Efficient and Robust Monolithic Finite Element Multilevel Krylov Subspace Solvers for the Solution of Stationary Incompressible Navier-Stokes Equations.

Dissertation
zur Erlangung des Grades eines
Doktors der Naturwissenschaften
(Dr. rer. nat.)

Der Fakultät für Mathematik der
Technischen Universität Dortmund
vorgelegt von

Absaar Ul Jabbar

im September 2018

Dissertation

Efficient and Robust Monolithic Finite Element Multilevel Krylov Subspace Solvers for the Solution of Stationary Incompressible Navier-Stokes Equations.

Fakultät für Mathematik
Technische Universität Dortmund

Erstgutachter: Prof. Dr. Stefan Turek

Zweitgutachter: Prof. Dr. Heribert Blum

Tag der mündlichen Prüfung: 13.12.2018

Abstract

Multigrid methods belong to the best-known methods for solving linear systems arising from the discretization of elliptic partial differential equations. The main attraction of multigrid methods is that they have an asymptotically mesh-independent convergence behavior. Multigrid with Vanka (or local multilevel pressure Schur complement method) as smoother have been frequently used for the construction of very efficient coupled monolithic solvers for the solution of the stationary incompressible Navier-Stokes equations in 2D and 3D. However, due to its innate Gauß-Seidel/Jacobi character, Vanka has a strong influence of the underlying mesh, and therefore, coupled multigrid solvers with Vanka smoothing very frequently face convergence issues on meshes with high aspect ratios. Moreover, even on very nice regular grids, these solvers may fail when the anisotropies are introduced from the differential operator.

In this thesis, we develop a new class of robust and efficient monolithic finite element multilevel Krylov subspace methods (MLKM) for the solution of the stationary incompressible Navier-Stokes equations as an alternative to the coupled multigrid-based solvers. Different from multigrid, the MLKM utilizes a Krylov method as the basis in the error reduction process. The solver is based on the multilevel projection-based method of Erlangga and Nabben, which accelerates the convergence of the Krylov subspace methods by shifting the small eigenvalues of the system matrix, responsible for the slow convergence of the Krylov iteration, to the largest eigenvalue.

Before embarking on the Navier-Stokes equations, we first test our implementation of the MLKM solver by solving scalar model problems, namely the convection-diffusion problem and the anisotropic diffusion problem. We validate the method by solving several standard benchmark problems. Next, we present the numerical results for the solution of the incompressible Navier-Stokes equations in two dimensions. The results show that the MLKM solvers produce asymptotically mesh-size independent, as well as Reynolds number independent convergence rates, for a moderate range of Reynolds numbers. Moreover, numerical simulations also show that the coupled MLKM solvers can handle (both mesh and operator based) anisotropies better than the coupled multigrid solvers.

Key words: Monolithic multilevel methods, Krylov subspace, GMRES, FEM, Navier-Stokes equations, saddle point problems

To my parents,
my wife,
and my children, Fatimah and Yahya

Acknowledgments

In the name of Allah, the most Merciful, the most Beneficent.

All the praises to Almighty Allah for his countless blessings on me, for providing me with the opportunity, strength, and perseverance to accomplish this work successfully.

Firstly, I would like to express my sincere gratitude to my advisor Prof. Stefan Turek for his continuous support, guidance, encouragement, and the trust he always showed in me throughout the duration of this thesis. Despite leading a large research group and his commitments being a Dean of the faculty, he was always there to help me solve my problems, both academic and social. I am truly obliged to him and would like to take this opportunity to express it formally. I am also thankful to Prof. Dr. Heribert Blum for showing his willingness to review my thesis.

I would like to express my appreciation to all my colleagues at the Institut für Angewandte Mathematik und Numerik LS III, TU Dortmund, for providing me the friendly environment during my stay here. A special thanks goes to our old colleague Dr. Michael Köster, who paved the way in the initial phase of my PhD by helping me understand and implement the code. I also thank Peter Zajac for providing the wonderful support related to the FEATFLOW software, and for being patient to my queries. I am also grateful to Dr. Ouazzi and Dr. Damanik for their fruitful discussions from time to time. Gratitude is also due to Dr. Becker and Sven Buijssen for their IT support, and Frau Drees and Frau Lamprecht for their administrative support. I would like to thank Dr. Safi and Omid Ahmadi for bearing me in the room.

I gratefully acknowledge the Higher Education Commission (HEC) of Pakistan and Faculty of Mathematics, TU Dortmund, for supporting me financially that made my PhD work possible.

Finally, I can never ever thank enough to my family members for all they have done for me. Especially my parents and my loving wife whose patience, love, support and prayers enabled me to achieve this success.

Absaar Ul Jabbar

Contents

1 Introduction

Everything on this earth is either a fluid or interacting with a fluid. The atmosphere that surrounds this planet and oceans that constitute 70 % of this earth are fluids. Fluid flows are vital for the existence of the life that we live. Our body gets oxygen due to the fluid flow and blood flow provides important nutrients to all parts of the body. Ocean waves and wind fields regulate the global climate by spreading the uneven solar radiations received by the Earth surface. Similarly, fluid flows play a pivotal role in many industries, for instance, power generation, oil and gas exploration, food processing, and chemical manufacturing to mention a few. This ubiquitous nature of the fluids and the importance of fluid flows in our life has made the study of fluid flows a fundamental and evolving field that spans major areas in science and technology.

The Navier-Stokes equations (NSE) are at the heart of studying fluid flows, and they are used to describe the physics of various flows of scientific and engineering interest. They are used by:

- meteorologists to predict the weather conditions
- aerodynamics engineers to design aircraft that have low resistance and high lift forces
- civil engineers to build safe hydraulic structures such as dams
- chemical engineers to design and optimize industrial plants and equipment
- mechanical engineers to design pumps, turbines, and HVACR (heating, ventilation, air conditioning, and refrigeration) systems

Unfortunately, except for the very simple flow scenarios, the exact analytical solution of the Navier-Stokes equations is in general not known. The wide spectrum of applications of the Navier-Stokes equations attracted the attention of scientists, engineers, mathematicians, and computer scientists during the past few decades, to develop numerical methods for the approximate solution of the Navier-Stokes equations. This interest led to the rapid development of a wide range of numerical and algorithmic tools for solving complex fluid flow problems, which emerged as a new scientific field termed as *computational fluid dynamics* (CFD). CFD encompass numerical techniques for the solution of many different types of fluid flows. This thesis is only concerned with the numerical solution of the (steady-state) Navier–Stokes equations governing the flow of a Newtonian, incompressible viscous fluid, which may be a very small subset of the CFD. However, the numerical techniques presented in this thesis can be easily extended to other more complicated fluid flows, for instance non-isothermal, non-Newtonian, and/or turbulent fluid flows.

Numerical solution of NSE involves the application of discretization techniques such as finite element methods (FEM), finite volume methods (FVM), or finite difference methods (FDM) that transform the infinite dimensional partial differential equations into a finite dimensional algebraic system of equations. In this thesis, we use FEM discretizations which can handle complex geometries (mostly encountered in practice) more efficiently than FVM and FDM. LBB compatible mixed finite elements are employed to ensure the well-posedness of the algebraic system of equations.

Since NSE are nonlinear, the resulting algebraic system of equations coming from FEM discretization is also nonlinear. Suitable linearization techniques such as Picard iteration or Newton iteration are applied to the nonlinear discrete system. The linearization process results in a linear system of saddle point type, with a large zero block on the main diagonal due to the absence of pressure in the continuity equation of the incompressible flows. The linear saddle point systems coming from the FEM discretization are sparse and in practice involve hundreds of millions of unknowns. Solving such linear systems is the major bottleneck of the numerical solution process since solution and storage of these saddle point systems constitutes most of the computational resources (CPU time and memory) of the whole solution process.

The solution of saddle point problems is pivotal in the design of solution algorithms for Navier-Stokes equations. These solution algorithms can be broadly categorized into two groups, namely *segregated* or *coupled* methods, depending on the way the saddle point linear system is solved. Segregated algorithms decouple the velocity and pressure variables, and thereby, solve the reduced systems for each variable separately. Coupled algorithms respect the natural coupling between the solution variables and solve them simultaneously. Both classes of solvers have their own problem-dependent strengths and weaknesses and hence have specific application areas. Coupled solvers are suitable for flows involving low Reynolds numbers resp. high viscosity parameters, however, they are very expensive for non-stationary high Reynolds number flows. On the other hand, segregated solvers are suitable for such non-stationary flows, whereas, they face convergence issues for stationary and non-stationary flows involving high viscosities. The focus of this thesis is only on coupled solvers for stationary incompressible Navier-Stokes equations.

1.1 Open problem

Coupled saddle point linear systems arising from the discretization of the Navier-Stokes equations are nonsymmetric, indefinite and ill-conditioned, and solving such systems pose a great challenge for the development of efficient and robust monolithic numerical solvers. Direct (sparse) linear solvers give the exact solution to the linear system and are very stable. However, their intrinsic computational costs and in particular memory requirements are very high for the big linear systems originating from the discretization of NSE (especially in the 3D case), and they become impractical for use as standalone solvers. For such highly sparse and large coupled linear systems, iterative linear solvers are a better choice because of their low memory requirements. Moreover, in numerical algorithms for nonlinear Navier-Stokes equations, the accuracy requirements on each inner linear solve are low, therefore, iterative methods are more suitable since they can be stopped as soon as the desired accuracy is reached.

Preconditioned Krylov subspace methods or multigrid methods are mostly employed as linear solvers in coupled nonlinear iterative solvers. A coupled geometric multigrid with *local pressure Schur complement smoothing* (a generalization of Vanka smoother) is a very efficient state of the art solver that produces mesh size and Reynolds number independent convergence rates, for stationary flow problems[132]. This solver is currently the standard coupled solver in the FEATFLOW software. The Vanka smoother is a strongly coupled iterative relaxation technique, originally presented by Vanka in 1986 [138] to solve the Navier-Stokes equations using finite difference method. The concept of Vanka smoothing is simple, the local saddle point subsystem associated with each mesh cell (or a cluster of cells) is extracted and solved exactly by treating all the variables simultaneously. The calculated local degrees of freedom are then updated by Jacobi or Gauss-Seidel relaxation iterations.

From scalar elliptic problems such as diffusion and convection-diffusion problems, it is well known that multigrid with pointwise Jacobi smoothing or pointwise Gauss-Seidel smoothing suffers serious convergence issues for the highly convective flows or highly distorted meshes (meshes with large aspect ratio elements or with a large difference in neighboring elements size). Multigrid with Vanka smoother, due to its implicit Jacobi/Gauß-Seidel nature, also suffers similar convergence issues on such grids.

For scalar problems, the convergence issues of MG solvers for highly convective or highly anisotropic situations can be mitigated by employing incomplete LU factorization as a smoother to the multigrid solver. However, in the case of coupled NSE, the straightforward application of ILU factorization is not possible; it may suffer breakdown due to the zero pivots, or the resulting factorization is not stable. Although various reordering, dropping, scaling and pivoting strategies have been proposed in the literature to avoid breakdowns and to produce stable LU factors, however, still the resulting factorization of nonsymmetric, indefinite and ill-conditioned saddle point problems is often not of good quality, and produce poor convergence rates. Therefore, designing a robust as well as an efficient coupled numerical solver for the solution of the incompressible Navier-Stokes equations, which produces mesh-size, mesh-shape, and Reynolds number independent convergence behavior, is an open challenge for the scientific community.

1.2 Thesis contribution

In this thesis, we propose a new coupled multilevel FEM solver for the monolithic solution of stationary incompressible Navier-Stokes equations, which can be considered as an alternative to the existing coupled multigrid method. The solver is based on the multilevel Krylov subspace method (MLKM) proposed by Nabben and Erlangga in [54] to solve the scalar model problems. To the best of our knowledge, no one has so far used such multilevel Krylov subspace solution techniques for the solution of Navier-Stokes equations in a coupled way.

The main idea of MLKM method is to apply a projection-type shift preconditioner which shifts the small eigenvalues of the system matrix, responsible for the slow convergence of the Krylov subspace methods, to the largest eigenvalue. This clustering of the eigenvalues away from zero accelerates the convergence of Krylov subspace solver. The multilevel Krylov subspace method contains the ingredients of Krylov subspace methods as well as the multigrid method; grid hierarchies similar to multigrid are used for the recursive application of shift preconditioning and the solution is extracted from the Krylov subspaces. As a consequence, it inherits properties of both the solver classes, i.e., robustness from the Krylov subspace solvers and level independent convergence rates from the multigrid methods.

Shift preconditioning alone does not produce acceptable convergence rates in many situations and is often applied in combination with the “traditional preconditioners” to further improve the convergence behavior of the Krylov subspace solvers. MLKM algorithm in [54] requires explicit calculation of the inverse of the preconditioner matrix $\mathbf{M}^{-1}$ at every mesh level in the initialization phase of the algorithm. Such matrices are in general dense, and working with such dense matrices is very expensive and not advisable. This restricts the user to Jacobi preconditioning only, which is not very effective in most practical problems. In the particular case of incompressible NSE, point-wise Jacobi preconditioning cannot be applied due to the zeros on the diagonal.

Our implementation of MLKM algorithm in the FEATFLOW software is different from the one given in [54] and allows the use of any iterative method as a preconditioner to the Krylov subspace solver. In this thesis, we have successfully used the traditional preconditioners such as Jacobi, Gauß-Seidel, and ILU to the MLKM method for solving scalar convection-diffusion and anisotropic diffusion problems.

Moreover, this flexibility in preconditioning allows extending the MLKM method to solve coupled system of equations of the saddle point type such as the Navier-Stokes equations. We have used local pressure Schur complement or Vanka preconditioning to MLKM method, which has resulted into a new class of efficient and robust monolithic multilevel numerical solution techniques for saddle point type problems. The coupled MLKM solver produces mesh-size and Reynolds number independent convergence rates; moreover, it is more robust than coupled MG solvers towards handling the anisotropic meshes and solving the flows involving higher Reynolds numbers.

1.3 Thesis Outline

Below we present an outline of the thesis, which gives a brief overview of all the chapters to follow.

Chapter 2 gives a basic introduction to the finite element method, which is a spatial discretization method used in this thesis. Section 2.1 explains the working principle of the finite element method, thereby, showing how the method converts the infinite dimensional partial differential equations into a finite dimensional linear system of equations. Section 2.2 discusses the finite element basis functions (or shape functions) and their properties; the section also includes the description of bilinear (Q_1) and biquadratic (Q_2) quadrilateral finite elements and the construction of their shape functions. The chapter ends with a discussion of FEM error estimates in section 2.3.

Chapter 3 reviews the most popular numerical methods for the solution of sparse linear systems arising from the FEM discretization of PDEs. Section 3.1 mentions some popular sparse direct methods and discusses their strengths and limitations. Iterative methods for sparse linear systems are described in section 3.2. Basic iterative methods such as Jacobi, Gauß-Seidel, and SOR methods are reviewed in section 3.2.1. In section 3.2.2, working principle of Krylov subspace methods is explained, and some famous Krylov subspace methods such as conjugate gradient, CGNR, and GMRES are presented as representative Krylov subspace methods. Multigrid and its components are discussed in section 3.2.3. Section 3.3 concludes the chapter with a discussion of node renumbering strategies being used in this thesis.

Chapter 4 is mainly devoted to the discussion of multilevel Krylov space method (MLKM), which is a combination of ideas from Krylov subspace solvers and multigrid methods. The chapter starts with the discussion of an important concept in the context of Krylov subspace solvers called preconditioning, with the traditional preconditioners being discussed in section 4.1 and the eigenvalue distribution preconditioners in section 4.2. After that we review in detail the construction of the MLKM solver proposed by Nabben and Erlangga in section 4.3, followed by our implementation of MLKM solver in the context of FEATFLOW solver in section 4.4 and discuss how it differs from the MLKM algorithm of Nabben and Erlangga. At the end of the chapter, section 4.5 briefly compares the MLKM solver with the multigrid solvers and explains how both the solvers are different from each other.

Chapter 5 deals the scalar convection diffusion problem. FEM formulation of the problem is presented in section 5.1, with the stabilization techniques for the convective term discussed in section 5.1.1. Numerical results of the MLKM solver for the solution of convection-diffusion problem, and its comparison with the multigrid solvers are presented in section 5.2.

Chapter 6 presents the numerical results for the anisotropic diffusion problem, with section 6.1 presenting the results for the operator-based anisotropy and section 6.1.1 discussing the results for the grid-based anisotropy.

Chapter 7 extends the work from the previous chapters of solving the scalar PDEs using FEM/MLKM solver and presents a new monolithic FEM/MLKM solver for the numerical solution of stationary incompressible Navier-Stokes equations in a fully coupled way. Section 7.1 introduces the Navier-Stokes problem and discusses the boundary conditions for the problem. Weak formulation of Navier-Stokes equations and their FEM discretization are discussed in section 7.2 and 7.3 respectively. In section 7.4, we consider the LBB condition for the well posedness of the weak formulation of the Navier-Stokes equations, and in section 7.5, we discuss our choice of LBB-stable mixed finite element pairs. Section 7.6 is dedicated to the construction of coupled multilevel Krylov subspace solver, and we discuss its various components; we explain the linearization of nonlinear equation using Newton method (section 7.6.1) and fixed point iteration (section 7.6.2), functioning of the local pressure Schur complement preconditioner (section 7.6.3), and the calculation of optimal damping parameters (section 7.6.4).

Chapter 8 presents numerical results for the solution of steady incompressible Navier-Stokes equations, for various characteristic flow scenarios. In section 8.1, we validate our coupled MLKM solver by solving standard benchmark problems using conforming as well as non-conforming finite elements, on structured and unstructured meshes. In section 8.2, we study the influence of different parameters of coupled MLKM solver on its performance. Finally, in section 8.3, we do the performance comparison of coupled MLKM solver with the existing monolithic solvers available in the FEATFLOW software with respect to the shape of the mesh and the size of Reynolds number.

Chapter 9 is the concluding chapter that summarizes the work being presented in this thesis and provides directions for the possible future research initiatives.

2 Introduction to Finite Element Method

In this chapter, we give the introduction of the finite element method (FEM), used for the spatial discretization of model problems throughout this thesis. FEM is a powerful numerical technique used for the approximate solution of partial differential equations. The method originated from solving the elasticity and structural mechanics problems in aerospace engineering, but due to its robustness, flexibility, and accuracy, its use quickly spread to other disciplines of engineering and applied sciences. The method, in contrast to finite differences, can handle problems on complex geometries really well. Moreover, the method provides discretization error estimates at reasonable cost, which allows adaptive mesh refinements to compute the solution to the desired accuracy optimally [10, 9, 142].

FEM converts the original partial differential equation representing the physical system into an integral form, called the *variational or weak form*, defined over the problem domain. This domain is then subdivided into a number of geometrically simple, smaller pieces called *finite elements.* Simple piecewise polynomial functions, called *trial functions* are defined on these finite elements, and the solution of the variational integral is approximated by the linear combination of these finite set of trial functions. This process converts the infinite dimensional PDE into a finite dimensional algebraic system, just like finite difference method (FDM). However, in FDM the solution is only known at the discrete points, whereas in FEM the solution is known throughout the domain as a piecewise polynomial function.

2.1 Fundamentals of the finite element method

To explain the basics of finite element solution procedure, we consider the following linear model problem

$$\begin{aligned} \mathcal{L}u &= f \quad && \text{in} \quad && \Omega, && (2.1a)\\ u &= g && \text{on} && \Gamma_D, && (2.1b)\\ \nabla u.\mathbf{n} &= \beta && \text{on} && \Gamma_N, && (2.1c) \end{aligned}$$

where $\mathcal{L}$ is the second order elliptic Laplacian operator, i.e., $\mathcal{L} = -\Delta$ and $\Omega \subset \mathbb{R}^d$, $d = 1, 2, 3$ with boundary $\Gamma = \Gamma_D \cup \Gamma_N$. Dirichlet boundary condition is prescribed on Γ_D and Neumann boundary condition on Γ_N part of Γ. Since the operator $\mathcal{L}$ is second order, a classical solution $u : \Omega \mapsto \mathbb{R}$ that satisfies (2.1) is required to be twice continuously differentiable, that is, $u \in C^2(\Omega)$ [52, 114].

