up smoothing process, it follows the same way of GAMG.

<sup>3</sup><https://hydre.readthedocs.io/en/latest/index.html>

<sup>4</sup>[https://mooseframework.inl.gov/releases/moose/v1.0.0/application\\_development/hybre.html](https://mooseframework.inl.gov/releases/moose/v1.0.0/application_development/hybre.html)