2.1.1 Weak formulation

Finite element method, unlike finite difference methods, does not approximate the partial differential equation directly, rather it formulates the original PDE into a more suitable integral formulation known as *variational formulation.* In what follows, we use the *method of weighted residuals* to construct the variational form of (2.1). To this end, we multiply the residual of equation (2.1a) with a *test* or *weighting function* v and integrate over Ω to obtain

$$\int_\Omega (\mathcal{L}[u] - f) v d\Omega = 0. \tag{2.2}$$

Generally, the integral of the product of two functions represents the so-called L^2 inner product which induces the L^2 norm $\|.\|_0$

$$(v, w) := \int_\Omega vw d\Omega, \qquad \|v\|_0 := \sqrt{(v, v)}. \tag{2.3}$$

A function v is called *square integrable* if the inner product (v, v) exists. If u is a solution to the original PDE (2.1), it is also a solution to integral form (2.2) for all square integrable functions v [81]. We may write (2.2) in inner product form as

$$(v, \mathcal{L}[u] - f) \ = \ 0, \qquad \forall v \in L^2(\Omega). \tag{2.4}$$

Equation (2.4) is called the *variational form* of problem (2.1). In FEM procedure, the strong continuity requirement on the solution function u is relaxed by applying the integration by parts on the second derivative terms through the application of Green's theorem

$$\int_\Omega -v\nabla.\nabla u \ d\Omega = \int_\Omega \nabla v.\nabla u \ d\Omega - \int_\Gamma v\nabla u.\mathbf{n} \ ds. \tag{2.5}$$

By doing so we shift one derivative from u to v. Thus in the resulting formulation, v has more continuity requirement than the functions in $L^2(\Omega)$, whereas, the strong continuity requirement $C^2(\Omega)$ of u is weakened. For this reason, the new formulation is called the *weak formulation*, and its solution is called *weak solution.* Since the highest derivatives involved are now of first order, it suffices for u and v to be elements of the *Sobolev space* $\mathcal{H}^1$, which contains all the square integrable functions whose first weak derivatives are also square integrable,

$$\mathcal{H}^1(\Omega) = \Big\{ w \in L^2(\Omega), \frac{\partial w}{\partial x_i} \in L^2(\Omega) \qquad i = 1, ..., d \Big\}. \tag{2.6}$$

The treatment of the boundary integral resulting from the application of Green's theorem (2.5) requires greater attention. Since u is a member of $\mathcal{H}^1$ and it must also satisfy (2.1b), therefore we take $u \in \mathcal{H}^1_E$, with the subscript E showing that solution u satisfies the essential Dirichlet boundary condition. Weighting function v should not vary at the boundary where the solution is specified, which means that it should vanish at the Dirichlet boundary, and hence belongs to space

$$\mathcal{H}_0^1 := \Big\{ w \in \mathcal{H}^1 : w|_{\Gamma_D} = 0 \Big\}. \tag{2.7}$$

At Neumann boundary, the derivative value of the solution function u is specified. When u is not prescribed at the boundary, the test function need not vanish there, i.e., $v|_{\Gamma_N} \neq 0$. Since v is zero at Dirichlet boundary, and nonzero at Neumann boundary, the boundary integral in Green's theorem (2.5) can be restricted to Γ_N only

$$\int\limits_{\Gamma} v\nabla u.\mathbf{n}ds \;=\; \underbrace{\int\limits_{\Gamma_D} v\nabla u.\mathbf{n}ds}_{=\,0} + \int\limits_{\Gamma_N} v\nabla u.\mathbf{n}ds \;=\; \int\limits_{\Gamma_N} v\beta =: (v,\beta). \tag{2.8}$$

Application of the Green's theorem on (2.4) and subsequent use of (2.8), results in the following *weak form*
Find $u \in \mathcal{H}_E^1$ *such that*

$$a(v,u) = b(v) \qquad \forall v \in \mathcal{H}_0^1, \tag{2.9}$$

where

$$a(v,u) = \int_{\Omega} \nabla v.\nabla u \; d\Omega$$

is called the *bilinear form*, and

$$b(v) := (v,f) + (v,\beta),$$

is called the *linear form*. Solution space $\mathcal{H}_E^1$ is an affine space, and instead, we can write $\mathcal{H}_E^1 = u_g \oplus \mathcal{H}_0^1$, where $u_g \in \mathcal{H}^1$ such that $u_g = g$ on Γ_D. Clearly, for pure homogeneous Dirichlet boundary condition ($\Gamma = \Gamma_D$), we have $\mathcal{H}_E^1 = \mathcal{H}_0^1$, and the weak form in this case becomes
Find $u \in \mathcal{H}_0^1$ *such that*

$$a(v,u) = b(v) \qquad \forall v \in \mathcal{H}_0^1. \tag{2.10}$$

In finite element method, the infinite dimensional spaces $\mathcal{H}_E^1$ and $\mathcal{H}_0^1$ are replaced by their convenient finite dimensional subsets $\mathcal{S}_E^h$ and $\mathcal{S}_0^h$, respectively. To this end, we subdivide the computational domain into a regular partition called *triangulation*, containing N_{el} non-overlapping, and nonempty convex subdomains Ω_k, each with piecewise smooth boundary Γ_k, such that

$$\overline{\Omega} = \bigcup_{k=1}^{N_{el}} \overline{\Omega}_k, \qquad \Omega_k \cap \Omega_l = \emptyset \quad for \quad k \neq l.$$

The finite element spaces are characterized by the span of basis functions defined on these subdomains. We consider the approximations of the form

$$u(x) \approx u_h = u_g^h + U_h = u_g^h + \sum_{j=1}^{N} u_j \phi_j, \tag{2.11a}$$

$$v(x) \approx v_h = \sum_{i=1}^{N} v_i \psi_i. \tag{2.11b}$$

where u_g^h is the discrete version of u_g. The approximate u_h is called *trial function*, and belongs to the finite-dimensional function space $\mathcal{S}_E^h \subset \mathcal{H}_E^1$, called *trial space*. Similarly, v_h is termed as *test function* belonging to the finite-dimensional *test space* $\mathcal{S}_0^h \subset \mathcal{H}_0^1$. The basis functions ϕ_j and ψ_i are pre-selected piecewise polynomials that vanish on Γ_D.

Remark:
Dirichlet boundary conditions are essentially built into the trial and test spaces, therefore, they are also called *essential* boundary conditions. Neumann boundary conditions are also called *natural* boundary conditions, because they are automatically included in the weak form, and are not explicitly imposed in trial and test spaces.
Inserting (2.11) into equation (2.9), we have the discretized form of the weak formulation

$$a(v_h, U_h) = b(v_h) - a(v_h, g^h) \qquad \forall v_h \in \mathcal{S}_0^h. \tag{2.12}$$

See [127] for more details on the treatment of non-homogeneous Dirichlet boundary conditions. In Galerkin finite elements (that we have used in this thesis), the trial and test spaces are taken to be the same, i.e., $\phi_i = \psi_i$. Since (2.12) holds for all functions $v_h \in \mathcal{S}_0^h$, it also holds for the basis functions ϕ_i. This yields the linear algebraic system

$$\sum_{j=1}^{N} a(\phi_i, \phi_j) u_j = b(\phi_i) - a(\phi_i, g^h), \qquad i = 1, ..., N, \tag{2.13}$$

for the unknowns $u_1, ..., u_N$. In practical finite element implementation, the equation (2.13) is restricted to the element level, and the bilinear and linear forms are evaluated elementwise using the local basis functions $\phi^{(k)}$, also called *shape functions*. The system matrix and the right-hand side vector are assembled by the summation of contributions from each element

$$\sum_{k=1}^{N_{el}} a_k(\phi_i^{(k)}, \phi_j^{(k)}) u_j = b_k(\phi_i^{(k)}) - a_k(\phi_i^{(k)}, g^h), \qquad i, j = 1, ..., n \tag{2.14}$$

where N_{el} is the number of elements and n is the number of local degrees of freedom in each element. For an exhaustive discussion of finite element methods, we refer the interested readers to the literature [21, 34, 44, 81, 57, 103, 102].

2.2 Finite element shape functions

Finite element formulation hinges critically on shape functions for its working. These local basis functions are used with the computed nodal solution values u_j to interpolate the approximate solution u_h inside the element Ω_k

$$u_h(\mathbf{x}) = \sum_{j=1}^{n} u_j \phi_j^{(k)}(\mathbf{x}) \qquad \forall \mathbf{x} \in \Omega_k, \tag{2.15}$$

where n denotes the number of local degrees of freedom for a single element Ω_k. Summation of the local interpolation functions over the whole domain yields the global approximation of u_h, given by

$$u_h(\mathbf{x}) = \sum_{j=1}^{N} u_j \phi_j(\mathbf{x}) \qquad \forall \mathbf{x} \in \Omega. \tag{2.16}$$

where N denotes the total number of degrees of freedom. As we shall see shortly, the construction of a shape function on each element involves geometric information from that element only. This provides FEM with the flexibility to incorporate more accurate higher order interpolation functions locally or to add more refined meshes in the regions involving steep solution gradients, such as boundary layers. The shape functions are required to possess the following general properties, so that the resulting finite element method provides accurate approximations at reasonable costs [88]:

1. ***Interpolation property:*** The shape function $\phi_j^{(k)}$ is one at the node j and zero at all other nodes in the domain

$$\phi_j(\mathbf{x}_i) = \delta_{ij} = \begin{cases} 1, & \text{if} j = i, \\ 0, & \text{if } j \neq i. \end{cases} \tag{2.17}$$

   where δ_{ij} is the Kronecker delta. This property makes it possible to use the expansion coefficients as nodal solution values in relation (2.15).

2. ***Constant sum property:*** The sum of all the local basis functions $\phi_j^{(k)}$ should be equal to one on each element Ω_k. This property ensures the correct representation of constant functions through the shape functions.

$$\sum_{j=1}^{N} \phi_j(\mathbf{x}) = 1. \tag{2.18}$$

3. ***Conservation property:*** The sum of derivatives of all the shape functions should vanish at any point in the element.

$$\sum_{j=1}^{N} \nabla\phi_j(\mathbf{x}) = 0. \tag{2.19}$$

4. ***Local support property:*** For computational efficiency, it is required that a local shape function $\phi_j^{(k)}$ vanishes over any element boundary (an edge in 2D, a surface in 3D) that does not contain the node j. This requirement results in global basis functions with compact local support, which consequently produce a computationally desirable sparse linear system.

Next, we show with examples how these local basis functions are built from simple polynomials defined piecewise over the finite elements. In particular, we employ Lagrangian polynomials, since they provide a systematic way of generating the shape functions of any order. As we deal with two-dimensional problems in this thesis and employ quadrilateral finite elements for the discretization, we find it helpful to present the shape functions for Q_1 and Q_2 quadrilateral finite elements.

2.2.1 Bilinear element (Q_1)

Bilinear element Q_1 consists of four nodes at the four corners of the quadrilateral as shown in the Figure 2.1. Instead of expressing the shape functions directly on arbitrary physical elements Ω_k, they are typically defined on some reference element with simple geometry. Let $\Omega_{ref} = [-1,1]^2$ be the reference element with a local coordinate system (ξ, η) introduced at its center as depicted in Figure 2.1 (right). Interpolation polynomials on the reference quadrilateral are constructed by the tensor product of one dimensional linear Lagrange polynomials. One dimensional linear shape functions are given by [103]

$$\hat{\vartheta}_1 = \frac{1-\xi}{2}, \quad \hat{\vartheta}_2 = \frac{1+\xi}{2}, \quad -1 \leq \xi \leq 1. \tag{2.20}$$

Bilinear Lagrange shape functions resulting from the tensor product of their 1D counterparts (2.20) are listed below:

$$\begin{aligned}
\hat{\chi}_1(\xi,\eta) &= \hat{\vartheta}_1(\xi)\hat{\vartheta}_1(\eta) = 1/4[1-\xi-\eta+\xi\eta] \\
\hat{\chi}_2(\xi,\eta) &= \hat{\vartheta}_2(\xi)\hat{\vartheta}_1(\eta) = 1/4[1+\xi-\eta-\xi\eta] \\
\hat{\chi}_3(\xi,\eta) &= \hat{\vartheta}_2(\xi)\hat{\vartheta}_2(\eta) = 1/4[1+\xi+\eta+\xi\eta] \\
\hat{\chi}_4(\xi,\eta) &= \hat{\vartheta}_1(\xi)\hat{\vartheta}_2(\eta) = 1/4[1-\xi+\eta-\xi\eta]
\end{aligned} \tag{2.21}$$

It is easy to show that these shape functions possess all the general properties discussed before. A shape function $\hat{\chi}_j^{(k)}$ varies linearly along the two edges containing the node j and is identically equal to zero at the other two edges. Thus, the global basis function χ_j resulting from the union of all the shape functions at the node j is non-zero at all the elements containing node j and vanishes at all other elements.

The bilinear map $F_k : \Omega_{ref} \mapsto \Omega_k$, defines a transformation between the reference element Ω_{ref} and the physical element Ω_k as follows:

$$\begin{bmatrix} x \\ y \end{bmatrix} = F_k(\xi,\eta) = \sum_{i=1}^{4} \begin{bmatrix} x_i \\ y_i \end{bmatrix} \hat{\chi}_i(\xi,\eta). \tag{2.22}$$

Above transformation leads to an isoparametric mapping on each element, where same bilinear interpolation functions are used for both the geometry (mappings of nodal coordinates) and the unknown function u. For the inverse mapping $F_k^{-1} : \Omega_k \mapsto \Omega_{ref}$ to hold, it is necessary that the physical quadrilateral Ω_k must be *convex.*

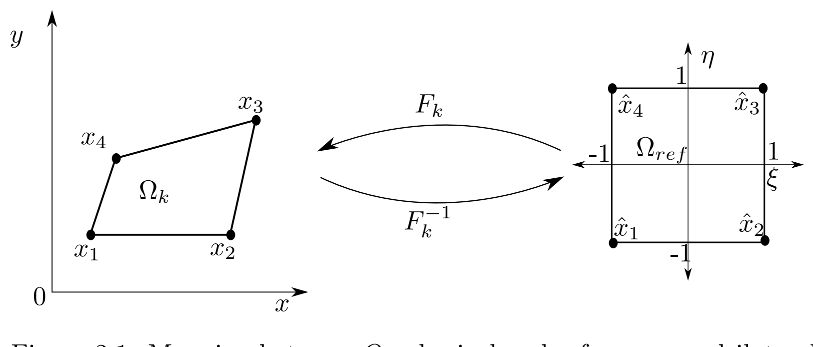


Figure 2.1: Mapping between Q_1 physical and reference quadrilaterals.

2.2.2 Biquadratic element (Q_2)

Biquadratic finite element Q_2 consists of nine nodal degrees of freedom, with four corner nodes, four mid-side nodes and one node at the center of quadrilateral as shown in Figure 2.2. Shape functions on these nodes are constructed by the tensor product of the following one dimensional quadratic Lagrange polynomials

$$\hat{\theta}_1(\xi) = \frac{\xi(\xi-1)}{2}, \quad \hat{\theta}_2(\xi) = \frac{\xi(\xi+1)}{2}, \quad \hat{\theta}_3(\xi) = \frac{(1-\xi^2)}{2}, \quad -1 \le \xi \le 1.$$

Biquadratic shape functions at the nodes of the reference element are as follows:

$$\begin{aligned}
&\hat{\phi}_1(\xi,\eta) = \hat{\theta}_1(\xi)\hat{\theta}_1(\eta), \quad \hat{\phi}_4(\xi,\eta) = \hat{\theta}_1(\xi)\hat{\theta}_2(\eta), \quad \hat{\phi}_7(\xi,\eta) = \hat{\theta}_3(\xi)\hat{\theta}_2(\eta) \\
&\hat{\phi}_2(\xi,\eta) = \hat{\theta}_2(\xi)\hat{\theta}_1(\eta), \quad \hat{\phi}_5(\xi,\eta) = \hat{\theta}_3(\xi)\hat{\theta}_1(\eta), \quad \hat{\phi}_8(\xi,\eta) = \hat{\theta}_1(\xi)\hat{\theta}_3(\eta) \qquad (2.23) \\
&\hat{\phi}_3(\xi,\eta) = \hat{\theta}_2(\xi)\hat{\theta}_2(\eta), \quad \hat{\phi}_6(\xi,\eta) = \hat{\theta}_2(\xi)\hat{\theta}_3(\eta), \quad \hat{\phi}_9(\xi,\eta) = \hat{\theta}_3(\xi)\hat{\theta}_3(\eta)
\end{aligned}$$

Bilinear mapping (2.22) can be used for the transformations between the Q_2 reference and physical elements, which will be *subparametric* mapping in this scenario (Figure: 2.2, left). It is also possible to define *isoparametric* mapping using the biquadratic shape functions (2.23), which are useful for discretizing curved boundaries (Figure: 2.2, right). For more details, please follow the references [103, 34]

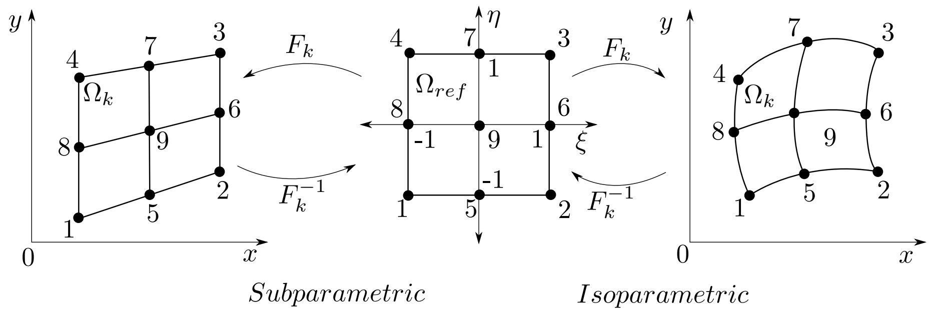


Figure 2.2: Bilinear and biquadratic mappings between Q_2 physical and reference elements.

2.3 Error estimates for FEM

In this section, we mention the results of the so-called *a priori error estimates* for the finite element method without going into their details and proofs. The finite element solution error $u - u_h$, can be bounded in some $n-$norm $\|.\|_n$ by the following relation

$$\|u - u_h\|_n \le Ch^{\alpha}\|u\|_r \qquad \text{with } \alpha = \min(k+1-n, r-n), \qquad (2.24)$$

where h is the maximum mesh width, C is the constant independent of mesh width, k is the *polynomial degree* of the finite element basis functions, and r is the regularity of the exact solution with $\|u\|_r$ representing the measure of smoothness of u. Now for a sufficiently smooth u, the bound (2.24) for the H^1-norm ($n = 1$) becomes

$$\|u - u_h\|_1 \leq Ch^k \|u\|_r, \tag{2.25}$$

and similarly, provided that the problem is H^2 regular which means that $f \in L_2(\Omega)$ and the solution of the dual problem is in $H_2(\Omega)$ and satisfies the a priori bound

$$\|u\|_{H_2} \leq c\|f\|_{L_2},$$

then the a priori error bound in the L_2 or H^0 norm ($n = 0$) becomes

$$\|u - u_h\|_0 \leq Ch^{k+1} \|u\|_r. \tag{2.26}$$

For proof of the a priori errors see the book by C. Johnson [81]. Please note that the *polynomial degree* here means the highest degree complete polynomial that can be represented exactly by the shape functions. In case of bilinear Q_1 element, although the local basis functions (2.21) contain quadratic monomial $(\xi\eta)$, however, they can represent polynomials of degree one accurately. Thus, for Q_1 element we have $k = 1$, and for a sufficiently smooth u, the error is $\mathcal{O}(h)$ in $\mathcal{H}^1$-norm and $\mathcal{O}(h^2)$ in the L_2-norm. Similarly, the shape functions (2.23) of Q_2 element contain higher order terms, but the highest degree polynomial that they can represent exactly is two. In this case, therefore, we have $k = 2$ and the solution error is $\mathcal{O}(h^2)$ in $\mathcal{H}^1$-norm and $\mathcal{O}(h^3)$ in the L_2-norm.

In general, we do not know the exact solution, and these a priori error estimates can never be computed in realistic simulations. Nevertheless, these error estimates provide an excellent debugging tool for the code validation purposes. To debug the code, the problem with a sufficiently smooth known exact solution is solved. If our numerical scheme/code is correctly implemented, then the solution error with each mesh refinement should reduce with some constant factor. For example, for the Q_1 finite element discretization, the error should reduce with a factor of two in the $\mathcal{H}^1$-norm and with a factor of four in the L_2-norm. If these factors are not observed in the error reduction, then there is most probably a bug in the code.

3 Solution Methods for Sparse Linear Systems

Discretization of linear PDEs as seen in chapter 2, and linearization and discretization of nonlinear PDEs (see chapter 7), lead to the *linear system* of equations of the form:

$$\mathbf{Ax} = \mathbf{b}, \tag{3.1}$$

where the coefficient matrix $\mathbf{A}$ is large and sparse. A *sparse matrix* is a matrix with enough zeroes so that special algorithms and data structures can be devised to save time and memory by exploiting these zeros [41]. Solving such large and sparse systems is the bottleneck of the modeling and simulation process, as it takes most of the computational cost in the whole process. The research community has put a huge amount of effort to develop algorithms that can efficiently solve large sparse linear systems, and till today this is still a hot research area. These enormous research activities over the period of decades have resulted in a large number of linear solvers, which can be grouped into two main categorize namely direct methods and iterative methods.

We begin this chapter with a brief overview of the direct methods for sparse linear systems, and mention their strengths and weaknesses. Next section discusses the iterative methods, which are the preferred choice over the direct methods, for large sparse linear systems. These iterative methods fall under three subgroups, namely, basic iterative methods, Krylov subspace methods, and multigrid methods, based on their working principle. We explain the working principles of each group and mention some well-known solvers from each of them. We also discuss the node renumbering strategies that are quite often employed to help improve the performance of linear solvers.

3.1 Sparse direct methods

Direct methods are smart variants of Gaussian elimination and involve the following three steps:

1. Perform matrix factorization

$$\mathbf{PAQ} = \mathbf{LU}, \tag{3.2}$$

where **P** and **Q** are permutation matrices chosen to reduce fill-ins and to maintain stability, **L** and **U** are lower triangular and upper triangular matrices, respectively.

2. Perform forward elimination

$$\mathbf{Lz} = \mathbf{Pb}.$$

3. Perform back-substitution

$$\mathbf{UQ^T x} = \mathbf{z}.$$

If the matrix **A** is symmetric, then following factorization is used:

$$\mathbf{PAP^T} = \mathbf{LDL^T}, \tag{3.3}$$

where **D** is (block) diagonal matrix [2].

Dense Gaussian elimination algorithm costs $\mathcal{O}(n^3)$ flops whereas forward, and backward substitutions require $\mathcal{O}(n^2)$ flops, where n is the number of unknowns. However various very effective and efficient sparse implementations of Gaussian elimination have been developed by using sophisticated techniques to exploit the sparsity in the coefficient matrix to minimize the computational costs. *Frontal method* [79] is one such technique that was initially developed for solving symmetric positive-definite (SPD) sparse linear systems coming from finite element applications. In the frontal method, the Gaussian elimination process is carried out in parallel with the finite element assembly process. During the assembly process as soon as the variable is entirely summed (i.e., subsequent assemblies do not affect the values in its rows and columns), row operations are carried out to make the entries below diagonal zero, and the resulting row is saved as a new row of upper triangular U matrix; see for instance Johnson [81, page 117-120]. Th name frontal method comes from the fact that at each elimination step, only the entries (also termed active variables) in the small dense matrix, called *frontal matrix* are modified, which forms the front that separates the eliminated variables (behind the front) from the not-yet activated variables (after the front) in the finite-element mesh.

The frontal approach has many benefits over the standard Gaussian elimination method[2]. First, it allows the use of efficient BLAS subroutines for dense matrix calculations. Second, it improves the stability of algorithm by applying the pivoting only in the frontal matrix. Thirdly, because of the small size of the frontal matrices, the elimination process can be efficiently carried out in a fast cache memory.

Reordering schemes are often used in a frontal method to reduce the number of fill-ins during the elimination process; this reduces both the computational and memory costs of the method. However, the reordering results into frontal matrices that are too small to achieve proper exploitation of the memory hierarchy available in modern processors. Moreover, the frontal method lacks the scope of parallelism except what can be accomplished within the dense BLAS operations. *The multifrontal method* by Duff and Reid [48], which is, in fact, the generalization of the frontal method, addresses the above issues by using many fronts at the same time. Prior to factorization, the method comprises *analysis* phase that involves symbolic preprocessing operations to determine various independent fronts that can be processed in parallel. This in addition to providing more possibility of parallelism, also gives the liberty of using sparsity preserving pivot orderings. For a detailed description of the multifrontal method see also [86].

Multifrontal methods perform efficiently if the matrix has a structural symmetric pattern, and may give poor performance for matrices whose patterns are highly unsymmetric. Davis and Duff in [40] propose the first efficient unsymmetric-pattern multifrontal method (UMFPACK) for general matrices with a highly unsymmetric structural pattern. The term UMFPACK is also used for the user-callable subroutines package available for the solution of the unsymmetric sparse linear system, that uses unsymmetric multifrontal method [38]. For more details on sparse direct solvers see [47],[39], [42].

Despite the fact that the classy multifrontal approach combined with the intelligent renumbering schemes increase the efficiency of direct solvers greatly, for very large sparse systems (as for instance in 3D case) there associated CPU costs are still very high and more importantly, their huge memory requirements in such cases make them even impractical for use [130]. Nevertheless, they are used as a building block inside iterative multilevel solvers for large sparse systems.

3.2 Iterative Methods

In many areas of scientific computing, iterative methods have almost replaced direct methods for solving general, large sparse linear systems. In iterative methods, the coefficient matrix remains unchanged, which means no additional memory is needed for the fill-ins. Therefore, memory requirements for iterative methods are much less than direct methods. Iterative methods generate a sequence of approximate solution vectors $\{\mathbf{x}_k\}, k = 0, 1, ...$ ($\mathbf{x}_0$ given), with the iterates converging towards the exact solution $\mathbf{x}$, as $k \to \infty$

$$\lim_{k\to\infty} \|\mathbf{x} - \mathbf{x}_k\| = 0.$$

The iteration process can be stopped as soon as the desired accuracy is achieved. These features make iterative methods very attractive for solving large sparse systems, especially if the accuracy requirements are low.

Iterative methods are also particularly suitable for nonlinear and nonstationary problems. In such cases, the solution of the linear system is part of an outer iteration loop: Fixed point or Newton-Raphson linearization iteration for a nonlinear problem and time stepping iteration for a time-dependent problem. For each inner linear solve, a good start vector in the form of a solution of the previous outer iteration is available; moreover, the accuracy requirement for inner solve is generally low. Both properties lead to the fact that few iterations are required to solve the linear system approximately.

Iterative methods for solving equation (3.1) can be broadly categorized as:

- Basic Iterative Methods

- Krylov Subspace Methods
- Multigrid Methods

In the following sections, we give an overview of these methods. For a more comprehensive survey of iterative methods for linear systems we refer to [122].

3.2.1 Basic iterative methods

The main idea of basic iterative methods is to *split* the coefficient matrix $\mathbf{A}$ as a sum of two matrices

$$\mathbf{A} = \mathbf{M} - \mathbf{N},$$

where $\mathbf{M}$ is an easily invertible matrix. Then (3.1) can be written as a sequence $\mathbf{x}_k$

$$\mathbf{M}\mathbf{x}_{k+1} = \mathbf{N}\mathbf{x}_k + \mathbf{b}. \tag{3.4}$$

It can be easily seen that for a converging iteration process ($\mathbf{x}_k \rightarrow \mathbf{x}$), the vector $\mathbf{x}$ is also the solution to (3.1). Replacing $\mathbf{N}$ by $\mathbf{N} = \mathbf{M} - \mathbf{A}$ in equation (3.4), we may write

$$\mathbf{x}_{k+1} = \mathbf{x}_k + \mathbf{M}^{-1}\mathbf{r}_k, \tag{3.5}$$

where $\mathbf{r}_k = \mathbf{b} - \mathbf{A}\mathbf{x}_k$ is the residual at k^{th} iteration. Equation (3.5) is often called the basic *preconditioned Richardson iteration.* Adding a damping parameter (relaxation parameter) ω leads to its damped version:

$$\mathbf{x}_{k+1} = \mathbf{x}_k + \omega\mathbf{M}^{-1}\mathbf{r}_k. \tag{3.6}$$

The choice of $\mathbf{M}$, also called *preconditioner*, is very crucial and different choices lead to different iterative methods. Desired properties for a good preconditioner include that it should be (spectrally) close to $\mathbf{A}$, should be easy to build and apply. To show some basic choices of $\mathbf{M}$, we express the matrix $\mathbf{A}$ as the matrix sum

$$\mathbf{A} = \mathbf{D} - \mathbf{E} - \mathbf{F},$$

in which $\mathbf{D}$ is the diagonal of $\mathbf{A}$, -$\mathbf{E}$ and -$\mathbf{F}$ are strict lower triangular and strict upper triangular parts of $\mathbf{A}$ respectively. Choosing $\mathbf{M}$ to be the diagonal of $\mathbf{A}$ results in *point Jacobi iteration*:

$$\mathbf{M}^{JAC} := \mathbf{D}$$

In Jacobi method, each equation (unknown) is treated independently. Therefore, the order in which the equations are treated does not affect the convergence behavior of the method. For this reason, the reordering techniques discussed later, will not be applied to Jacobi method. It is also called *a method of simultaneous displacements*[140] since the updates could be done simultaneously. This feature makes Jacobi method a good candidate for parallel implementation.

Letting $\mathbf{M}$ be the lower triangular part of $\mathbf{A}$ defines what is called *point Gauß-Seidel iteration*:

$$\mathbf{M}^{GS} := (\mathbf{D} - \mathbf{E})$$

In contrast to the Jacobi method, in the Gauß-Seidel method the equations are examined sequentially, one at a time, in such a way that the most current estimates of previously computed components are used in calculations. The updates cannot be done simultaneously, because each component of the new iterate relies on the already calculated components. Another consequence of sequential dependence is that the elements of new iterate depend on the order in which equations are solved. If the order changes, it also alters the elements of new iterate. Depending on the ordering, families of Gauß-Seidel preconditioners with different numerical properties can be constructed[145].

In case of sparse $\mathbf{A}$, the presence of zeroes may nullify the influence of some of the preceding components. By employing prudent reordering schemes, it may be possible to reduce such dependence further, thus allowing certain groups of unknowns to be processed in parallel. Nevertheless, reorderings aimed at parallelism can hurt the convergence of Gauß-Seidel method, and there is always a tradeoff between parallelism and convergence rate [12].

Successive overrelaxation method (SOR) extrapolates between the current and previous iterates of Gauß-Seidel componentwise. This extrapolation is achieved by using an additional relaxation parameter $\tilde{\omega}$ (different from ω in (3.6)) with the lower triangular part; the idea is to accelerate the convergence rate of the iterates to the solution.

$$\mathbf{M}^{SOR} := (\mathbf{D} - \tilde{\omega}\mathbf{E})$$

Defining $\mathbf{e}_k = \mathbf{x} - \mathbf{x}_k$ to be the k^{th} iteration error, equation (3.4) takes the form

$$\begin{aligned} \mathbf{M}(\mathbf{x} - \mathbf{x}_{k+1}) &= \mathbf{N}(\mathbf{x} - \mathbf{x}_k) \\ \mathbf{e}_{k+1} &= \mathbf{M}^{-1}\mathbf{N}\mathbf{e}_k \\ \mathbf{e}_{k+1} &= (\mathbf{M}^{-1}\mathbf{N})^k\mathbf{e}_0 \end{aligned} \tag{3.7}$$

The matrix $(\mathbf{M}^{-1}\mathbf{N})$ is called the *iteration matrix.* Using Jordan form its easy to show that $(\mathbf{M}^{-1}\mathbf{N})^k \to 0$ as $k \to \infty$, if for all the eigenvalues of iteration matrix it holds that $|\lambda| < 1$; which implies that $\mathbf{e}_{k+1} \to 0$ as $k \to \infty$. Hence the eigenvalues of iteration matrix play a vital role in the convergence of basic iterative methods. Above discussion can be formulated as a theorem.

Theorem 3.2.1. *The iterative method* (3.4) *converges to the exact solution* $\mathbf{x} = \mathbf{A}^{-1}\mathbf{b}$ *for any starting vector* $\mathbf{x}_0$*, if the spectral radius*[1] *of iteration matrix* $\rho(\mathbf{M}^{-1}\mathbf{N}) < 1$.

The convergence condition given in the above theorem is guaranteed only if the coefficient matrix $\mathbf{A}$ of the linear systems is M-matrix. Moreover, the theorem gives the criterion for the convergence of basic iterative schemes, but does not say anything about the rate of convergence of these methods. These schemes have local stencil since they involve averaging of nodal values from immediate neighbors in some order and neglect the effects of neighbors which are computationally far away. The detailed Fourier analysis of these schemes [24], [71] reveals that such local averagings remove the local (or high frequency) error components quickly in just a few iterations, leaving the low frequency or smooth (less oscillatory) components relatively unchanged. However, once the error field is smoothed out (i.e., it consists of only low frequency errors), the convergence rates of basic relaxation schemes are significantly reduced. For many applications, their convergence rates are very sensitive to the value ω, and it is difficult

[1]Spectral radius of $\mathbf{A} = \rho(\mathbf{A}) = \max|\lambda_i|$, where λ_i are the eigenvalues of $\mathbf{A}$ }

in practice to find an optimal choice of the damping parameter ω. Moreover, the convergence rates of these solvers depend on the mesh width h and on finer mesh levels, which are usually required for achieving the desired accuracy, the basic iterative methods converge very slowly. For these very reasons, they are almost never used as standalone solvers, instead, are used as preconditioners in advanced iterative methods such as Krylov subspace solvers and in particular as *smoothers* in multigrid solvers. For more discussion on the convergence of basic iterative methods see [140], [145] and [149].

3.2.2 Krylov subspace methods

Krylov subspace methods are the most widely used iterative methods to solve sparse linear systems, and are included in "Top 10" best algorithms of the 20th century [30]. The main reasons for this popularity are their low memory requirements and good approximation properties. Mathematically speaking, these methods are projection based methods (see [118], chapter 5). Instead of solving the potentially very huge linear system, Krylov methods through projections extract an approximate solution from an affine subspace $\mathbf{x}_0 + \mathcal{K}_m$ of dimension $m (m \ll n)$, where $\mathbf{x}_0$ is the initial guess and $\mathcal{K}_m$ is the *Krylov subspace* defined as:

$$\mathcal{K}_m(\mathbf{A}, \mathbf{r}_0) = span\{\mathbf{r}_0, \mathbf{A}\mathbf{r}_0, \mathbf{A}^2\mathbf{r}_0, ..., \mathbf{A}^{m-1}\mathbf{r}_0\},$$

with $\mathbf{r}_0 = \mathbf{b} - \mathbf{A}\mathbf{x}_0$. A popular choice is to take initial guess $\mathbf{x}_0 = 0$ which gives $\mathbf{r}_0 = b$, and the corresponding Krylov subspace is $\mathcal{K}_m(\mathbf{A}, \mathbf{b})$ generated by right-hand side $\mathbf{b}$ of the linear system being solved.

Why is Krylov subspace a nice subspace?

Now the question may arise why one would construct the solution from Krylov subspace, or why Krylov methods are a natural way to solve the linear systems? To answer this question we follow the discussion in [78], which uses the minimal polynomial of $\mathbf{A}$ to show that $\mathbf{x} = \mathbf{A}^{-1}\mathbf{b}$ (for nonsingular $\mathbf{A}$) is naturally contained in a Krylov subspace.
The minimal polynomial $p(z)$ of a matrix $\mathbf{A}$ is defined as a unique monic polynomial of smallest degree for which $p(\mathbf{A}) = 0$. If $\lambda_1, ..., \lambda_d$ are the distinct eigenvalues of $\mathbf{A}$ with λ_j having index m_j (the maximal dimension of the Jordan block containing λ_j), then

$$p(z) = \prod_{j=1}^{d} (z - \lambda_j)^{m_j},$$

moreover, for $m \equiv \sum_{j=1}^{d} m_j$, we can write

$$p(z) = \sum_{j=0}^{m} \gamma_j z^j,$$

with $\gamma_0 = \prod_{j=1}^{m}(-\lambda_j)^{m_j}$ ($\gamma_0 \neq 0$ for nonsingular $\mathbf{A}$). Since

$$0 = p(\mathbf{A}) = \gamma_0 I + \gamma_1 \mathbf{A} + ... + \gamma_m \mathbf{A}^m,$$

where $\mathbf{I}$ is the identity matrix, thus

$$\mathbf{A}^{-1} = \frac{1}{\gamma_0}\sum_{j=0}^{m-1} \gamma_{j+1}\mathbf{A}^j. \tag{3.8}$$

With $\mathbf{A}^{-1}$ in above form, $\mathbf{x} = \mathbf{A}^{-1}\mathbf{b}$ can be immediately seen as a member of Krylov space

$$\mathbf{x} = \mathbf{A}^{-1}\mathbf{b} = (\tilde{\gamma}_1\mathbf{b} + \tilde{\gamma}_2\mathbf{A}\mathbf{b} + ... + \tilde{\gamma}_m\mathbf{A}^{m-1}\mathbf{b}), \tag{3.9}$$

with $\tilde{\gamma}_i = \frac{\gamma_i}{\gamma_0}$. Therefore, if the degree of minimal polynomial is small, the dimension of Krylov subspace containing the solution is also small, and the Krylov method will converge faster.

Many Krylov subspace methods have been developed, and here we discuss few of the most widely used such methods.

Conjugate Gradient Method (CG)

Conjugate Gradient method by Hestenes and Stiefel [73] is an efficient and one of the best methods for symmetric ($\mathbf{A} = \mathbf{A}^T$) and positive definite ($\mathbf{x}^T\mathbf{A}\mathbf{x} > 0$ for $x \neq 0$) systems. For an SPD $\mathbf{A}$, *A-norm (or energy norm)* can be defined as

$$\|\mathbf{x}\|_A^2 := \mathbf{x}^T\mathbf{A}\mathbf{x}. \tag{3.10}$$

The CG method at the k^{th} iteration constructs an approximate solution $\mathbf{x}_k \in \mathbf{x}_0 + \mathcal{K}_k(\mathbf{A}, \mathbf{r}_0)$ that is optimal in the sense that it minimizes the energy norm of the error vector

$$\|\mathbf{x} - \mathbf{x}_k\|_A = \min_{\mathbf{y}\in\mathcal{K}_k(\mathbf{A},\mathbf{r}_0)} \|\mathbf{x} - \mathbf{y}\|_A. \tag{3.11}$$

The solution of the above minimization problem results in the conjugate gradient algorithm 3.1. The CG algorithm constructs search directions which are **A**-

Algorithm 3.1 CG algorithm

```
1: Compute $\mathbf{r}_0 = \mathbf{b} - \mathbf{A}\mathbf{x}_0$ for some initial guess $\mathbf{x}_0$, $\mathbf{p}_0 = \mathbf{r}_0$    ▷ initialization
2: for $k = 0, 1, ...$ until convergence, do
3:     $\mathbf{w}_k = \mathbf{A}\mathbf{p}_k$
4:     $\alpha_k = \frac{\mathbf{r}_k{}^T\mathbf{r}_k}{\mathbf{p}_k{}^T\mathbf{w}_k}$
5:     $\mathbf{x}_{k+1} = \mathbf{x}_k + \alpha_k\mathbf{p}_k$    ▷ update solution
6:     $\mathbf{r}_{k+1} = \mathbf{r}_k - \alpha_k\mathbf{w}_k$    ▷ update residual
7:     $\beta_k = \frac{\mathbf{r}_{k+1}{}^T\mathbf{r}_{k+1}}{\mathbf{r}_k{}^T\mathbf{r}_k}$
8:     $\mathbf{p}_{k+1} = \mathbf{r}_{k+1} + \beta_k\mathbf{p}_k$    ▷ update search direction
9: end for
```

orthogonal or *conjugate* to each other. This motivates the name of the algorithm: the directions (or gradients) of the updates are conjugate to each other.

$$\mathbf{A}\mathbf{p}_i.\mathbf{p}_j = \delta_{ij} \quad \forall i,j = 1,...,N.$$

For a symmetric $\mathbf{A}$, a three-term recurrence suffices for generating the orthogonal basis of Krylov subspace. The same is true for the residuals being generated by conjugate gradient algorithm as they are also orthogonal to each other. In CG method, however, two *coupled two-term* recursions are used: one for updating the residual and the other for updating the search direction. This feature of the conjugate gradient makes it very attractive solver since there is no need to store all the previous residuals or search directions.

The algorithm involves two ratios, one for calculating β_k and other for α_k. The algorithm breaks down if the denominator in these ratios happens to be zero. However, these breakdowns are *lucky breakdowns*, as they indicate that solution has been reached. For the case of β_k, the zero denominator means $\mathbf{r}_k{}^T\mathbf{r}_k = 0$, so $\mathbf{r}_k = 0$, and thus $\mathbf{x}_k = \mathbf{x}$, which means that the linear system has been solved. Similarly for α_k, if the denominator is zero we have $\mathbf{p}_k{}^T\mathbf{A}\mathbf{p}_k = 0$, which implies $\mathbf{p}_k = 0$. Now using the fact that $span\{\mathbf{p}_0, ..., \mathbf{p}_k\} = span\{\mathbf{r}_0, ..., \mathbf{r}_k\}$, this again implies that $\mathbf{r}_k = 0$ thus further implying that the solution has been reached.

The computational cost of conjugate gradient method involves one matrix-vector product, three vector updates, two inner products, and one norm evaluation (for stopping criterion). The scheme requires storage of 4 vectors ($\mathbf{x}, \mathbf{r}, \mathbf{p}$, and $\mathbf{w}$), along with some scalars. The convergence rate of CG scheme depends on the spectral condition number[2] of matrix $\mathbf{A}$.

Theorem 3.2.2. *At the k^{th} iteration the iterate obtained from the CG algorithm satisfies the following inequality:*

$$\|\mathbf{x} - \mathbf{x}_k\|_A \leq 2\Big(\frac{\sqrt{\kappa_2(\mathbf{A})} - 1}{\sqrt{\kappa_2(\mathbf{A})} + 1}\Big)^k \|\mathbf{x} - \mathbf{x}_0\|_A. \tag{3.12}$$

Proof. *See [89, p.187].*

Above theorem suggests that CG algorithm has a linear convergence, however, in practice a *superlinear convergence* of the scheme is observed if the extremal eigenvalues of the matrix are well separated. In [137], authors have shown that during the iteration process the Ritz values converge to extremal eigenvalues and as soon as eigenvalues of the original operator are well approximated by Ritz values, the error vector has no components related to the eigenvectors of these eigenvalues. Therefore, the CG algorithm converges as fast as for a related system in which these eigenvalues are missing. For more discussion on the convergence of CG see also [12] and [31].

According to equation (3.11), as CG iterations proceed there is a monotone decrease in energy norm of the error. In fact, this is merely a theoretical result, and in practice, we cannot compute $\|\mathbf{x} - \mathbf{x}_k\|_A$ because $\mathbf{x}$ is never known. In practice we compute $\|\mathbf{r}_k\|_2$, however, CG does not minimize this quantity and it may not reduce monotonously.

[2]If λ_{max} and λ_{min} are the largest and smallest eigenvalues of a symmetric positive definite matrix $\mathbf{A}$, then the spectral condition number of $\mathbf{A}$ is defined as $\kappa_2(\mathbf{A}) = \lambda_{max}/\lambda_{min}$.

Remark 3.2.3. *We list here the three sought-after characteristics of CG algorithm, which make it such a successful and widely used solver:*

1. *The solution obtained belongs to Krylov subspace i.e; $\mathbf{x}_k \in \mathcal{K}_k(\mathbf{A}, \mathbf{r}_0)$.*
2. *The algorithm involves short recurrences.*
3. *CG is based on a certain optimality property.*

In [56], Faber and Manteuffel have shown that for a general matrix $\mathbf{A}$ (the only condition on $\mathbf{A}$ is that it is nonsingular), it is impossible to construct a Krylov method which has all the characteristics mentioned in remark 3.2.3. Table 3.1 shows for three popular Krylov subspace methods for general matrices, which of these properties they possess.

	CGNR	Bi-CGSTAB	GMRES
$\mathbf{x}_k \in \mathcal{K}_k(\mathbf{A}, \mathbf{r}_0)$	✗	✓	✓
Short recurrence	✓	✓	✗
Optimality	✓	✗	✓

Table 3.1: Characteristics and Krylov methods.

CGNR method

The idea of CGNR method is that for a general linear system $\mathbf{Ax} = \mathbf{b}$ with a non-SPD $\mathbf{A}$, solve the related SPD system $(\mathbf{A}^T\mathbf{A})$ with conjugate gradient method:

$$\mathbf{A}^T\mathbf{Ax} = \mathbf{A}^T\mathbf{b}, \qquad \mathbf{A}^T\mathbf{A} \text{ is SPD.} \tag{3.13}$$

Note that CGNR will be very fast if $\mathbf{A}$ is close to the unitary matrix $(\mathbf{Q}^T\mathbf{Q} = \mathbf{I})$. The k^{th} iterate of CGNR minimizes the following energy norm of the error:

$$\begin{aligned} \|\mathbf{x} - \mathbf{x}_k\|_{A^TA} &= (\mathbf{x} - \mathbf{x}_k)^T\mathbf{A}^T\mathbf{A}(\mathbf{x} - \mathbf{x}_k) \\ &= (\mathbf{Ax} - \mathbf{Ax}_k)^T(\mathbf{Ax} - \mathbf{Ax}_k) \\ &= \|\mathbf{b} - \mathbf{Ax}_k\|_2^2. \end{aligned}$$

Therefore, CGNR produces iterates in the Krylov subspace $\mathcal{K}_k(\mathbf{A}^T\mathbf{A}, \mathbf{A}^T\mathbf{r}_0)$ that minimize the norm of the residual related to the original linear system $\mathbf{Ax} = \mathbf{b}$. Generalized minimal residual (GMRES) solver also minimizes the same residual norm but in different Krylov subspace. Although the approach of CGNR seems to be easy, there are some serious drawbacks associated which hinder the scheme to be a popular method of choice for most of the applications. First, because $\kappa(\mathbf{A}^T\mathbf{A}) = \kappa(\mathbf{A}^2)$, the convergence rate of the conjugate gradient method may reduce significantly, as it now depends on the square of the condition number of the actual coefficient matrix. Second, the preconditioning normally used in Krylov subspace methods to improve the performance is difficult to apply on normal equations (see [118, section 10.8, p. 339]). Third, as seen in algorithm 3.2, the scheme requires two matrix-vector products per iteration which increases the computational cost of the algorithm considerably.

Algorithm 3.2 CGNR algorithm

```
1: Compute $\mathbf{r}_0 = \mathbf{b} - \mathbf{A}\mathbf{x}_0$, $\mathbf{z}_0 = \mathbf{A}^T\mathbf{r}_0$, $\mathbf{p}_0 = \mathbf{z}_0$    ▷ initialization
2: for $k = 0, 1, ...$ until convergence, do
3:     $\mathbf{w}_k = \mathbf{A}\mathbf{p}_k$
4:     $\alpha_k = \frac{\mathbf{z}_k{}^T\mathbf{z}_k}{\mathbf{w}_k{}^T\mathbf{w}_k}$
5:     $\mathbf{x}_{k+1} = \mathbf{x}_k + \alpha_k\mathbf{p}_k$    ▷ update solution
6:     $\mathbf{r}_{k+1} = \mathbf{r}_k - \alpha_k\mathbf{w}_k$    ▷ update residual
7:     $\mathbf{z}_{k+1} = \mathbf{A}^T\mathbf{r}_{k+1}$
8:     $\beta_k = \frac{\mathbf{z}_{k+1}{}^T\mathbf{z}_{k+1}}{\mathbf{z}_k{}^T\mathbf{z}_k}$
9:     $\mathbf{p}_{k+1} = \mathbf{z}_{k+1} + \beta_k\mathbf{p}_k$    ▷ update search direction
10: end for
```

Generalized Minimal Residual (GMRES) Method

The generalized minimal residual (GMRES) method developed by Saad and Schultz in 1986 [121] is applicable to general matrices. We follow [78] to explain the working idea of GMRES. In the k^{th} iteration, the GMRES algorithm chooses the 'optimal' solution $\mathbf{z}_k \in \mathcal{K}_k(\mathbf{A}, \mathbf{r}_0)$, in such a way that the residual is minimized in Euclidean norm over $\mathcal{K}_k(\mathbf{A}, \mathbf{r}_0)$. For $\mathbf{x}_k = \mathbf{x}_0 + \mathbf{z}_k$ with $\mathbf{z}_k \in \mathcal{K}_k(\mathbf{A}, \mathbf{r}_0)$,

$$\|\mathbf{r}_k\|_2 = \|\mathbf{b} - \mathbf{A}\mathbf{x}_k\|_2 = \min_{\mathbf{z}\in\mathcal{K}_k(\mathbf{A},\mathbf{r}_0)} \|\mathbf{r}_0 - \mathbf{A}\mathbf{z}\|_2. \tag{3.14}$$

GMRES solves the above least squares problem by constructing the orthonormal basis for $\mathcal{K}_k(\mathbf{A}, \mathbf{r}_0)$ using *Arnoldi iteration*, which is nothing but modified Gram-Schmidt procedure adapted for the Krylov subspace. Arnoldi method works as follows: for a given set of orthonormal basis $\{\mathbf{v}_1, \mathbf{v}_2, ..., \mathbf{v}_j\}$ for $\mathcal{K}_j(\mathbf{A}, \mathbf{r}_0)$, the basis is expanded by orthonormalizing the vector $\mathbf{A}\mathbf{v}_j$ against the previous basis. If we collect the orthonormal basis vectors in a matrix form, $\mathbf{V}_j = (\mathbf{v}_1...\mathbf{v}_j)$, we can write

$$\mathbf{A}\mathbf{V}_j = \mathbf{V}_{j+1}\mathbf{H}_j,$$

where $\mathbf{H}_j$ is an upper Hessenberg matrix[3] of size $(j+1) \times j$. Now any vector $\mathbf{z} \in \mathcal{K}_k$ can be written as

$$\mathbf{z} = \mathbf{V}_k\mathbf{y}, \tag{3.15}$$

for some $\mathbf{y}$. So

$$\begin{aligned}\mathbf{r}_0 - \mathbf{A}\mathbf{z} = & \ \mathbf{r}_0 - \mathbf{A}\mathbf{V}_k\mathbf{y} \\ = & \ \beta\mathbf{v}_1 - \mathbf{V}_{k+1}\mathbf{H}_k\mathbf{y} \\ = & \ \mathbf{V}_{k+1}(\beta\mathbf{e}_1 - \mathbf{H}_k\mathbf{y}),\end{aligned} \tag{3.16}$$

where $\beta = \|\mathbf{r}_0\|_2$ and $\mathbf{e}_1$ is first column of identity matrix. As $\mathbf{V}_{k+1}$ is a unitary matrix, we have

$$\|\mathbf{r}_0 - \mathbf{A}\mathbf{z}\|_2 = \|\beta\mathbf{e}_1 - \mathbf{H}_k\mathbf{y}\|_2 \tag{3.17}$$

[3]A matrix with zeroes below first sub-diagonal

Thus the least squares problem in the k^{th} iteration of GMRES becomes

$$\min_{\mathbf{z}\in\mathcal{K}_k(\mathbf{A},\mathbf{r}_0)} \|\mathbf{r}_0 - \mathbf{A}\mathbf{z}\|_2 = \min_{\mathbf{y}} \|\beta\mathbf{e}_1 - \mathbf{H}_k\mathbf{y}\|_2 \tag{3.18}$$

Typically, it is inexpensive to compute the minimizer $\mathbf{y}$, as it involves solving a smaller $(m+1)\times m$ least squares problem. The GMRES method is shown in algorithm 3.3. The minimization problem (3.18) is solved by converting

Algorithm 3.3 GMRES algorithm

```
1: Compute $\mathbf{r}_0 = \mathbf{b} - \mathbf{A}\mathbf{x}_0$, $\beta = \|\mathbf{r}_0\|_2$, and $\mathbf{v}_1 = \mathbf{r}_0/\beta$
2: for $j = 1, ..., k$ until convergence, do
3:     $\mathbf{w}_j := \mathbf{A}\mathbf{v}_j$
4:     for $i = 1, ..., j$, do
5:         $h_{ij} := (\mathbf{w}_j, \mathbf{v}_i)$
6:         $\mathbf{w}_j := \mathbf{w}_j - h_{ij}\mathbf{v}_i$
7:     end for
8:     $h_{j+1,i} = \|\mathbf{w}_j\|_2$. If $h_{j+1,i} = 0$ set $k := j$ and exit loop
9:     $\mathbf{v}_{j+1} = \mathbf{w}_j/h_{j+1,i}$
10: end for
11: Define the $(k+1)\times k$ Hessenberg matrix $H_k = \{h_{ij}\}_{1\le i\le k+1, 1\le j\le k}$
12: Compute $\mathbf{y}$, the minimizer of $\|\beta\mathbf{e}_1 - \mathbf{H}_k\mathbf{y}\|_2$, and $\mathbf{x}_k = \mathbf{x}_0 + \mathbf{V}_k\mathbf{y}$
```

the upper Hessenberg matrix into an upper triangular system using the Givens rotations and then solving this triangular system for $\mathbf{y}$ (the minimizer of (3.18)). The Givens rotations are applied progressively at each step of the GMRES algorithm; the benefit of this approach is that it gives as a byproduct the norm of the actual residual at each step without additional arithmetic operations. As algorithm 3.3 does not explicitly provide the approximate solution at each step, it's difficult to determine when to stop. Using the Givens rotations, we have the norm of the residual available at hand to decide when to terminate. For a more detailed discussion on this issue, we refer to [118, section 6.5.4].

For general matrices, it is impossible to give convergence bounds similar to the one given in theorem 3.2.2 for the SPD case. Here we mention an analogous result for diagonalizable nonsymmetric matrices. Let $\mathcal{P}_k$ be the space of all polynomials of degree less than k and let $\sigma = \{\lambda_1, ..., \lambda_n\}$ be the spectrum of $\mathbf{A}$ with eigenvalues arranged in ascending order.

Theorem 3.2.4. *Let $\mathbf{A}\in\mathbb{R}^{n\times n}$ be a nonsymmetric and diagonalizable matrix with spectral decomposition $\mathbf{A} = \mathbf{X}\Lambda\mathbf{X}^{-1}$. Here $\mathbf{X} = \{\mathbf{x}_1, ..., \mathbf{x}_n\}$ be the right eigenvectors of $\mathbf{A}$ and $\Lambda = diag\{\lambda_1, ..., \lambda_n\}$. Let*

$$\epsilon^{(k)} = \min_{\substack{p\in\mathcal{P}_k \\ p(0)=1}} \max_{\lambda_i\in\sigma} |p(\lambda_i)|.$$

Then at the k^{th} iteration, GMRES produces the residual that satisfies the following inequality

$$\|\mathbf{r}_k\|_2 \le \kappa(\mathbf{X})\epsilon^{(k)}\|\mathbf{r}_0\|_2, \tag{3.19}$$

where $\kappa(\mathbf{X}) := \|\mathbf{X}\|_2\|\mathbf{X}^{-1}\|_2$ is the condition number of $\mathbf{X}$. Moreover, if all the eigenvalues lie inside a circle centered at $C\in\mathbb{R}$ with $C>0$ and having radius

R with $C > R$, then

$$\epsilon^{(k)} \leq \left(\frac{R}{C}\right)^k. \tag{3.20}$$

Proof. see [121] □

If $\mathbf{A}$ is SPD then $\kappa(\mathbf{X}) = 1$, for more general matrices its value is not known and expensive to compute. If $\kappa(\mathbf{X})$ is very large, then inequality (3.19) is not useful [67]. We can see that in (3.20), the ratio R/C will be smaller if the eigenvalues are more clustered (small R) and are away from the origin (large C). This suggests that GMRES will have faster convergence rates if the eigenvalues of $\mathbf{A}$ are clustered away from the origin. It is important to note that for GMRES convergence rates, eigenvalue distribution is much more important than the condition number of the matrix.

GMRES has some very nice features which make it a popular solution algorithm for nonsymmetric matrices. It is a stable method and no breakdowns occur; if $h_{j+1,i} = 0$ then $\mathbf{x}_k = \mathbf{x}$ and the solution is reached. The scheme satisfies the optimality property (3.14) and as a consequence has monotone convergence behavior. This is true since $\mathbf{r}_j$ is minimized over $\mathcal{K}_j$ and as $\mathcal{K}_{j+1} \supset \mathcal{K}_j$, the minimization over a larger subspace will result in smaller residual norm ($\|\mathbf{r}_{j+1}\| \leq \|\mathbf{r}_j\|$). GMRES, like CG, also exhibit the superlinear convergence behavior [144].

The disadvantage of GMRES is that it involves long recurrences, as the Arnoldi iteration requires all the previous k vectors for orthogonalization at the k^{th} iteration. Hence the work and memory requirements increase prohibitively for an increasing number of iterations. To avoid the excessive storage and computational costs, GMRES is restarted after m iterations, using the last approximate solution as an initial solution for next restart. The restarted GMRES is usually denoted by GMRES(m). However, restarting ruins many of the nice features of full GMRES, like GMRES(m) does not satisfy the minimization property as a whole and superlinear convergence behavior is inhibited [144]. The convergence behavior of restarted GMRES in many applications is very sensitive to the value of m, and an inappropriate choice of m may lead to the stagnation of the GMRES[75].

We like to mention here another variant of GMRES method, called *flexible GMRES* (FGMRES), which allows changing the preconditioner at each step. This flexibility allows the use of any other iterative solver as a preconditioner to GMRES (e.g., GMRES itself), and this feature can be exploited to build efficient iterative methods, possibly multilevel techniques. An essential aspect of FGMRES is that, like standard GMRES, it satisfies the optimality condition (3.14). FGMRES algorithm 3.4 can be implemented by doing a minimal modification to the preconditioned version of standard GMRES algorithm. Precisely speaking there is no additional mathematical cost involved but the memory cost doubles as in FGMRES an extra set of vectors need to be stored.

Algorithm 3.4 FGMRES algorithm

```
1: Compute $\mathbf{r}_0 = \mathbf{b} - \mathbf{A}\mathbf{x}_0$, $\beta = \|\mathbf{r}_0\|_2$, and $\mathbf{v}_1 = \mathbf{r}_0/\beta$
2: for $j = 1, ..., k$ do
3:     $\mathbf{z}_j = \mathbf{M}_j^{-1}\mathbf{v}_j$
4:     $\mathbf{w}_j := \mathbf{A}\mathbf{z}_j$
5:     for $i = 1, ..., j$, do
6:         $h_{ij} := (\mathbf{w}_j, \mathbf{v}_i)$
7:         $\mathbf{w}_j := \mathbf{w}_j - h_{ij}\mathbf{v}_i$
8:     end for
9:     $h_{j+1,i} = \|\mathbf{w}_j\|_2$.
10:    $\mathbf{v}_{j+1} = \mathbf{w}_j/h_{j+1,i}$
11: end for
12: Define $\mathbf{Z}_k := [\mathbf{z}_1, ..., \mathbf{z}_k]$,and the $(k+1) \times k$ Hessenberg matrix $H_k = \{h_{ij}\}_{1\leq i\leq k+1, 1\leq j\leq k}$
13: Compute $\mathbf{y}$, the minimizer of $\|\beta\mathbf{e}_1 - \mathbf{H}_k\mathbf{y}\|_2$, and $\mathbf{x}_k = \mathbf{x}_0 + \mathbf{Z}_k\mathbf{y}$
14: If satisfied Stop, else set $\mathbf{x}_0 := \mathbf{x}_k$ and go to 1.
```

Biconjugate Gradient Stabilized (Bi-CGSTAB) method

Biconjugate gradient stabilized method proposed by van der Vorst [143] belongs to the family of Krylov subspace solvers which solve nonsymmetric systems using Lanczos biorthogonalization. Other famous algorithms that belong to this family are Biconjugate gradient (Bi-CG) (see [84] and [58]) and Conjugate gradient squared (CGS) [126]. Instead of using the orthogonal sequence of Krylov vectors (as used by CG, CGNR, and GMRES), these methods rely on a pair of mutually orthogonal (biorthogonal) Krylov subspaces to compute an approximate solution. The drawback of this approach is that the solvers do not fulfill any optimality (or minimization) property and their convergence behavior may be quite irregular, sometimes leading to the breakdown of the algorithm. Bi-CGSTAB is an improved variant of this family developed with the goal to *stabilize* or *smooth* this irregular convergence behavior. The advantage of Bi-CGSTAB algorithm 3.5 is that it uses short recurrences and works well for unsymmetric systems. Contrary to GMRES it requires only six auxiliary vectors to be stored in memory and requires two matrix-vector products and four inner products per iteration.

Remark 3.2.5. *Krylov iteration methods are very robust, but their convergence depends on the condition number of matrix* $\mathbf{A}$*, which for the model problems that we investigate in turn depends on mesh width* h*. Hence the convergence rate of these methods slows down with the refinement of the mesh. A remedy for this problem is the well established multigrid methods.*

3.2.3 Multigrid methods

Geometric multigrid (GMG) methods are one of the best-known methods for solving linear systems arising from discretization of (elliptic) partial differential equations. Their computational complexity is $\mathcal{O}(N)$, and they give convergence rates that are independent of the mesh refinements. The GMG comprise the

Algorithm 3.5 BiCGSTAB algorithm

```
1: Compute $\mathbf{r}_0 = \mathbf{b} - \mathbf{A}\mathbf{x}_0$, for some initial guess $\mathbf{x}_0$    ▷ initialization
2: Choose $\hat{\mathbf{r}}_0$ such that $(\hat{\mathbf{r}}_0, \mathbf{r}_0) \neq 0$, e.g., $\hat{\mathbf{r}}_0 = \mathbf{r}_0$
3: $\mathbf{p}_0 = \mathbf{r}_0$
4: for $k = 0, 1, ...$ until convergence do
5:     $\mathbf{w}_k = \mathbf{A}\mathbf{p}_k$
6:     $\alpha_k = \frac{(\mathbf{r}_k, \hat{\mathbf{r}}_0)}{(\mathbf{w}_k, \hat{\mathbf{r}}_0)}$
7:     $\mathbf{s}_k = \mathbf{r}_k - \alpha_k \mathbf{w}_k$
8:     $\mathbf{t}_k = \mathbf{A}\mathbf{s}_k$
9:     $\gamma_k = \frac{(\mathbf{t}_k, \mathbf{s}_k)}{(\mathbf{t}_k, \mathbf{t}_k)}$
10:    $\mathbf{x}_{k+1} = \mathbf{x}_k + \alpha_k \mathbf{p}_k + \gamma_k \mathbf{s}_k$    ▷ update solution
11:    $\mathbf{r}_{k+1} = \mathbf{s}_k - \gamma_k \mathbf{t}_k$    ▷ update residual
12:    $\beta_k = \frac{(\mathbf{r}_{k+1}, \hat{\mathbf{r}}_0)}{(\mathbf{r}_k, \hat{\mathbf{r}}_0)} \times \frac{\alpha_k}{\gamma_k}$
13:    $\mathbf{p}_{k+1} = \mathbf{r}_{k+1} + \beta_k(\mathbf{p}_k - \gamma_k \mathbf{w}_k)$
14: end for
```

following ingredients:

- A hierarchy of grids as shown in figure 3.1
- Smoother (usually a basic iterative method)
- Grid transfer operators (Restriction and Prolongation)
- Coarse grid solver (mostly a sparse direct solver)

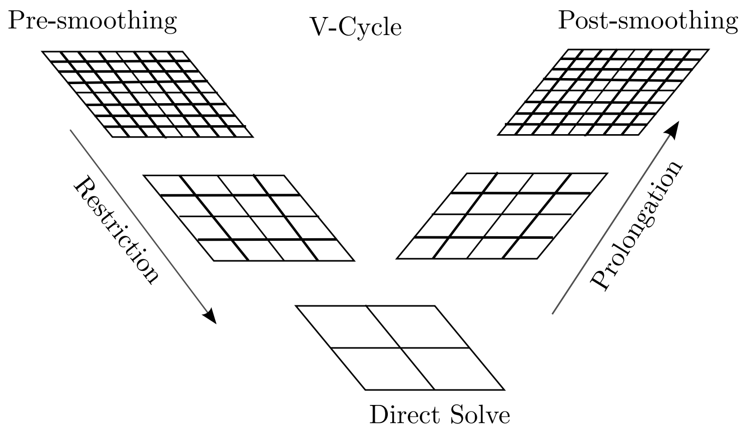


Figure 3.1: Multigrid V-cycle

The main idea of MG is to accelerate the convergence of basic relaxation schemes presented in section 3.2.1. These schemes rapidly remove highly oscillating errors in the solution, however, after the first few iterations, the error field is smoothed out and the convergence rate reduces significantly. MG lets the relaxation schemes do what they are good at only: remove the high-frequency error components by applying few relaxation iterations at the finer grid. This process is called *smoothing*, and the relaxation scheme itself is called *smoother*. Next, MG transfers the residual from the fine grid to the coarser grid by applying the *restriction operator*. The restriction operator uses the nodes from the fine grid and gathers information to the coarse grid using weighted averaging. As the smooth error function can be well approximated on the coarser grid without loss of essential information, the smooth error is calculated by solving the related coarser linear system.

The coarse grid solution is then projected back to the fine grid using the *prolongation operator*, for correction of the fine grid approximation. Hence, the prolongation operator does the opposite to that of restriction operator and spreads information from the coarse mesh to fine mesh. We use the standard finite element based grid transfer routines, that work on each solution component individually, to perform the grid transfer operations. The elementwise prolongation for the bilinear Q_1 finite element functions is shown schematically in figure 3.2. On the coarse mesh are shown the weights of the corresponding d.o.f's that are used to evaluate the d.o.f on the fine mesh. Each node in the fine mesh coincides with either the vertex, edge midpoint or the quadrilateral midpoint in the coarse mesh.

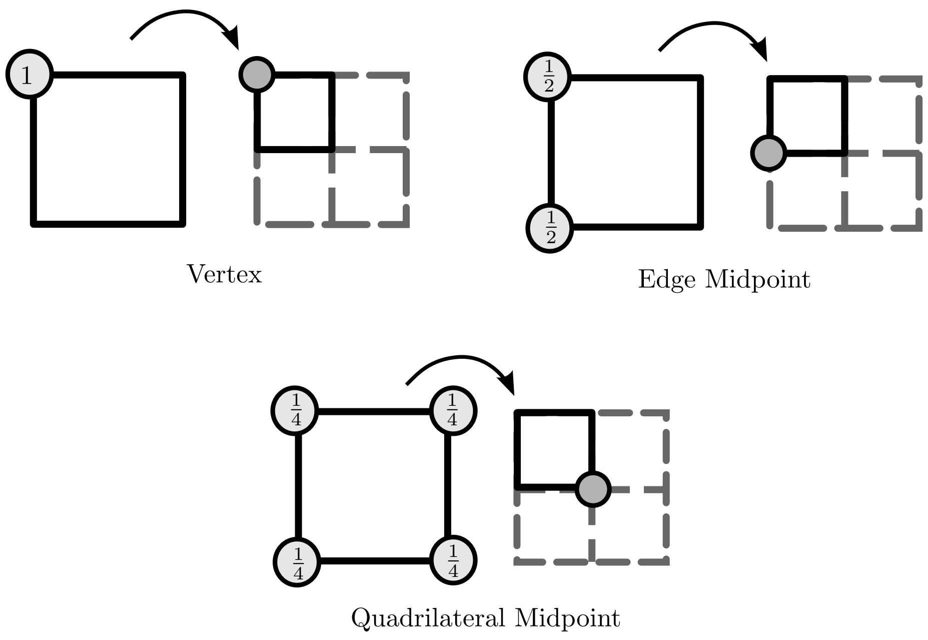


Figure 3.2: Schematic representation of prolongation in Q_1 element.

The coarse grid solve in algorithm 3.6 can be carried out using sparse direct solvers discussed in section 3.1. However, if the coarse grid problem is still large, direct solve can be very expensive or even impossible due to huge memory requirements. As the low-frequency errors are high-frequency errors on the coarser grid, the *multigrid* does not stop at second level and extends the two-grid method to more grid hierarchies. The way grid hierarchies are reached by multigrid is determined by the *multigrid cycles*. Figure 3.3 shows the three

Algorithm 3.6 Two-grid algorithm

```
1: Choose $\mathbf{x}_0^h$
2: Pre-smoothing:   $[\mathbf{x}_{i+\frac{1}{3}}^h, \mathbf{r}_{i+\frac{1}{3}}^h] = smooth(\mathbf{A}^h, \mathbf{x}_0^h, \mathbf{b}^h, \omega, \nu_1)$
3: Restriction:   $\mathbf{r}_{i+\frac{1}{3}}^H = \mathbf{I}_h^H \mathbf{r}_{i+\frac{1}{3}}^h$        $\mathbf{I}_h^H : \mathcal{G}^h \mapsto \mathcal{G}^H$
4: Coarse grid solve:   $\mathbf{e}^H = (\mathbf{A}^H)^{-1} \mathbf{r}_{i+\frac{1}{3}}^H$
5: Prolongation:   $\mathbf{e}^h = \mathbf{I}_H^h \mathbf{e}^H$        $\mathbf{I}_H^h : \mathcal{G}^H \mapsto \mathcal{G}^h$
6: Defect correction:   $\mathbf{x}_{i+\frac{2}{3}}^h = \mathbf{x}_{i+\frac{1}{3}}^h + \mathbf{e}^h$
7: Post-smoothing:   $[\mathbf{x}_{i+1}^h, \mathbf{r}_{i+1}^h] = smooth(\mathbf{A}^h, \mathbf{x}_{i+\frac{2}{3}}^h, \mathbf{b}^h, \omega, \nu_2)$
```

ν_1, ν_2	: number of pre- and post-smoothing steps
ω	: relaxation parameter

famous multigrid cycles namely V, W and F cycles. The type of the multigrid cycle along with other multigrid components play a vital role in the convergence behavior of MG method. If the geometric data of problem is not available or

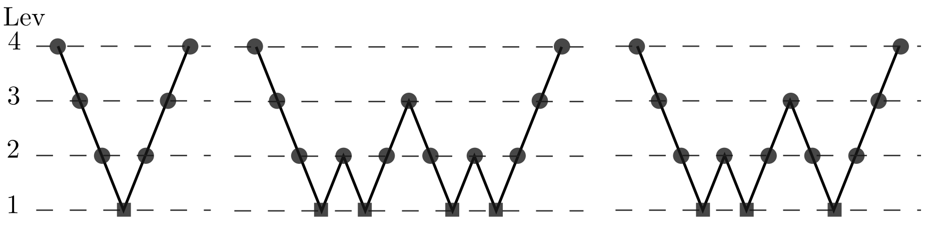


Figure 3.3: V-,W- and F-Multigrid cycles. ●– smoothing, ■– coarse grid solve

if the geometric complexity of the problem prohibits the use of GMG, then algebraic multigrid (AMG) methods can be used, which construct the level hierarchies directly from the system matrix. For more details on AMG methods we refer the interested readers to [128], [20], and [116].

3.3 Node reordering strategies

Reordering the node numbers can have a significant effect on the performance of solution methods for sparse linear systems. Renumbering can dramatically reduce the fill-ins, and thus considerably boost the performance of sparse direct solvers. Incomplete LU and Gauß-Seidel method are also quite often used as a preconditioner to Krylov subspace methods and as a smoother to multigrid for problems arising from numerical discretization of fluid flow problems. It has been widely reported in the literature that performance of both the schemes can be significantly enhanced in certain flow situations by choosing the correct numbering scheme (cf. [133, 72, 96]), as the renumbering makes them 'more exact'. ILU has a twofold benefit with a fill-in reducing reordering: First, ILU will drop fewer terms and likely produce more accurate L and U factors; secondly if ILU with a high level of fill-in is used, it will be more efficient. For Gauß-Seidel, the orderings which follow the flow of information are helpful, for example, numberings aligned with the flow stream in case of convection dominated flows

[72, 15]. Reorderings can also unravel the parallelism in linear solvers and are the most important ingredient in parallel implementations. (cf. [51, 42]).

Many renumbering techniques have been proposed, but unfortunately, there is no one the best performer for all situations. It is possible to make a numerical setup in which one scheme outperforms others, while in another setup it performs worst. Here we mention few grid ordering techniques:

1. Cuthill- Mckee ordering (CMK)
2. Reverse Cuthill-Mckee ordering (RCM)
3. Two level ordering (TLO)
4. Rowwise or x-coordinate sorting (GR)
5. Columnwise or y-coordinate sorting (GC)

The Cuthill-Mckee algorithm by E. Cuthill and J. McKee [33], is an automatic nodal numbering scheme that ensures significant bandwidth[4] improvement for a wide range of problems. It is a *level set* based technique that traverses the adjacency graph of a sparse matrix by level sets. It starts with level 1 consisting of one node (more nodes also possible), which can be the node with the lowest degree[5], and numbers it as the first vertex. The next level set contains the adjacent nodes[6] of the previous level set. The nodes in a level set are numbered from lowest to the highest degree for each neighbor node from the previous level. This process is repeated until all the nodes are numbered. As illustrated in figure 3.4, the finite element two-level ordering produce a matrix with a bandwidth equal to 7, whereas, the CMK ordering reduce the bandwidth to 3. Reduced bandwidth implies reduced fill-ins, and the scheme can be useful for direct methods and ILU based solvers. In [133], the author has shown for driven cavity stokes problem, MG solver with ILU as a smoother has better performance with Cuthill-Mckee renumbering.

Reverse Cuthill-Mckee ordering by A. George [64], is very similar to the CMK but numbers the grid points in reverse order. George noticed that *reversing* the Cuthill-Mckee ordering produces same bandwidth but yields the nonzero-pattern inside the bandwidth that better suits the Gaussian elimination based solvers. In [32], the superior performance of reverse scheme for various problems has been reported. In [87], authors have proved that compared with the CMK, the reverse ordering is *always* at least as good, from storage and operation counts viewpoint. Therefore, reverse Cuthill-Mckee is a more popular ordering choice among the scientific community. Minimum degree (MD)[62] and nested dissection (ND)[63] are two other popular reordering techniques primarily used in sparse direct solvers to reduce the fill-in, which are not bandwidth reducing schemes.

Two level ordering is used in `FEATFLOW`[7] (Finite Element Analysis and Tools for Flow problems, Version 2) software, for numbering the nodes in mesh refinements. In TLO, on the refined mesh, grid points from old mesh retain their

[4]Bandwidth of a matrix $\mathbf{A} \in \mathbb{R}^{n\times n}$, is defined as $\max_{1\leq i,j\leq n}\{|i-j|, a_{ij} \neq 0\}$
[5]The degree of node j is the number of edges meeting at j
[6]Two nodes are adjacent if they are connected by an edge
[7]For more information visit www.featflow.de

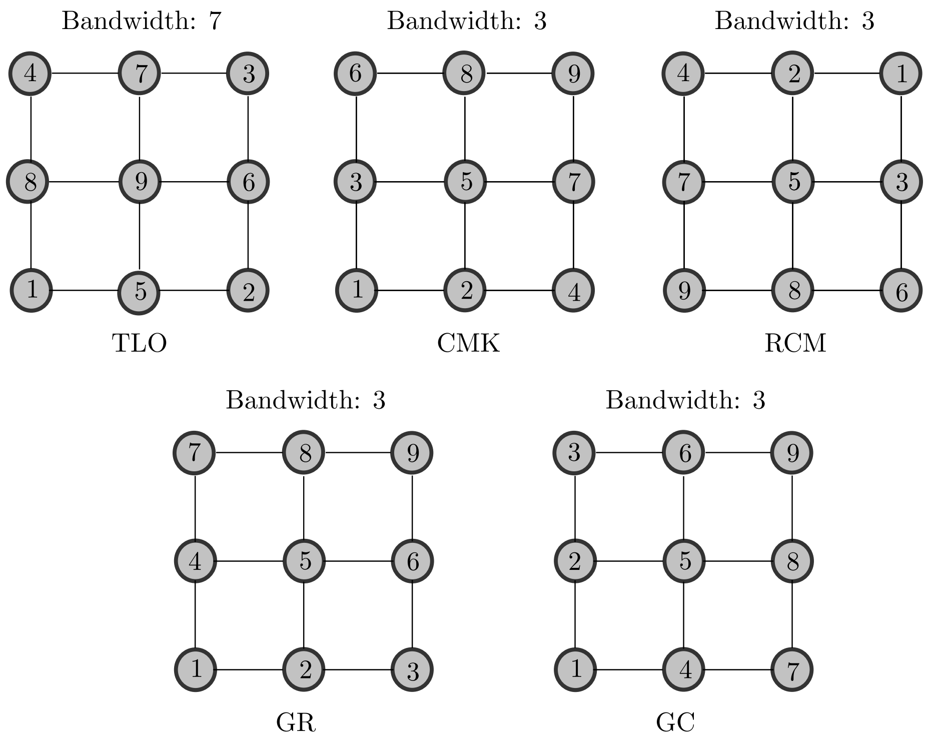


Figure 3.4: Node numbering of unit square mesh for different reordering techniques

numbers, while new vertices get new numbers. This renumbering is helpful for elementwise application of grid transfer routines in multigrid but produces large bandwidths.

Rowwise and *columnwise* sortings are *geometry-based* reorderings, which require the coordinate information of nodes. Although these techniques cannot be very useful for general applications, however, for some special situations they can beat other sophisticated schemes like RCM. For instance, for a convection dominated flow in a channel with convection in the x-direction and without any vortices involved, the value at a particular node is only influenced by the upstream nodes only. In such a case, if we use the upwind stabilization and use rowwise renumbering then the resulting coefficient matrix will have nonzeros only in the lower triangular part, and Gauß-Seidel or ILU will solve the problem in only one step. However, in most practical situations, there computational costs and memory requirements are very high, and also its difficult to use them for unstructured grids.

4 Multilevel Krylov subspace method

As the name suggests, the main focus of this chapter is the multilevel Krylov subspace method (MLKM), which incorporates components from the Krylov subspace iterative methods and multigrid methods. However, before embarking on the idea of MLKM, we start this chapter with the discussion of an important concept of *preconditioning*, that continues to play a pivotal role in the construction of efficient and reliable Krylov subspace solvers for handling challenging real life computational problems. The performance of Krylov subspace methods depends on the condition number $\kappa(\mathbf{A})$ of the coefficient matrix $\mathbf{A}$ of the linear system, and on the clustering of the eigenvalues of the coefficient matrix. We discuss two classes of preconditioners: first *the traditional preconditioners*, aimed at improving the condition number of the coefficient matrix; second *the eigenvalue distribution preconditioners*, aimed at the clustering of eigenvalues of the coefficient matrix.

Next, we review in detail the MLKM solver proposed by Nabben and Erlangga, which is primarily based on the concept of clustering the eigenvalues around maximum eigenvalue of the coefficient matrix, however, it also uses the traditional preconditioner to further improve the overall convergence behavior of the solver. After that, we explain our implementation of MLKM solver in the context of FEATFLOW solver and discuss how it differs from MLKM algorithm of Nabben and Erlangga. At the end of the chapter, we briefly compare the MLKM solver with the multigrid solver and explain how both the solvers are different from each other.

4.1 Preconditioning

In chapter 3, we have discussed that the convergence rate of Krylov subspace solvers, for both symmetric and nonsymmetric linear systems, is strongly dependent on the spectral properties of the system matrix. Conjugate gradient method, for example, works best when spectral condition number of matrix $\kappa(\mathbf{A})$ is small and/or eigenvalues are clustered around one [13]. Krylov subspace methods for general matrices, like GMRES, also have better convergence if the eigenvalues of a matrix are clustered away from the origin. However, the linear systems originating from the discretization of PDEs, do not possess such a desired eigenvalue distribution in most cases. *Preconditioning* is usually applied to these systems with the objective of having transformed systems with same solution as the original systems but with more favorable spectral properties for the iterative methods. Let $\mathbf{M}$ be an invertible matrix (called preconditioner), then the preconditioned linear system

$$\mathbf{M}^{-1}\mathbf{A}\mathbf{x} = \mathbf{M}^{-1}\mathbf{b}, \tag{4.1}$$

has the same solution as $\mathbf{A}\mathbf{x} = \mathbf{b}$, but is expected to be easily solvable by Krylov subspace methods. Please note that in the Krylov solver, the product $\mathbf{M}^{-1}\mathbf{A}$ is never computed explicitly, as it would be expensive and the resulting matrix would be dense. Instead, matrix-vector product and solution of linear system of the form $\mathbf{M}\mathbf{z} = \mathbf{w}$ are solved.

Preconditioning involves extra costs: a preconditioner setup or construction cost, and a preconditioner application cost at every iteration of the iterative solver. A good preconditioner is one for which the improvement in convergence rate is good enough to justify these extra costs. To be a good preconditioner $\mathbf{M}$ should generally possess the following properties:

- $\mathbf{M}$ is (spectrally) as close to $\mathbf{A}$ as possible.
- $\mathbf{M}$ should be cheap to construct and apply.

First property implies that the condition number of the preconditioned matrix should be close to one and the preconditioned iteration should converge faster. Second property ensures that each preconditioned iteration is economical. However, if we try to achieve one property, other property is compromised. Therefore, while constructing a preconditioner, there is always a trade-off between the two properties.

In equation (4.1), the preconditioner is applied on the left side of the matrix $\mathbf{A}$; this is called *left preconditioning.* Alternatively, one can also apply $\mathbf{M}$ from the right, in which case it is called *right preconditioning*:

$$\mathbf{A}\mathbf{M}^{-1}\mathbf{u} = \mathbf{b}, \qquad \mathbf{x} = \mathbf{M}^{-1}\mathbf{u}. \tag{4.2}$$

Split preconditioning can also be applied:

$$\mathbf{M}_1^{-1}\mathbf{A}\mathbf{M}_2^{-1}\mathbf{u} = \mathbf{M}_1^{-1}\mathbf{b}, \qquad \mathbf{x} = \mathbf{M}_2^{-1}\mathbf{u}. \tag{4.3}$$

Matrices $\mathbf{M}^{-1}\mathbf{A}$, $\mathbf{A}\mathbf{M}^{-1}$, and $\mathbf{M}_1^{-1}\mathbf{A}\mathbf{M}_2^{-1}$ are all similar and have same eigenvalues. For CG method, the convergence will be same in all cases. For GMRES method, if the preconditioned matrix is far from normal, convergence rates can vary greatly depending on if the preconditioning is applied from left or right (see [118, p. 255] for a detailed discussion).

Customarily, the preconditioners can be categorized based on their construction approach as follows:

- Physics-based preconditioners (application specific)
- Coefficient matrix based preconditioners (general purpose)

Physics-based preconditioners (also called PDE based preconditioners) are based on the simpler nearby PDEs which are easy to solve. For instance, Carey et al. [28], have used full factorization of Stokes problem as a preconditioner to Krylov solvers for solving Newtonian and Non-Newtonian flows. In [36], Dahl and Wille have used Stokes based ILU factorization as a preconditioner to the Bi-CGSTAB solver for solving the Navier-Stokes equations. For more information on the physics-based preconditioners, we refer the interested readers to [14, 8, 150], and the references therein. The physics based-preconditioners are application specific and can not be applied to general problems. Although these preconditioners are at times very effective, they require a complete application specific knowledge. They are very sensitive to the details of the problem, and small changes in the problem can drastically affect the performance of the solver. Carey et al. have reported that Stokes based preconditioner worked fine for driven cavity flow problem at low Reynolds numbers, but for relatively higher Reynolds numbers ($Re = 300$ and more) the performance was poor compared with the frontal solver.

Matrix-based preconditioners are also called algebraic preconditioners and are built from the information available in the coefficient matrix. They are easier to develop and apply, are universal and achieve reasonable efficiency on a wide range of problems. In the following section, we discuss some matrix-based preconditioners which we will be using in our thesis. Brenzi has given an excellent survey of algebraic preconditioning techniques in his article[13], and we refer the interested readers to this manuscript for more details.

4.1.1 Basic iterative methods as preconditioners

From the preconditioned system (4.1) we have:

$$(\mathbf{I} - \mathbf{M}^{-1}(\mathbf{M} - \mathbf{A})\mathbf{x} = \mathbf{M}^{-1}\mathbf{b},$$

and in iteration form, we may write

$$\begin{aligned}\mathbf{x}_{k+1} &= (\mathbf{I} - \mathbf{M}^{-1}\mathbf{A})\mathbf{x}_k + \mathbf{M}^{-1}\mathbf{b} \\ &= \mathbf{x}_k + \mathbf{M}^{-1}\mathbf{r}_k.\end{aligned}$$

Above equation is the same as preconditioned Richardson iteration (3.5). This means that application of one iteration of the basic iterative methods is the same as preconditioning of linear system. Hence using $\mathbf{M}^{JAC}$, $\mathbf{M}^{GS}$ and $\mathbf{M}^{SOR}$ in preconditioned system (4.1) will result in Jacobi, Gauss-Seidel, and Successive over-relaxation preconditioners:

Jacobi preconditioning

$$\mathbf{D}^{-1}\mathbf{A}\mathbf{x} = \mathbf{D}^{-1}\mathbf{b}. \tag{4.4}$$

Gauss-Seidel preconditioning

$$(\mathbf{D} - \mathbf{E})^{-1}\mathbf{A}\mathbf{x} = (\mathbf{D} - \mathbf{E})^{-1}\mathbf{b}. \tag{4.5}$$

Successive over-relaxation preconditioning

$$(\mathbf{D} - \tilde{\omega}\mathbf{E})^{-1}\mathbf{A}\mathbf{x} = (\mathbf{D} - \tilde{\omega}\mathbf{E})^{-1}\mathbf{b}. \tag{4.6}$$

When applying the preconditioner to the symmetric matrix, it is important not to lose the symmetry in preconditioned system matrix, so that Krylov subspace methods for symmetric matrices (like CG) can be applied. Aforementioned basic iterative preconditioners when applied would not retain the symmetry, and therefore, symmetric version of SOR called *symmetric successive over-relaxation* (SSOR) preconditioner is used. In 1972, Axelsson [6] published an extensive study of SSOR preconditioning for accelerating the convergence of conjugate gradient method [see also [118]].

An advantage of the basic iterative preconditioners is that they do not involve any construction cost, and they do not require an extra matrix for their storage.

4.1.2 ILU Preconditioner

Incomplete LU (ILU) factorization preconditioner is one of the most widely used algebraic preconditioners. The idea of incomplete factorization was first introduced by Buleev [27] and by Varga [139] independently in the late 1950s. Nonetheless, Meijerink and van der Vorst were the first ones who identified the potential of ILU as a preconditioner in their remarkable work [95]. They observed that for a class of M-matrices, the convergence rate of conjugate gradient method could be significantly improved if it is preconditioned with incomplete factorization. Influenced by this phenomenal work, since then ILU preconditioning has been used by many people for solving the discretized systems arising from various PDEs. Munksgaard [104] formed incomplete factorization using different dropping and reorderings strategies and used it as a preconditioner to CG to solve sparse symmetric positive definite matrices. Manteuffel [93] used incomplete factorization preconditioning to solve large sparse symmetric linear systems that arise from the application of finite element methods. Dutto [50] applied the idea to solve the compressible Navier-Stokes equation problem. An excellent introduction to ILU and its variants can be found in the books by Hackbusch [69] and Axelsson [7].

ILU Idea

Full LU decomposition of large sparse systems results in **L** and **U** matrices which are remarkably less sparse. Even if the fill-in reducing reorderings are used, the related computational and memory costs of resulting factorization are often so enormous that the method becomes impractical to use. The basic idea of incomplete LU factorization is straightforward: drop out some of the fill-ins that occur during the elimination process to preserve sparsity. Instead of exact factorization $\mathbf{A} = \mathbf{LU}$, this results in incomplete (or approximate) factorization

$$\mathbf{A} = \tilde{\mathbf{L}}\tilde{\mathbf{U}} - \mathbf{R},$$

where $\tilde{\mathbf{L}}$ and $\tilde{\mathbf{U}}$ are the incomplete lower and upper triangular matrices respectively, and $\mathbf{R}$ is called the *residual* matrix containing the discarded entries. Various ILU algorithms have been proposed which differ on the dropping criteria of the fill-ins. ILU(l) algorithms allow the fill-ins based on their position in the matrix, whereas ILU(t) allow the fill-ins based on their size. Unlike basic iterative methods preconditioners, all ILU based preconditioners involve the construction cost and require extra matrix for their storage.

In ILU(l), l stand for the level of fill-in. In ILU(0), level of fill-in is zero, i.e., ILU(0) takes the sparsity pattern of matrix $\mathbf{A}$, and does not allow any fill-in. Let $\mathbf{S}$ be defined as

$$\mathbf{S} = \{(i, j)|i \neq j; 1 \leq i, j \leq n\},$$

then the sparsity pattern $\mathbf{S}_0$ for ILU(0) is a subset of $\mathbf{S}$

$$\mathbf{S}_0 = \{(i, j)|\ a_{ij} \neq 0\}.$$

The ILU(0) algorithm is defined constructively in Algorithm 4.1. ILU(0) precon-

Algorithm 4.1 ILU(0) algorithm

```
1: for $i = 2, ..., n$ do
2:     for $k = 1, ..., i-1$ and for $(i, k) \in \mathbf{S}_0$ do
3:         Compute $a_{ik} = a_{ik}/a_{kk}$
4:         for $j = k+1, ..., n$ and for $(i, j) \in \mathbf{S}_0$ do
5:             Compute $a_{ij} := a_{ij} - a_{ik}a_{kj}$
6:         end for
7:     end for
8: end for
```

ditioning has been reported to be very effective for PDEs resulting in M-matrices or diagonally dominant matrices [13]. However, for more difficult problems (such as highly nonsymmetric matrices), ILU(0) may result in an inaccurate approximation of $\mathbf{A}$ and give no significant improvement in convergence rates [29]. For such cases, more accurate incomplete factorizations that allow some fill-ins may work better. In ILU(l) a hierarchy of more accurate factorizations is obtained based on the concept of *level of fill*. A level of fill is assigned to each matrix element that is processed during the factorization process. Here we follow the level of fill definition of Saad [118, Definition 10.5, page 298]. The initial level of fill of a sparse matrix entry a_{ij} is defined by

$$lev_{ij} = \begin{cases} 0 & \text{if } a_{ij} \neq 0 \text{ or } i = j, \\ \infty & \text{otherwise.} \end{cases}$$

Each time a_{ij} is modified in algorithm 4.1, its level of fill must be updated by

$$lev_{ij} = \min\{lev_{ij}, lev_{ik} + lev_{kj} + 1\}.$$

Note that during the iteration process, the level of fill of an element never increases. Therefore, for nonzero elements of the original matrix $\mathbf{A}$, the level of fill will always remain zero during the whole elimination process.

Fill-ins are discarded based on the values of their level of fill. For ILU(1), all fill-ins whose level of fill is higher than 1 are discarded. The cost to construct and apply ILU(1) preconditioner are reasonable, and in many applications it gives a considerable improvement over ILU(0). In practical implementations, the symbolic factorization is done before numerical factorization, to assign the level of fills and to determine the structure of L and U factors.

ILU(l) with a lower level of fill is not robust for matrices which are far from being diagonally dominant, because the fill-ins being dropped are not of the smaller size and the resulting L and U factors do not approximate A very well. For higher levels of fill, the associated construction and application costs of ILU(l) increase rapidly, and are rarely used in practice. For many such cases, ILU(t) with droppings based on their size results in efficient preconditioner. However, the difficulty with this approach is to choose a good *drop tolerance* value, which usually is done by trial and error method and is very much problem dependent. Saad [117], in his ILUT algorithm, has combined both the position based and value based approaches. Modified ILU (MILU) is another variant of incomplete factorization that *compensates* the effect of dropping, by adding all the elements to be dropped in a row and subtract it from a diagonal entry in U (see [49, 68] for more details)

4.2 Preconditioning based on the spectral information

Preconditioners mentioned in the previous section are also called *traditional* preconditioners. These traditional preconditioners improve the condition number of the original matrix in many cases, however, condition number is not the only indicator of convergence improvement. It has been reported in [137], that convergence rates of the conjugate gradient can be significantly higher, if the eigenvalues of system matrix are clustered near one. Similarly, for GMRES it is well known that its convergence can be better if eigenvalues of the system matrix are clustered away from zero. The traditional preconditioners do not take into consideration the details of the eigenvalue spectrum of the matrix during their operation, and consequently, the spectrum of $\mathbf{M}^{-1}\mathbf{A}$ may still have many eigenvalues near zero. These near zero eigenvalues hamper the convergence of Krylov subspace methods. The convergence of Krylov subspace methods can be improved, if by some means during the iteration process, the components corresponding to these small eigenvalues can be removed from the residual vector.

Deflation is the technique that is used to deal with the problematic part of the spectrum in the (un)preconditioned linear system. There are essentially two ways to implement deflation techniques. In the first approach, called *augmentation* or *enrichment*, the eigenvectors related to the small eigenvalues are augmented to the Krylov subspace, then these eigenvectors will have no components in the residuals. Nicolaides [108] showed the convergence improvement of CG method using augmentation. Morgen [97, 99, 98] has shown that if at each GMRES restart, the approximate eigenvectors corresponding to the smallest eigenvalues are formed and added to the Krylov subspace, restarted GMRES considerably improves convergence rates and retains the residual optimality property. In the second approach, called *projection* or *deflation*, a projection matrix is constructed from the offending eigenvectors and is used as a preconditioner, which deflates the small eigenvalues to zero. We explain the construction and properties of the projection based deflation preconditioner in detail below.

4.2.1 Deflation preconditioner

For symmetric positive definite systems, Frank and Vuik in [60], have proposed a projection matrix that deflates smallest eigenvalues and used it as a preconditioner to CG. Here we call this a *two-level deflation preconditioner*. However, we aim to build a solver that can ultimately solve coupled Navier-Stokes equations, which are not SPD. Therefore, following [53], we define here the deflation preconditioner to solve the sparse linear system $\mathbf{Ax} = \mathbf{b}$ where $\mathbf{A} \in \mathbb{R}^{n\times n}$ is a general nonsingular matrix. To deflate m smallest eigenvalues to zero, the deflation preconditioners can be constructed as

$$\mathbf{P}_D = \mathbf{I} - \mathbf{AK}, \qquad \mathbf{Q}_D = \mathbf{I} - \mathbf{KA}, \qquad \mathbf{K} = \mathbf{ZE}^{-1}\mathbf{Y}^T, \qquad \mathbf{E} = \mathbf{Y}^T\mathbf{AZ}. \tag{4.7}$$

$\mathbf{P}_D$ and $\mathbf{Q}_D$ are the left and right deflation preconditioners respectively. It can be easily verified that $\mathbf{P}_D$ and $\mathbf{Q}_D$ are projectors since $\mathbf{P}_D^2 = \mathbf{P}_D$ and $\mathbf{Q}_D^2 = \mathbf{Q}_D$. Here $\mathbf{Z}$, $\mathbf{Y} \in \mathbb{R}^{n\times m}$ are full rank matrices, with $\mathbf{Z}$ called *deflation subspace* corresponding to the offending eigenvalues that we want to project out from the residual. The solution of a linear system preconditioned with deflation is carried out as follows. Decompose the solution vector into

$$\mathbf{x} = (\mathbf{I} - \mathbf{Q}_D)\mathbf{x} + \mathbf{Q}_D\mathbf{x}, \tag{4.8}$$

where

$$(\mathbf{I} - \mathbf{Q}_D)\mathbf{x} = \mathbf{ZE}^{-1}\mathbf{Y}^T\mathbf{Ax} = \mathbf{ZE}^{-1}\mathbf{Y}^T\mathbf{b}. \tag{4.9}$$

If $m \ll n$, then the matrix $\mathbf{E} \in \mathbb{R}^{m\times m}$ can be easily computed and inverted, and calculating (4.9) is not a problem. The main task is to compute the factor $\mathbf{Q}_D\mathbf{x}$ in (4.8), and this is done by solving the following system for $\tilde{\mathbf{x}}$

$$\mathbf{P}_D\mathbf{A}\tilde{\mathbf{x}} = \mathbf{P}_D\mathbf{b}, \tag{4.10}$$

using Krylov subspace solver for nonsymmetric systems (GMRES or Bi-CGSTAB) and then premultiplying the solution with $\mathbf{Q}_D$. The result is then added to (4.9) to get the solution $\mathbf{x}$. Since the solution $\tilde{\mathbf{x}}$ is an approximate solution coming from Krylov subspace solver, it may still have some components in the deflated space $\mathbf{Z}$, which is also the null space of $\mathbf{P}_D\mathbf{A}$. Therefore, the projected solution $\mathbf{Q}_D\tilde{\mathbf{x}}$ is used.

For a nonsingular matrix $\mathbf{A}$, let the spectrum of $\mathbf{A}$ is given by

$$\sigma(A) = \{\lambda_1, \lambda_2, ..., \lambda_n\},$$

with $|\lambda_i| \leq |\lambda_{i+1}|$, for $i = 1, ..., n$. The following theorem gives the spectrum of deflated matrix $\mathbf{P}_D\mathbf{A}$.

Theorem 4.2.1. *Let $\mathbf{A}$ be nonsingular and diagonalizable, and let $\mathbf{Z}$ and $\mathbf{Y}$ be the left and right eigenvectors corresponding to the first m eigenvalues of $\mathbf{A}$, chosen such that $\mathbf{Y}^T\mathbf{Z} = \mathbf{I}_m$, then*

$$\sigma(P_D A) = \{0, ..., 0, \lambda_{m+1}, ..., \lambda_n\}.$$

Proof. Here $\mathbf{E}{=}\mathbf{Y}^T\mathbf{AZ} = diag(\lambda_1, ..., \lambda_m) := \Lambda_m$ For $i = 1, ..., m$

$$\mathbf{P}_D\mathbf{AZ} = \mathbf{AZ} - \mathbf{AZE}^{-1}\underbrace{\mathbf{Y}^T\mathbf{AZ}}_{=\mathbf{E}} = 0.$$

For $i = m+1, ..., n$

$$\mathbf{P}_D\mathbf{A}\mathbf{z}_i = (\mathbf{I} - \mathbf{A}\mathbf{Z}\mathbf{E}^{-1}\mathbf{Y}^T)\mathbf{A}\mathbf{z}_i = \lambda_i\mathbf{z}_i - \mathbf{Z}\Lambda_m\Lambda_m^{-1}\lambda_i \underbrace{\mathbf{Y}^T\mathbf{z}_i}_{=0} = \lambda_i\mathbf{z}_i.$$

□

As can be seen, deflation with eigenvectors changes the first m eigenvalues to zero, while the rest of the eigenvalues remain unchanged. Also, note that the eigenvectors of $\mathbf{P}_D\mathbf{A}$ and $\mathbf{A}$ remain the same.

A Krylov method (like GMRES) builds solution for the problem 4.10 from the Krylov subspace

$$\mathcal{K}_k(P_DA, r_0) = \{r_0, P_DAr_0, (P_DA)^2r_0, ..., (P_DA)^{k-1}r_0\},$$

with the residual $\mathbf{r}_0 = \mathbf{P}_D(\mathbf{b} - \mathbf{A}\mathbf{x}_0)$. Since the null space of $\mathbf{P}_D\mathbf{A}$ never enters into the Krylov iteration, the zero eigenvalues do not effect the Krylov iteration. Therefore, we define the effective condition number of the deflated system 4.10 as

$$\kappa_{eff}(P_DA) = \frac{\lambda_n}{\lambda_{m+1}}, \tag{4.11}$$

which is less than the condition number of the linear system $\mathbf{A}$ and the Krylov subspace method is expected to perform better on this deflated system.

4.2.2 Balancing Preconditioner

Mandel [90] proposed a projection-like preconditioner for symmetric matrices, called *balancing preconditioner*, which has been widely used in domain decomposition methods. Here we discuss the balancing preconditioner very briefly, and refer to the literature for more details [91, 92, 46, 129, 111]. Again following [53], we write the balancing preconditioner for nonsymmetric systems as

$$\mathbf{P}_B = \mathbf{Q}_D\mathbf{P}_D + \mathbf{Z}\mathbf{E}^{-1}\mathbf{Y}^T. \tag{4.12}$$

Theorem 4.2.2. *Let* $\mathbf{A}$ *be nonsingular and diagonalizable, and let* $\mathbf{Z}$ *and* $\mathbf{Y}$ *be the left and right eigenvectors corresponding to the first* m *eigenvalues of* $\mathbf{A}$*, chosen such that* $\mathbf{Y}^T\mathbf{Z} = \mathbf{I}_m$*, then*

$$\sigma(P_BA) = \{1, ..., 1, \lambda_{m+1}, ..., \lambda_n\}.$$

Proof. See [53] □

The effect of balancing preconditioner on $\mathbf{A}$ is very similar to that of deflation preconditioner, with the exception that the first m eigenvalues are shifted to 1 instead of 0. Moreover, $\mathbf{P}_D\mathbf{A}$, $\mathbf{P}_B\mathbf{A}$ and $\mathbf{A}$ all have same eigenvectors.

Deflation and balancing preconditioners have been compared for symmetric positive definite systems in [106], and for nonsymmetric systems in [53]. In [106], it has been shown that CG applied to the deflated preconditioned system always has smaller A-norm of the error than that for CG applied to the system preconditioned with balancing preconditioner. Similarly, in [53], authors have shown that under certain conditions the GMRES with deflation preconditioner produces residuals whose 2-norm is always less than the 2-norm of residuals of GMRES with balancing preconditioner. Moreover, deflation preconditioning has less computational costs than the balancing preconditioning.

From equation 4.11, it is easy to infer that if we use a larger deflation subspace (larger m), we can have reduced effective condition number and consequently much better convergence rates for the Krylov subspace solvers. However, note that both deflation and balancing preconditioning involve inversion of $\mathbf{E} \in \mathbb{R}^{m\times m}$, which will be expensive for larger values of m. Nabben and Vuik [105] have shown that deflation preconditioning is sensitive to the inaccurate inversion of $\mathbf{E}$, which means that iterative methods cannot be used to invert $\mathbf{E}$ approximately. The demand for exact inversion of $\mathbf{E}$ limits the potential of deflation preconditioner. On the other hand, a big advantage of balancing preconditioner is that it is insensitive to the inaccurate solve of Galerkin matrix $\mathbf{E}$. This allows the use of larger deflation subspace and the corresponding large $\mathbf{E}$ matrix can be inverted approximately using iterative methods.

Deflation or balancing preconditioning can be used in combination with the traditional preconditioning, for which case the linear system becomes

$$\mathbf{P}_D\mathbf{M}^{-1}\mathbf{A}\mathbf{x} = \mathbf{P}_D\mathbf{M}^{-1}\mathbf{b} \qquad \text{or} \qquad \mathbf{P}_B\mathbf{M}^{-1}\mathbf{A}\mathbf{x} = \mathbf{P}_B\mathbf{M}^{-1}\mathbf{b} \tag{4.13}$$

4.3 Multilevel Krylov Subspace Method

Erlangga and Nabben in 2008 [54], have proposed a new projection-like method which, like balancing preconditioner, is insensitive to the inaccurate solve of Galerkin system $\mathbf{E}$, and has the computational demands similar to that of deflation preconditioner. The basic idea of this method is to shift small eigenvalues that are responsible for the slow convergence of Krylov subspace solvers to an a priori fixed constant, thus resulting in a more clustered spectrum. Shifting of eigenvalues to a nonzero constant allows the resultant coarse level system to be solved inexactly or iteratively by a few steps of Krylov method. To further improve the convergence of coarse level solve, the new shifting operator is again used as a preconditioner for a Krylov subspace method. Repeated application of the shifting operator as a preconditioner results in a multilevel method called *multilevel projection Krylov method* (MLKM) by the authors. Here we briefly explain the construction of MLKM and refer to authors article [54] for a more detailed discussion.

4.3.1 Deflation from an eigenvalue computation viewpoint

Deflation process has been used for quite a long in eigenvalue problem solving algorithms [147]. We first show here that deflation preconditioner (4.9) can be developed from the deflation process used in computing few of the smallest eigenvalues in eigenvalue computation. From this eigenvalue computation viewpoint, then we develop the stable abstract projection type preconditioner of MLKM algorithm that shifts small eigenvalues to a priori constant.

Consider a preconditioned linear system

$$\hat{\mathbf{A}}\hat{\mathbf{x}} = \hat{\mathbf{b}}, \tag{4.14}$$

with $\hat{\mathbf{A}} = \mathbf{M}_1^{-1}\mathbf{A}\mathbf{M}_2^{-1}$, $\hat{\mathbf{x}} = \mathbf{M}_2\mathbf{x}$, and $\hat{\mathbf{b}} = \mathbf{M}_1^{-1}\mathbf{b}$, with $\mathbf{M}_1$ and $\mathbf{M}_2$, be the nonsingular matrices. For left preconditioning, set $\mathbf{M}_1 = \mathbf{M}$ and $\mathbf{M}_2 = \mathbf{I}$; for right preconditioning set $\mathbf{M}_1 = \mathbf{I}$ and $\mathbf{M}_2 = \mathbf{M}$. For the purpose of analysis we assume that the eigenvalues of $\hat{\mathbf{A}}$, are all positive and real, and let the spectrum of $\hat{\mathbf{A}}$, denoted by $\sigma(\hat{A})$ is given by $\sigma(\hat{A}) = \{\lambda_1, ..., \lambda_n\}$ and $\lambda_i \leq \lambda_{i+1}, \forall i \in \mathcal{N}$. Power method is one of the simplest and oldest methods used for computing the extremal eigenvalues of a system [119]. The method starts with an arbitrary nonzero initial vector $\mathbf{v}_0$ and generates a sequence of vectors

$$\mathbf{v}_k = \frac{1}{\alpha_k}\mathbf{A}^k\mathbf{v}_0,$$

with α_k to be the element of $\mathbf{A}^k\mathbf{v}_0$ with the largest absolute value. For large values of k, α_k and $\mathbf{v}_k$ converge to the largest eigenvalue and largest eigenvector respectively. If the method is applied to $\hat{\mathbf{A}}^{-1}$, it will approximate the smallest eigenvalue λ_1 and corresponding eigenvector $\mathbf{z}_1$ of the matrix $\hat{\mathbf{A}}$. To find the next smallest eigenvalue in the spectrum, we have to first deflate λ_1 to zero, and then apply Power method again on the deflated system. λ_1 can be deflated to zero using the *Wielandt* deflation as follows:

$$\hat{\mathbf{A}}_1 = \hat{\mathbf{A}} - \lambda_1\mathbf{z}_1\mathbf{y}^T, \qquad \mathbf{y}^T\mathbf{z}_1 = 1, \tag{4.15}$$

where $\mathbf{y}$ is an arbitrary vector. If we apply the Power method to $\hat{\mathbf{A}}_1^{-1}$, we get the eigenpair $(\lambda_2, \mathbf{z}_2)$. Deflating λ_1 to zero is not the only choice to enable computation of λ_2 from Power method. Instead, if the smallest eigenvalue is shifted to some value larger than λ_2, Power method can compute eigenpair $(\lambda_2, \mathbf{z}_2)$. We can *generalize* the Wielandt deflation process by

$$\hat{\mathbf{A}}_{1,\gamma 1} = \hat{\mathbf{A}} - \gamma_1\mathbf{z}_1\mathbf{y}^T, \qquad \mathbf{y}^T\mathbf{z}_1 = 1, \qquad \gamma_1 \in \mathbb{R}. \tag{4.16}$$

Theorem 4.3.1. *The spectrum of* $\hat{\mathbf{A}}_{1,\gamma 1}$ *as defined in* (4.16) *is given by*

$$\sigma(\hat{A}_{1,\gamma 1}) = \{\lambda_1 - \gamma_1, \lambda_2, ..., \lambda_n\}. \tag{4.17}$$

Proof. See [54] □

In generalized deflation, if we choose $\gamma_1 = \lambda_1$, then from theorem 4.3.1 we get the spectrum of Wielandt deflation in (4.15) as $\sigma(\hat{A}_1) = \{0, \lambda_2, ..., \lambda_n\}$. Similarly, for the choice $\gamma_1 = \lambda_1 - \lambda_n$, we get the spectrum as $\sigma(\hat{A}_{1,\lambda_n}) = \{\lambda_n, \lambda_2, ..., \lambda_n\}$.

Deflation can be applied with several vectors to deflate many eigenvalues simultaneously. Suppose m smallest eigenvalues with $\mathbf{Z} = [\mathbf{z}_1 ... \mathbf{z}_m]$ the corresponding eigenvectors are previously computed. Next,to compute the $(m+1)th$ eigenpair using Power method, the first m eigenvalues can be deflated simultaneously by applying the following deflation

$$\hat{\mathbf{A}}_m = \hat{\mathbf{A}} - \mathbf{Z}\Gamma_m\mathbf{Y}^T \qquad \text{with } \mathbf{Y}^T\mathbf{Z} = 1, \tag{4.18}$$

where $\Gamma_m = diag(\gamma_1, ..., \gamma_m)$ and $Y = [\mathbf{y}_1, ..., \mathbf{y}_m]$.

Theorem 4.3.2. *Let $\hat{\mathbf{A}}_m$ be defined as in* (4.18)*, then for $i = 1, ..., m$ we have:*

(i) If we set $\gamma_i = \lambda_i$, then $\sigma(\hat{\mathbf{A}}_m) = \{0, ..., 0, \lambda_{m+1}, ..., \lambda_n\}$

(ii) If we set $\gamma_i = \lambda_i - \lambda_n$, then $\sigma(\hat{\mathbf{A}}_m) = \{\lambda_n, ..., \lambda_n, \lambda_{m+1}, ..., \lambda_n\}$

Proof. See [54] □

So far in the above discussion, the matrix $\mathbf{Y}$ is chosen to be composed of arbitrary vectors. If we choose $\mathbf{Y} = [y_1...y_r]$ to be the left eigenvector matrix of $\mathbf{A}$, then the eigenvalue matrix $\hat{\mathbf{E}}$ satisfies the following relation

$$\hat{\mathbf{E}} = \mathbf{Y}^T \hat{\mathbf{A}} \mathbf{Z} = diag(\lambda_1, ..., \lambda_m).$$

Now for the first case in theorem 4.3.2 with $\gamma_i = \lambda_i$ and for $i = 1, ..., m$, we have $\Gamma_m = \hat{\mathbf{E}}$. Therefore, from (4.18) we have

$$\hat{\mathbf{A}}_m = \hat{\mathbf{A}} - \mathbf{Z}\hat{\mathbf{E}}\mathbf{Y}^T = \hat{\mathbf{A}} - \mathbf{Z}\hat{\mathbf{E}}\hat{\mathbf{E}}^{-1}\hat{\mathbf{E}}\mathbf{Y}^T = \hat{\mathbf{A}} - \hat{\mathbf{A}}\mathbf{Z}\hat{\mathbf{E}}^{-1}\mathbf{Y}^T\hat{\mathbf{A}}.$$

Above relation can be rewritten as

$$\hat{\mathbf{A}}_m = (\mathbf{I} - \hat{\mathbf{A}}\mathbf{Z}\hat{\mathbf{E}}^{-1}\mathbf{Y}^T)\hat{\mathbf{A}} =: \hat{\mathbf{P}}_D\hat{\mathbf{A}}, \tag{4.19}$$

or as

$$\hat{\mathbf{A}}_m = \hat{\mathbf{A}}(\mathbf{I} - \mathbf{Z}\hat{\mathbf{E}}^{-1}\mathbf{Y}^T\hat{\mathbf{A}}) =: \hat{\mathbf{A}}\hat{\mathbf{Q}}_D. \tag{4.20}$$

Here $\hat{\mathbf{P}}_D$ and $\hat{\mathbf{Q}}_D$ are the left and right deflation preconditioners similar to the one defined in (4.7). Now if deflation preconditioner is applied to (4.14), i.e.,

$$\hat{\mathbf{P}}_D\hat{\mathbf{A}}\hat{\mathbf{x}} = \hat{\mathbf{P}}_D\hat{\mathbf{b}} \qquad \text{or} \qquad \hat{\mathbf{A}}\hat{\mathbf{Q}}_D\mathbf{u} = \hat{\mathbf{b}} \qquad \text{with } \mathbf{u} = \hat{\mathbf{Q}}_D^{-1}\hat{\mathbf{x}}, \tag{4.21}$$

then from theorem 4.3.2, the spectrum of the deflated system is given by

$$\sigma(\hat{\mathbf{P}}_D\hat{\mathbf{A}}) = \sigma(\hat{\mathbf{A}}\hat{\mathbf{Q}}_D) = \{0, ..., 0, \lambda_{m+1}, ..., \lambda_n\}. \tag{4.22}$$

Since $\kappa_{eff}(\hat{\mathbf{P}}_D\hat{\mathbf{A}}) = \kappa_{eff}(\hat{\mathbf{A}}\hat{\mathbf{Q}}_D) = \lambda_n/\lambda_{m+1} \leq \lambda_n/\lambda_1 = \kappa(\hat{\mathbf{A}})$, a Krylov method (like GMRES) applied on the deflated system will converge faster as compared to when applied on $\hat{\mathbf{A}}$. It is evident from the relation for effective condition number that larger deflation subspace implies better convergence, however, inverting larger deflation subspace exactly is not feasible from solver performance viewpoint. Instead, if the iterative method is used to invert large $\hat{\mathbf{E}}$, the small eigenvalues are not shifted exactly to zero, but to some very small value $0 < \epsilon \ll \lambda_1$, which makes the convergence more worse. The following proposition explains this.

Proposition 4.3.3. *Let $\mathbf{Z} = [\mathbf{z}_1...\mathbf{z}_m]$ and $\mathbf{Y}^T = [\mathbf{y}_1...\mathbf{y}_m]^T$ be the right and left eigenvectors of $\hat{\mathbf{A}}$ respectively, and let $\tilde{\mathbf{P}}_D = \mathbf{I} - \hat{\mathbf{A}}\mathbf{Z}\tilde{\mathbf{E}}^{-1}\mathbf{Y}^T$, with*

$$\tilde{\mathbf{E}}^{-1} = diag(\frac{1-\epsilon}{\lambda_1}...\frac{1-\epsilon}{\lambda_m}),$$

where $|\epsilon| \ll 1$. Then the spectrum of $\tilde{\mathbf{P}}_D\hat{\mathbf{A}}$ is given by

$$\sigma(\tilde{\mathbf{P}}_D\hat{\mathbf{A}}) = \{\epsilon\lambda_1, ..., \epsilon\lambda_m, \lambda_{m+1}, ..., \lambda_n\}. \tag{4.23}$$

Proof. For $i = i, ..., m$, we have

$$\begin{aligned}\tilde{\mathbf{P}}_D\hat{\mathbf{A}}\mathbf{Z} &= (\mathbf{I} - \hat{\mathbf{A}}\mathbf{Z}\tilde{\mathbf{E}}^{-1}\mathbf{Y}^T)\hat{\mathbf{A}}\mathbf{Z}\\ &= \hat{\mathbf{A}}\mathbf{Z} - \hat{\mathbf{A}}\mathbf{Z}\ diag(\frac{1-\epsilon}{\lambda_1}, ..., \frac{1-\epsilon}{\lambda_m})\mathbf{Y}^T\hat{\mathbf{A}}\mathbf{Z}\\ &= \hat{\mathbf{A}}\mathbf{Z} - \hat{\mathbf{A}}\mathbf{Z}\ diag(\frac{1-\epsilon}{\lambda_1}, ..., \frac{1-\epsilon}{\lambda_m})\ diag(\lambda_1, ..., \lambda_m)\\ &= \mathbf{Z}\ diag(\epsilon\lambda_1, ..., \epsilon\lambda_m).\end{aligned}$$

For $i = m+1, ..., n$,

$$\tilde{\mathbf{P}}_D\hat{\mathbf{A}}\mathbf{z}_i = \hat{\mathbf{A}}\mathbf{z}_i - \hat{\mathbf{A}}\mathbf{Z}\tilde{\mathbf{E}}^{-1}\mathbf{Y}^T\hat{\mathbf{A}}\mathbf{z}_i = \lambda_i\mathbf{z}_i - \hat{\mathbf{A}}\mathbf{Z}\tilde{\mathbf{E}}^{-1}\lambda_i\underbrace{\mathbf{Y}^T\mathbf{z}_i}_{=0} = \lambda_i\mathbf{z}_i.$$

□

From (4.23), for $0 \sim \epsilon \ll \lambda_1$, the condition number $\kappa(\tilde{\mathbf{P}}_D\hat{\mathbf{A}}) = \lambda_n/\epsilon\lambda_1$, will be very large, and Krylov solver will have serious convergence problems.

4.3.2 Stable shifting preconditioner

Since shifting the small eigenvalues to zero restricts the use of large deflation subspace, it seems a potential idea to shift the problematic eigenvalues to some constant other than zero, which may enable the use of large deflation subspaces. Based on this idea, Erlangga and Naben have proposed a projection-type preconditioner which is stable with respect to the inexact solve of the Galerkin system. For the construction of this stable preconditioner, we consider the second case of theorem 4.3.2, where $\gamma_i = \lambda_i - \lambda_n$, for $i = 1, ..., m$. For this choice of γ_i, we have $\Gamma_m = \hat{\mathbf{E}} - \lambda_n\mathbf{I}_m$, where $\mathbf{I}_m$ is an $(m \times m)$ identity matrix. Again by taking $\mathbf{Y}^T$ to be the left eigenvectors of $\hat{\mathbf{A}}$, we get

$$\begin{aligned}\hat{\mathbf{A}}_{m,\gamma} &= \hat{\mathbf{A}} - \mathbf{Z}\Gamma_m\mathbf{Y}^T = \hat{\mathbf{A}} - \mathbf{Z}(\hat{\mathbf{E}} - \lambda_n\mathbf{I}_m)\mathbf{Y}^T = \hat{\mathbf{A}} - \mathbf{Z}\hat{\mathbf{E}}\mathbf{Y}^T + \lambda_n\mathbf{Z}\mathbf{Y}^T\\ &= \hat{\mathbf{A}} - \mathbf{Z}\hat{\mathbf{E}}\hat{\mathbf{E}}^{-1}\hat{\mathbf{E}}\mathbf{Y}^T + \lambda_n\mathbf{Z}\hat{\mathbf{E}}^{-1}\hat{\mathbf{E}}\mathbf{Y}^T = \hat{\mathbf{A}} - \hat{\mathbf{A}}\mathbf{Z}\hat{\mathbf{E}}^{-1}\mathbf{Y}^T\hat{\mathbf{A}} + \lambda_n\mathbf{Z}\hat{\mathbf{E}}^{-1}\mathbf{Y}^T\hat{\mathbf{A}}.\end{aligned}$$

Therefore,

$$\hat{\mathbf{A}}_{m,\gamma} = (\mathbf{I} - \hat{\mathbf{A}}\mathbf{Z}\hat{\mathbf{E}}^{-1}\mathbf{Y}^T + \lambda_n\mathbf{Z}\hat{\mathbf{E}}^{-1}\mathbf{Y}^T)\hat{\mathbf{A}} = \hat{\mathbf{P}}_N\hat{\mathbf{A}}, \tag{4.24}$$

with

$$\hat{\mathbf{P}}_N = \mathbf{I} - \hat{\mathbf{A}}\mathbf{Z}\hat{\mathbf{E}}^{-1}\mathbf{Y}^T + \lambda_n\mathbf{Z}\hat{\mathbf{E}}^{-1}\mathbf{Y}^T = \hat{\mathbf{P}}_D + \lambda_n\mathbf{Z}\hat{\mathbf{E}}^{-1}\mathbf{Y}^T, \tag{4.25}$$

is the *left stable shifting* preconditioner. Notice that $\hat{\mathbf{P}}_N^2 \neq \hat{\mathbf{P}}_N$, therefore, it is not a projection, but we call it *projection-like* preconditioner since it projects the small eigenvalues to λ_n. The spectrum of $\hat{\mathbf{P}}_N\hat{\mathbf{A}}$ is similar to that of $\hat{\mathbf{P}}_D\hat{\mathbf{A}}$, with same effective condition number, therefore, a Krylov method applied to any of them will have similar convergence behavior. Note that if we take $\lambda_n = 0$ in equation (4.25), we get the deflation preconditioner. Similarly, a right version of shifting preconditioner can also be obtained as follows:

$$\begin{aligned}\hat{\mathbf{A}}_{m,\gamma} &= \hat{\mathbf{A}} - \mathbf{Z}\hat{\mathbf{E}}\mathbf{Y}^T + \lambda_n\mathbf{Z}\mathbf{Y}^T = \hat{\mathbf{A}} - \mathbf{Z}\hat{\mathbf{E}}\hat{\mathbf{E}}^{-1}\hat{\mathbf{E}}\mathbf{Y}^T + \lambda_n\mathbf{Z}\hat{\mathbf{E}}\hat{\mathbf{E}}^{-1}\mathbf{Y}^T\\ &= \hat{\mathbf{A}} - \hat{\mathbf{A}}\mathbf{Z}\hat{\mathbf{E}}^{-1}\mathbf{Y}^T\hat{\mathbf{A}} + \lambda_n\hat{\mathbf{A}}\mathbf{Z}\hat{\mathbf{E}}^{-1}\mathbf{Y}^T.\end{aligned}$$

Thus,

$$\hat{\mathbf{A}}_{m,\gamma} = \hat{\mathbf{A}}(\mathbf{I} - \mathbf{Z}\hat{\mathbf{E}}^{-1}\mathbf{Y}^T\hat{\mathbf{A}} + \lambda_n\mathbf{Z}\hat{\mathbf{E}}^{-1}\mathbf{Y}^T) = \hat{\mathbf{A}}\hat{\mathbf{Q}}_N, \tag{4.26}$$

with

$$\hat{\mathbf{Q}}_N = \mathbf{I} - \mathbf{Z}\hat{\mathbf{E}}^{-1}\mathbf{Y}^T\hat{\mathbf{A}} + \lambda_n\mathbf{Z}\hat{\mathbf{E}}^{-1}\mathbf{Y}^T = \hat{\mathbf{Q}}_D + \lambda_n\mathbf{Z}\hat{\mathbf{E}}^{-1}\mathbf{Y}^T. \tag{4.27}$$

The following proposition shows that $\hat{\mathbf{P}}_N$ is stable with respect to the inexact solve of Galerkin matrix $\hat{\mathbf{E}}$.

Proposition 4.3.4. *Let* $\mathbf{Z} = [\mathbf{z}_1 ... \mathbf{z}_m]$ *and* $\mathbf{Y}^T = [\mathbf{y}_1 ... \mathbf{y}_m]^T$ *be the right and left eigenvectors of* $\hat{\mathbf{A}}$ *respectively, and let* $\tilde{\mathbf{P}}_N = \mathbf{I} - \hat{\mathbf{A}}\mathbf{Z}\tilde{\mathbf{E}}^{-1}\mathbf{Y}^T + \lambda_n\mathbf{Z}\tilde{\mathbf{E}}^{-1}\mathbf{Y}^T$, *with*

$$\tilde{\mathbf{E}}^{-1} = diag(\frac{1-\epsilon}{\lambda_1} ... \frac{1-\epsilon}{\lambda_m}),$$

where $|\epsilon| \ll 1$. *Then the spectrum of* $\tilde{\mathbf{P}}_N\hat{\mathbf{A}}$ *is given by*

$$\sigma(\tilde{\mathbf{P}}_N\hat{\mathbf{A}}) = \{(1-\epsilon)\lambda_n + \epsilon\lambda_1, ..., (1-\epsilon)\lambda_n + \epsilon\lambda_m, \lambda_{m+1}, ..., \lambda_n\}. \tag{4.28}$$

Proof. For $i = 1, ..., m$, we have

$$\begin{aligned}
\tilde{\mathbf{P}}_N\hat{\mathbf{A}}\mathbf{Z} &= \hat{\mathbf{A}}\mathbf{Z} - \hat{\mathbf{A}}\mathbf{Z}\tilde{\mathbf{E}}^{-1}\mathbf{Y}^T\hat{\mathbf{A}}\mathbf{Z} + \lambda_n\mathbf{Z}\tilde{\mathbf{E}}^{-1}\mathbf{Y}^T\hat{\mathbf{A}}\mathbf{Z} \\
&= \hat{\mathbf{A}}\mathbf{Z} - \hat{\mathbf{A}}\mathbf{Z}\ diag(\frac{1-\epsilon}{\lambda_1}, ..., \frac{1-\epsilon}{\lambda_m})\ diag(\lambda_1, ..., \lambda_m) \\
&\quad + \lambda_n\mathbf{Z}\ diag(\frac{1-\epsilon}{\lambda_1}, ..., \frac{1-\epsilon}{\lambda_m})\ diag(\lambda_1, ..., \lambda_m) \\
&= \mathbf{Z}\ diag(\lambda_1\epsilon, ..., \lambda_m\epsilon) + \mathbf{Z}\ diag(\lambda_n(1-\epsilon), ..., \lambda_n(1-\epsilon)) \\
&= \mathbf{Z}\ diag(\lambda_n(1-\epsilon) + \lambda_1\epsilon, ..., \lambda_n(1-\epsilon) + \lambda_m\epsilon).
\end{aligned}$$

For $i = m+1, ..., n$,

$$\begin{aligned}
\tilde{\mathbf{P}}_N\hat{\mathbf{A}}\mathbf{z}_i &= \hat{\mathbf{A}}\mathbf{z}_i - \hat{\mathbf{A}}\mathbf{Z}\tilde{\mathbf{E}}^{-1}\mathbf{Y}^T\hat{\mathbf{A}}\mathbf{z}_i + \lambda_n\mathbf{Z}\tilde{\mathbf{E}}^{-1}\mathbf{Y}^T\hat{\mathbf{A}}\mathbf{z}_i \\
&= \lambda_i\mathbf{z}_i - \hat{\mathbf{A}}\mathbf{Z}\tilde{\mathbf{E}}^{-1}\lambda_i\underbrace{\mathbf{Y}^T\mathbf{z}_i}_{=0} + \lambda_n\mathbf{Z}\tilde{\mathbf{E}}^{-1}\lambda_i\underbrace{\mathbf{Y}^T\mathbf{z}_i}_{=0} = \lambda_i\mathbf{z}_i.
\end{aligned}$$

□

If $\hat{\mathbf{E}}$ is inverted without sufficient accuracy, and if $0 \sim \epsilon \ll 1$ and $\epsilon < \lambda_1 < \lambda_{m+1}$, we have

$$\kappa(\tilde{\mathbf{P}}_N\hat{\mathbf{A}}) = \frac{(1-\epsilon)\lambda_n + \epsilon\lambda_m}{\lambda_{m+1}} \approxeq \frac{\lambda_n}{\lambda_{m+1}} = \kappa(\hat{\mathbf{P}}_N\hat{\mathbf{A}}).$$

In $\tilde{\mathbf{P}}_N\hat{\mathbf{A}}$, λ_{m+1} still remains the smallest eigenvalue. Therefore, for a system preconditioned with $\hat{\mathbf{P}}_N$, the convergence rate of Krylov method will not be much affected, if Galerkin system $\hat{\mathbf{E}}$ is solved approximately with some iterative method. This gives the liberty to use larger deflation subspace to cluster many small eigenvalues around λ_n, and Krylov methods have better convergence rates for such clustered spectra.

Deflation process with general vectors

In the previous discussion, the deflation subspaces $\mathbf{Z}$ and $\mathbf{Y}$ are taken as invariant subspaces corresponding to the smallest eigenvalues of $\mathbf{A}$ and $\mathbf{A}^T$ respectively. However, these invariant subspaces are generally not available at hand, and computing them is something not suggestive, because of the substantial computational demands. If we choose $\mathbf{Z} = \mathbf{Y} \in \mathbb{R}^{n\times m}$ to be full rank and consist of arbitrary vectors, then $\hat{\mathbf{P}}_D$ still deflates the first m eigenvalues to zero (since $\hat{\mathbf{P}}_D\hat{\mathbf{A}}\mathbf{Z} = 0$). However, the remaining eigenvalues will not remain untouched, but instead will be changed. Similarly, for $\hat{\mathbf{P}}_N$, the first m eigenvalues are still shifted to λ_n (because $\hat{\mathbf{P}}_N\hat{\mathbf{A}}\mathbf{Z} = \lambda_n\mathbf{Z}$), and the rest of the eigenvalues change in a similar fashion to that for $\hat{\mathbf{P}}_D$. Above discussion can be stated in the form of the theorem as

Theorem 4.3.5. *Let $\hat{\mathbf{A}}$ be a nonsingular matrix, and $\mathbf{Z}$, $\mathbf{Y} \in \mathbb{R}^{n\times m}$ be any full rank matrices. If the spectrum of $\hat{\mathbf{P}}_D\hat{\mathbf{A}}$ is given by*

$$\sigma(\hat{\mathbf{P}}_D\hat{\mathbf{A}}) = \{0, ..., 0, \mu_{m+1}, ..., \mu_n\}, \tag{4.29}$$

then the spectrum of $\hat{\mathbf{P}}_N\hat{\mathbf{A}}$ is

$$\sigma(\hat{\mathbf{P}}_N\hat{\mathbf{A}}) = \{\lambda_n, ..., \lambda_n, \mu_{m+1}, ..., \mu_n\}. \tag{4.30}$$

Proof. See [54]. □

If λ_i and μ_i are the eigenvalues of $\hat{\mathbf{A}}$ and $\hat{\mathbf{P}}_D\hat{\mathbf{A}}$ respectively, then for nonsymmetric matrix $\hat{\mathbf{A}}$, it is not possible to make any relation between λ_i and μ_i. However, if $\mathbf{A}$ is SPD, then $\hat{\mathbf{P}}_D\hat{\mathbf{A}}$ is also symmetric, and in this case, it can be shown that $\mu_n \leq \lambda_n$ [74]. Thus deflation preconditioner will result in a better-clustered spectrum than shifting preconditioner

$$\kappa(\hat{\mathbf{P}}_D\hat{\mathbf{A}}) := \frac{\mu_n}{\mu_{m+1}} \leq \frac{\lambda_n}{\mu_{m+1}} =: \kappa(\hat{\mathbf{P}}_N\hat{\mathbf{A}}).$$

However, there always exist $\omega_1 \in \mathbb{R}$ for which $\omega_1\lambda_n = \mu_n$. If in the shifting preconditioner, the small eigenvalues are shifted to new constant $\omega_1\lambda_n$, instead of λ_n, then we have

$$\hat{\mathbf{P}}_N = \mathbf{I} - \hat{\mathbf{A}}\mathbf{Z}\hat{\mathbf{E}}^{-1}\mathbf{Y}^T + \omega_1\lambda_n\mathbf{Z}\hat{\mathbf{E}}^{-1}\mathbf{Y}^T, \tag{4.31}$$

and

$$\sigma(\hat{\mathbf{P}}_N\hat{\mathbf{A}}) = \{\mu_n, ..., \mu_n, \mu_{m+1}, ..., \mu_n\}. \tag{4.32}$$

Now the quality of clustering for deflation and shifting preconditioners is same, since $\kappa(\hat{\mathbf{P}}_D\hat{\mathbf{A}}) = \kappa(\hat{\mathbf{P}}_N\hat{\mathbf{A}})$, and similar convergence behaviors can be expected for $\hat{\mathbf{P}}_D\hat{\mathbf{A}}$ and $\hat{\mathbf{P}}_N\hat{\mathbf{A}}$.

Maximum eigenvalue approximation

Another issue related to the implementation of $\hat{\mathbf{P}}_N$ is that it involves maximum eigenvalue λ_n of $\hat{\mathbf{A}}$, which is not known. Exact computation of λ_n is expensive, Erlangga and Nabben have shown it numerically that an approximation to the maximum eigenvalue is sufficient. If the estimated eigenvalue $\lambda_{n,est}$ is not very far from the exact λ_n (i.e., the error is not of the order of λ_n), the convergence rate of Krylov subspace solver is unaltered. Moreover, there always exists some $\omega_2 \in \mathbb{R}$ such that $\lambda_n = \omega_2 \lambda_{n,est}$. Maximum eigenvalue can be estimated either using Gerschgorin's method or by some other means (for example by simple guess). Erlangga and Nabben have used the following consequence of Gerschgorin's theorem [74, page 24], to estimate the maximum eigenvalue for the Poisson and convection-diffusion problems.

$$\lambda_n \leq \max_{i \in \mathbb{N}} \sum_{j \in \mathbb{N}} |a_{i,j}| = \lambda_{n,est}. \tag{4.33}$$

The scaling factors ω_1 and ω_2 can be combined and replaced with single scaling factor as follows

$$\mu_n = \omega_1 \lambda_n = \omega_1 \omega_2 \lambda_{n,est} = \omega \lambda_{n,est},$$

where $\omega = \omega_1 \omega_2$ is called the shift scaling factor.

Remarks

We list some important remarks related to the stable shift preconditioners $\hat{\mathbf{P}}_N$ and $\hat{\mathbf{Q}}_N$.

- $\hat{\mathbf{P}}_N$ and $\hat{\mathbf{Q}}_N$ are not projection operators, as $\hat{\mathbf{P}}_N^2 \neq \hat{\mathbf{P}}_N$ and $\hat{\mathbf{Q}}_N^2 \neq \hat{\mathbf{Q}}_N$.
- $\hat{\mathbf{P}}_N$ and $\hat{\mathbf{Q}}_N$ are nonsingular.
- $\hat{\mathbf{P}}_N \hat{\mathbf{A}} \neq \hat{\mathbf{A}} \hat{\mathbf{Q}}_N$, but $\sigma(\hat{\mathbf{P}}_N \hat{\mathbf{A}}) = \sigma(\hat{\mathbf{A}} \hat{\mathbf{Q}}_N)$
- $\hat{\mathbf{P}}_N \hat{\mathbf{A}}$ and $\hat{\mathbf{A}} \hat{\mathbf{Q}}_N$ are not symmetric, even if $\hat{\mathbf{A}}$ is symmetric. Therefore, a Krylov method for nonsymmetric matrices (GMRES, BiCGSTAB, etc.) has to be used.

4.3.3 Two level implementation

Stable left preconditioning when applied to equation (4.14), results in the following left preconditioned system

$$\hat{\mathbf{P}}_N \hat{\mathbf{A}} \hat{\mathbf{x}} = \hat{\mathbf{b}}, \tag{4.34}$$

with

$$\hat{\mathbf{A}} = \mathbf{M}^{-1}\mathbf{A}, \qquad \hat{\mathbf{x}} = \mathbf{x}, \qquad \hat{\mathbf{b}} = \hat{\mathbf{P}}_N \mathbf{M}^{-1}\mathbf{b},$$

and

$$\hat{\mathbf{P}}_N = \mathbf{I} - \mathbf{M}^{-1}\mathbf{A}\mathbf{Z}\hat{\mathbf{E}}^{-1}\mathbf{Y}^T + \omega\lambda_{n,est}\mathbf{Z}\hat{\mathbf{E}}^{-1}\mathbf{Y}^T, \qquad \hat{\mathbf{E}} = \mathbf{Y}^T(\mathbf{M}^{-1}\mathbf{A})\mathbf{Z}. \tag{4.35}$$

Similarly, stable right preconditioning results in the following right preconditioned system

$$\hat{\mathbf{A}} \hat{\mathbf{Q}}_N \hat{\mathbf{x}} = \hat{\mathbf{b}}, \tag{4.36}$$

with

$$\hat{\mathbf{A}} = \mathbf{A}\mathbf{M}^{-1}, \qquad \mathbf{x} = \mathbf{M}^{-1}\mathbf{Q}_N\hat{\mathbf{x}}, \qquad \hat{\mathbf{b}} = \mathbf{b},$$

and

$$\hat{\mathbf{Q}}_N = \mathbf{I} - \mathbf{Z}\hat{\mathbf{E}}^{-1}\mathbf{Y}^T\mathbf{A}\mathbf{M}^{-1} + \omega\lambda_{n,est}\mathbf{Z}\hat{\mathbf{E}}^{-1}\mathbf{Y}^T, \qquad \hat{\mathbf{E}} = \mathbf{Y}^T(\mathbf{A}\mathbf{M}^{-1})\mathbf{Z}. \tag{4.37}$$

Left preconditioned GMRES can be used to solve (4.34), while right preconditioned GMRES can be employed to solve (4.36). Since $\hat{\mathbf{A}}\hat{\mathbf{Q}}_N$ and $\hat{\mathbf{P}}_N\hat{\mathbf{A}}$ have same spectra, the GMRES is expected to have similar convergence behavior for both left and right preconditioned systems. In algorithm 4.2, the right preconditioned GMRES is shown for solving equation (4.34). At k^{th} iteration, GMRES builds the solution from the Krylov subspace

$$\mathcal{K}^k\{\mathbf{r}_o, \mathbf{A}\mathbf{M}^{-1}\hat{\mathbf{Q}}_N\mathbf{r}_0, ..., (\mathbf{A}\mathbf{M}^{-1}\hat{\mathbf{Q}}_N)^{k-1}\mathbf{r}_0\},$$

with $\mathbf{r}_0 = \mathbf{b} - \mathbf{A}\mathbf{x}_0$. To calculate the Arnoldi vectors, GMRES requires the matrix $\hat{\mathbf{Q}}_N$. However, $\hat{\mathbf{Q}}_N$ is not a sparse matrix, and calculating it explicitly is not advisable particularly for large $\mathbf{A}$. Therefore, in MLKM implementation, the direct calculation of $\hat{\mathbf{Q}}_N$ is avoided and its application on some vector $\mathbf{v}_j$ is carried out as shown in lines 4-8 of algorithm 4.2. When inner iterations

Algorithm 4.2 Two level Flexible GMRES right preconditioned with $\hat{\mathbf{Q}}_N$ and $\mathbf{M}$

```
1:  Choose $\mathbf{x}_0$ ($\mathbf{x}_0 = 0$) $\omega$ and $\lambda_{n,est}$
2:  Compute $\mathbf{r}_0 = \mathbf{b} - \mathbf{A}\mathbf{x}_0$, $\beta = \|\mathbf{r}_0\|_2$, and $\mathbf{v}_1 = \mathbf{r}_0/\beta$
3:  for $j = 1, 2, , ..., k$ do
4:      $s := \mathbf{A}\mathbf{M}^{-1}\mathbf{v}_j$
5:      Restriction: $\mathbf{v}_R := \mathbf{Y}^T(s - \omega\lambda_{n,est}\mathbf{v}_j)$.
6:      Coarse-grid solve: $\tilde{\mathbf{v}} = \hat{\mathbf{E}}^{-1}\mathbf{v}_R$.
7:      Prolongation: $\mathbf{v}_P := \mathbf{Z}\tilde{\mathbf{v}}$.
8:      Correction: $\mathbf{x}_j := \mathbf{v}_j - \mathbf{v}_P$
9:      $\mathbf{w} := \mathbf{A}\mathbf{M}^{-1}\mathbf{x}_j$
10:     for $i = 1, 2, ..., j$ do
11:         $h_{i,j} = (\mathbf{w}, \mathbf{v}_j)$.
12:         $\mathbf{w} = \mathbf{w} - h_{i,j}\mathbf{v}_i$.
13:     end for
14:     Compute $h_{j+1,j} := \|\mathbf{w}\|_2$ and $\mathbf{v}_{j+1} = \mathbf{w}/h_{j+1,j}$.
15: end for
16: Define $\mathbf{X}_k := [\mathbf{x}_1, ..., \mathbf{x}_k]$ and $\hat{\mathbf{H}}_k = \{h_{i,j}\}_{1\leq i\leq j+1; 1\leq j\leq k}$.
17: Compute $\mathbf{y}_k$ the minimizer of $\|\beta\mathbf{e}_1 - \hat{\mathbf{H}}_k\mathbf{y}\|_2$ and $\mathbf{x}_k = \mathbf{x}_0 + \mathbf{M}^{-1}\mathbf{X}_k\mathbf{y}_k$.
```

are used, $\hat{\mathbf{Q}}_N$ is in general not constant, therefore, flexible version of GMRES is used to accommodate the variable preconditioner. This requires storage of $\mathbf{X}_k = [\mathbf{x}_1, ..., \mathbf{x}_k]$, in addition to $\mathbf{V}_k = [\mathbf{v}_1, ..., \mathbf{v}_k]$ vectors, and the solution is built using $\mathbf{X}_k$. One iteration of algorithm 4.2 requires two preconditioner solves, followed by two matrix-vector multiplications (line 4 and 9), and one solve involving smaller dimension matrix $\hat{\mathbf{E}}$. If we set $\mathbf{M} = \mathbf{I}$, this work is equal to the work with deflation preconditioner $\mathbf{P}_D$, and less than that with balancing preconditioner $\mathbf{P}_B$.

4.3.4 Multilevel implementation

If large deflation subspaces are used, then the Galerkin system solve in algorithm 4.2, given by

$$\hat{\mathbf{E}}\tilde{\mathbf{v}} \;=\; \mathbf{v}_R, \tag{4.38}$$

should be performed with some (inner) Krylov method using weak termination criterion, to ensure optimal overall computational complexity of the algorithm. If the condition number of Galerkin matrix $\hat{\mathbf{E}}$ is large, the Krylov method will require many iterations to converge. The convergence rate can be enhanced by again applying the shift preconditioner $\hat{\mathbf{Q}}_N$ to (4.38). Even if the condition number of Galerkin system is not large, the action of shift preconditioner can further enhance the convergence of Krylov iteration. Application of this shift operator further involves a coarse grid solve similar to (4.38) at the next coarser level, which can be performed by another inner Krylov iteration. Each inner iteration solves a much smaller Galerkin system than the previous inner solve. A recursive application of this process results in a multilevel Krylov method (MLKM).

4.4 MLKM in context of FEATFLOW

Erlangga and Nabben in [54], have used *Galerkin coarse grid projection* as a representation of the Galerkin system, i.e., $\hat{\mathbf{E}} = \mathbf{Y}^T(\mathbf{A}\mathbf{M}^{-1})\mathbf{Z}$. Their MLKM implementation requires the explicit calculation of Galerkin matrices at all coarse levels in the initialization phase of the algorithm. Notice that to calculate these matrices at each coarse level, one needs to have preconditioner matrix $\mathbf{M}^{-1}$ available at hand at these levels. However, the preconditioner matrix may not be explicitly available in many cases (for example how to get the $\mathbf{M}^{-1}$ matrix if a Krylov iteration is used as a preconditioner, or if one iteration of multigrid is used as a preconditioner). Even if $\mathbf{M}^{-1}$ is explicitly available, generally the $\mathbf{M}^{-1}$ matrix will be dense, and the resulting Galerkin matrix will also be dense. Computing and storing the dense $\mathbf{M}^{-1}$ and $\hat{\mathbf{E}}$ matrices will be too much time and memory consuming, something we do not want at all. The only possibility is to use Jacobi as a preconditioner, for which the $\mathbf{M}^{-1}$ and $\hat{\mathbf{E}}$ will not be dense, and this is the choice being used by authors in [54]. However, in general, the diagonal preconditioning does not achieve much in terms of convergence rate improvement.

In FEATFLOW, we do not explicitly calculate and store the $\mathbf{M}^{-1}$ matrix; instead, a preconditioner is applied to some vector to get the preconditioned vector. Hence, the $\mathbf{M}^{-1}$ matrix is not readily available to calculate the Galerkin matrix. In our implementation of MLKM algorithm, we use *discretization coarse grid approximation* to setup Galerkin system at the coarse levels. With this approach, the Galerkin system at the second level reads

$$\mathbf{A}^{(2h)}\mathbf{M}^{(2h)^{-1}}\tilde{\mathbf{v}} = \mathbf{v}_R, \tag{4.39}$$

where $\mathbf{A}^{(2h)}$ is a coarse grid matrix, which like fine grid matrix $\mathbf{A}^{(h)}$, is obtained from the discretization of the original equation at the second level, and $\mathbf{M}^{(2h)}$

is the preconditioner matrix based on $\mathbf{A}^{(2h)}$. Thus, at the second level MLKM solves the system

$$\mathbf{A}^{(2h)}\mathbf{M}^{(2h)^{-1}}\hat{\mathbf{Q}}_N\mathbf{v}' = \mathbf{v}_R, \qquad \tilde{\mathbf{v}} = \hat{\mathbf{Q}}_N\mathbf{v}', \tag{4.40}$$

using FGMRES iterations.

The advantage of using the discretized coarse grid matrix for Galerkin system is that we have the liberty of using any iterative method as a preconditioner $\mathbf{M}$, without having a fear of getting into the trouble of dense matrices. However, an apparent disadvantage is that at the lowest level we cannot exactly solve the associated Galerkin system of the form (4.39) since we do not have the matrix $\mathbf{M}$ available with us. Nevertheless, our numerical results have shown that at coarsest level whether we solve exactly or approximately, it does not significantly affect the convergence of the MLKM solver. Moreover, in most of the real-life applications, the geometries are large and complex, and the grids are highly irregular, thus making it impossible to coarsen the grids to the point that direct methods could be used to solve the coarse problems. In MLKM, the coarse grid systems at all levels are solved approximately using few iterations of GMRES.

To facilitate the explanation of our multilevel Krylov method implementation in more detail, we introduce new notations to cater for the level identification. Suppose that L levels are used, with $l = L$ to be the finest level on which we want to solve the problem, and $l = 1$ to be the coarsest level. Let $\mathbf{A}^{(l)}$ be the discretized system matrices at each level l and $\mathbf{M}^{(l)}$ be the corresponding preconditioner matrices. At the level L, MLKM solves the following system

$$\mathbf{A}^{(L)}\mathbf{M}^{(L)^{-1}}\mathbf{Q}_N^{(L)}\hat{\mathbf{x}}^{(L)} = \mathbf{b}^{(L)}, \qquad \mathbf{x}^{(L)} = \mathbf{M}^{(L)^{-1}}\mathbf{Q}_N^{(L)}\hat{\mathbf{x}}^{(L)}, \tag{4.41}$$

where the shift operator is given by

$$\begin{aligned} \mathbf{Q}_N^{(L)} &= \mathbf{I}^{(L)} - \mathbf{Z}^{(L,L-1)}\mathbf{E}^{(L-1)^{-1}}\mathbf{Y}^{(L,L-1)^T}\mathbf{A}^{(L)}\mathbf{M}^{(L)^{-1}} \\ &\quad + \omega^{(L)}\lambda_{n,est}^{(L)}\mathbf{Z}^{(L,L-1)}\mathbf{E}^{(L-1)^{-1}}\mathbf{Y}^{(L,L-1)^T}, \\ &\text{with} \\ \mathbf{E}^{(L-1)} &= \mathbf{A}^{(L-1)}\mathbf{M}^{(L-1)^{-1}}. \end{aligned} \tag{4.42}$$

The Galerkin system at level $l = (L-1)$ is given by $\mathbf{E}^{(L-1)}\tilde{\mathbf{v}}^{(L-1)} = \mathbf{v}_R^{(L-1)}$. Letting $\tilde{\mathbf{v}}^{(L-1)} = \mathbf{x}^{(L-1)}$ and $\mathbf{v}_R^{(L-1)} = \mathbf{b}^{(L-1)}$, at level $(L-1)$ MLKM solves the system of the form

$$\mathbf{A}^{(L-1)}\mathbf{M}^{(L-1)^{-1}}\mathbf{Q}_N^{(L-1)}\hat{\mathbf{x}}^{(L-1)} = \mathbf{b}^{(L-1)}, \qquad \mathbf{x}^{(L-1)} = \mathbf{Q}_N^{(L-1)}\hat{\mathbf{x}}^{(L-1)}, \tag{4.43}$$

with projection preconditioner

$$\begin{aligned} \mathbf{Q}_N^{((L-1)} &= \mathbf{I}^{((L-1)} - \mathbf{Z}^{((L-1,L-2)}\mathbf{E}^{(L-2)^{-1}}\mathbf{Y}^{((L-1,L-2)^T}\mathbf{A}^{((L-1)}\mathbf{M}^{((L-1)^{-1}} \\ &\quad + \omega^{((L-1)}\lambda_{n,est}^{((L-1)}\mathbf{Z}^{((L-1,L-2)}\mathbf{E}^{(L-2)^{-1}}\mathbf{Y}^{((L-1,L-2)^T} \\ &\text{and} \\ \mathbf{E}^{(L-2)} &= \mathbf{A}^{(L-2)}\mathbf{M}^{(L-2)^{-1}}. \end{aligned} \tag{4.44}$$

Likewise, at each subsequent level till $l = 2$, the Galerkin system similar to (4.43) is solved. At the coarsest level $l = 1$, the following Galerkin system is solved approximately using few iterations of FGMRES

$$\mathbf{A}^{(1)}\mathbf{M}^{(1)^{-1}}\hat{\mathbf{x}}^{(1)} = \mathbf{b}^{(1)}, \qquad \mathbf{x}^{(1)} = \hat{\mathbf{x}}^{(1)}. \tag{4.45}$$

If we look at the relation between $\hat{\mathbf{x}}$ and $\mathbf{x}$ in equations (4.41),(4.43), and (4.45), it is clear that at finest level $l = L$, MLKM extracts solution from $\mathbf{x}_k = \mathbf{x}_0 + \mathbf{M}^{-1}\mathbf{X}_k\mathbf{y}_k$, at levels $1 < l < L$ from $\mathbf{x}_k = \mathbf{x}_0 + \mathbf{X}_k\mathbf{y}_k$, and at the coarsest level $l = 1$ from $\mathbf{x}_k = \mathbf{x}_0 + \mathbf{V}_k\mathbf{y}_k$ (see line 17 of algorithm 4.2). Algorithm 4.3 represents our implementation of MLKM in the FEATFLOW software.

Erlangga and Nabben in [54] have used piece-wise constant interpolation matrix as $\mathbf{Z}$ and taken $\mathbf{Y} = \mathbf{Z}$. With this choice of deflation subspaces, they have used (4.33) for approximating the maximum eigenvalue $\lambda_{n,est}$. We have used standard finite element based inter-grid transfer operators of multigrid as deflation subspaces $\mathbf{Z}$ and $\mathbf{Y}^T$. The first reason for using these grid transfer operators is that they are already implemented in FEATFLOW. Secondly, we have observed that with these standard MG grid transfer operators, setting maximum eigenvalue equal to one for all levels and choosing an appropriate value of ω, in most of the cases resulted in best convergence rates for the MLKM solver. Hence we can avoid the calculation of approximate eigenvalue of the preconditioned system matrix, thus reducing the computational head of the solver.

Algorithm 4.3 stops if either the convergence criterion set at the finest level is met or if the maximum number of iterations is reached. In MLKM algorithm, GMRES is not restarted at the coarse levels (for $l < L$) and maximum number of iterations at these levels is set equal to the Krylov subspace dimension at these levels. The MLKM iteration is represented with the Krylov dimensions set at these levels. For example, MLKM(4,2,1) means one iteration of MLKM at the finest level involves 4 FGMRES iterations at level $l = L - 1$, 2 FGMRES iterations at levels $1 < l < L - 1$, and 1 iteration at the coarsest level ($l = 1$). Figure 4.1 shows one iteration of MLKM(4,2,2) configuration.

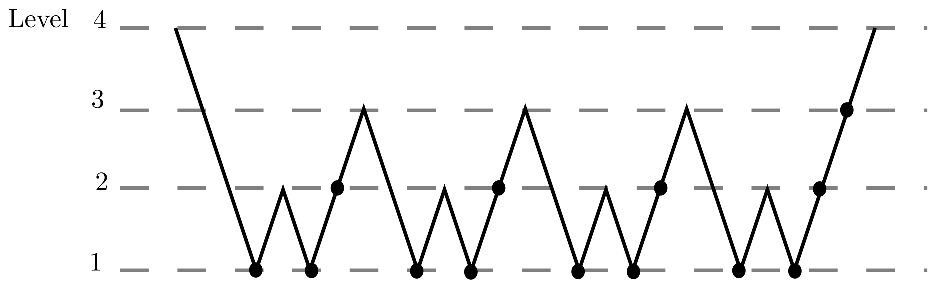


Figure 4.1: MLKM(4,2,2) iteration. ●– GMRES solve after completion of the number of GMRES iterations set at that level.

Algorithm 4.3 Multilevel Krylov Method

```
1: MLKM ($\mathbf{A}^{(l)}$, $\mathbf{M}^{(l)}$, $\mathbf{Z}^{(l,l-1)}$, $\mathbf{Y}^{(l,l-1)}$, $\mathbf{b}^{(l)}$, $\lambda_{n,est}^{(l)}$, $\omega^{(l)}$, $l$) with all the
   arguments already chosen or determined.
2: Choose $\mathbf{x}_0^{(l)}$ and calculate: $\mathbf{r}_0^{(l)} = \mathbf{b}^{(l)} - \mathbf{A}^{(l)}\mathbf{x}_0^{(l)}$, $\beta^{(l)} = \|\mathbf{r}_0^{(l)}\|_2$, and $\mathbf{v}_1^{(l)} =$
   $\mathbf{r}_0^{(l)}/\beta^{(l)}$
3: $ite = 0$
4: for $ite < maximum\ iterations$ do
5:     for $j = 1, ..., k$ do                    ▷ $k$ = Krylov subspace dimension
6:         $ite = ite + 1$
7:         $\mathbf{x}_j^{(l)} = \mathbf{v}_j^{(l)}$
8:         if ($l > 1$) then
9:             $s := \mathbf{A}^{(l)}\mathbf{M}^{(l)^{-1}}\mathbf{v}_j^{(l)}$
10:            Restriction: $\mathbf{v}_R^{(l-1)} := \mathbf{Y}^{(l,l-1)^T}(s - \omega^{(l)}\lambda_{n,est}^{(l)}\mathbf{v}_j^{(l)})$
11:            Coarse-grid solve: Solve $\mathbf{A}^{(l-1)}\mathbf{M}^{(l-1)^{-1}}\tilde{\mathbf{v}}^{(l-1)} = \mathbf{v}_R^{(l-1)}$ for
   $\tilde{\mathbf{v}}^{(l-1)}$ by a recursive call to MLKM ($\mathbf{A}^{(l)}$, $\mathbf{M}^{(l)}$, $\mathbf{Z}^{(l,l-1)}$, $\mathbf{Y}^{(l,l-1)}$, $\mathbf{b}^{(l)} =$
   $\mathbf{v}_R^{(l)}$, $\lambda_{n,est}^{(l)}$, $l = l - 1$)
12:            Prolongation: $\mathbf{v}_P^{(l)} := \mathbf{Z}^{(l,l-1)}\tilde{\mathbf{v}}^{(l-1)}$.
13:            Correction: $\mathbf{x}_j^{(l)} := \mathbf{v}_j^{(l)} - \mathbf{v}_P^{(l)}$
14:        end if
15:        $\mathbf{w}^{(l)} := \mathbf{A}^{(l)}\mathbf{M}^{(l)^{-1}}\mathbf{x}_j^{(l)}$
16:        for $i = 1, 2, ..., j$ do
17:            $h_{i,j}^{(l)} = (\mathbf{w}^{(l)}, \mathbf{v}_j^{(l)})$.
18:            $\mathbf{w}^{(l)} = \mathbf{w}^{(l)} - h_{i,j}^{(l)}\mathbf{v}_i^{(l)}$.
19:        end for
20:        Compute $h_{j+1,j}^{(l)} := \|\mathbf{w}^{(l)}\|_2$ and $\mathbf{v}_{j+1}^{(l)} = \mathbf{w}^{(l)}/h_{j+1,j}^{(l)}$.
21:    end for
22:    Set $\mathbf{X}_k^{(l)} := [\mathbf{x}_1^{(l)}, ..., \mathbf{x}_k^{(l)}]$ and $\hat{\mathbf{H}}_k^{(l)} = \{h_{i,j}^{(l)}\}_{1\le i\le j+1; 1\le j\le k}$.
23:    Compute $\mathbf{y}_k^{(l)}$ the minimizer of $\|\beta^{(l)}\mathbf{e}_1 - \hat{\mathbf{H}}_k^{(l)}\mathbf{y}^{(l)}\|_2$.
24:    if ($l = 1$) then
25:        Compute $\mathbf{x}_k^{(l)} = \mathbf{x}_0^{(l)} + \mathbf{M}^{(l)^{-1}}\mathbf{X}_k^{(l)}\mathbf{y}_k^{(l)}$
26:    else
27:        Compute $\mathbf{x}_k^{(l)} = \mathbf{x}_0^{(l)} + \mathbf{X}_k^{(l)}\mathbf{y}_k^{(l)}$
28:    end if
29:    $\mathbf{v}_1^{(l)} = (\mathbf{b}^{(l)} - \mathbf{A}^{(l)}\mathbf{x}_k^{(l)})/\|\mathbf{b}^{(l)} - \mathbf{A}^{(l)}\mathbf{x}_k^{(l)}\|_2$
30: end for
```

4.5 Multilevel Krylov method vs. MultiGrid method

Although MLKM involves ingredients similar to that of multigrid, however, both solvers are different in many aspects. MLKM is different from multigrid and other multilevel solvers like domain decomposition methods, in a way that these multilevel solvers can be represented as a multistep fixed point iteration[54]. In multigrid, for example, the error at each iteration can be written in terms of the error at previous iteration by the relation:

$$\mathbf{e}^{(k+1)} = \mathbf{B}\ \mathbf{e}^{(k)}, \tag{4.46}$$

where $\mathbf{B} = \underbrace{(I - M_h^{-1} A_h)}_{\text{Post-smoothing}}\ \underbrace{(I - I_H^h A_H^{-1} I_h^H A_h)}_{Coarse-grid\ correction}\ \underbrace{(I - M_h^{-1} A_h)}_{\text{Pre-Smoothing}}.$

However, for multilevel Krylov method, no such fixed point relation can be established, and it can only be seen as a Krylov subspace iteration. Multigrid works on the solution vector $\mathbf{x}$ and the error vector $\mathbf{e}$, while MLKM works on the Arnoldi vectors which form the basis of Krylov subspace in which the approximate solution vector is contained. In MLKM, $\mathbf{M}^{-1}$ does not work as smoother but only as a preconditioner to improve the condition number of the matrix. If the matrix is not very ill conditioned, we can set $\mathbf{M} = \mathbf{I}$ and MLKM still works fine (see chapter 5, table 5.2). On the other hand, MG without smoother does not work.

Even though the coarse-grid correction of MLKM is similar to that of multigrid, the construction of inter-level transfer operators of MLKM does not require accurate interpolations; it only requires that the inter-grid transfer matrices are full rank. Erlangga and Nabben in [54], therefore, have used piece-wise constant interpolation matrix as $\mathbf{Z}$ and taken $\mathbf{Y} = \mathbf{Z}$. This choice of deflation subspaces leads to efficient MLKM solver, whereas, MG with prolongation and restriction based on piece-wise constant interpolation does not produce good results.

5 Convection-Diffusion Problem

In this chapter and the next chapter, we solve the scalar partial differential equations with MLKM solver and compare its performance with geometric multigrid, which is the standard multilevel solver technique available in FEATFLOW. We have chosen the convection-diffusion problem and anisotropic diffusion problem as our scalar model problems. These PDEs represent many important physical phenomena in nature. However, the main reason for choosing these partial differential equations as our model problems is that they exhibit numerical challenges for the solvers, that also appear in the more complex Navier-Stokes equations. Therefore, before embarking on the complicated systems of Navier-Stokes equations, we analyze the performance of MLKM solver on these relatively simple scalar problems.

The convection-diffusion problem is dealt with in this chapter, and the anisotropic diffusion problem is considered in the next chapter. We first describe the convection-diffusion problem and then give its weak formulation. We also briefly discuss the stabilization techniques used to stabilize the numerical results for highly convective flows. Numerical results begin with the code validation tests, where we ensure that our MLKM implementation is correct by solving the problem with a known exact solution and compare our numerical results with this exact solution. We calculate the solution error at different mesh refinement levels and observe the asymptotic error decay. Next, we present the numerical results for the solution of the convection-diffusion problem on various test cases and compare the performance of our multilevel Krylov method implementation in FEATFLOW with the multigrid solver. The solvers are tested for a wide range of problem parameters (Peclet number, mesh refinement) to check their robustness. We also analyze the performance of solvers for various preconditioners/smoothers.

5.1 Convection-diffusion problem

Convection and diffusion play a vital role in the transport of scalar quantities (e.g., temperature, density, concentration, and so forth) in various science and engineering applications. Convection is the transport phenomenon due to the bulk motion of the fluid, whereas diffusion attributes to the transport of quantities from high concentration areas to low concentration areas caused by the random molecular motion. In many situations both convection and diffusion occur side by side, for instance, the smoke coming out of the chimney spreads into the atmosphere due to the wind (convection) and due to concentration gradients (diffusion). The combined effect of both the processes can be studied by solving the convection-diffusion equation. If we take $\mathcal{L} = -d\Delta + \boldsymbol{c}.\nabla$ in equation (2.1), we get the steady state convection-diffusion boundary value problem as follows

$$\begin{aligned} -d\,\Delta u + \boldsymbol{c}.\nabla u \;&=\; f \quad \text{in} \quad \Omega, \\ u \;=\; g \quad \text{on} \quad \Gamma_D \quad &\text{and} \quad \nabla u.\mathbf{n} \;=\; \beta \quad \text{on} \quad \Gamma_N, \end{aligned} \tag{5.1}$$

where $\boldsymbol{c}$ is convective velocity and d is diffusive coefficient. The transport process may be quite different depending on the convective and diffusive transport rates. For example, on a windy day, the smoke coming out of the chimney moves downstream faster than it spreads out, whereas, on a calm day the smoke spreads out faster due to molecular diffusion than it moves downstream. The dimensionless number called *Peclet number* can quantify the relative strength of convection and diffusion

$$Pe = \frac{CL}{d}, \tag{5.2}$$

here C is the velocity magnitude, and L is the characteristic length in the problem. Equation (5.1) can be seen as a “linear scalar version” of Navier-Stokes equations, discussed in chapter 7, in which case the unknown $\mathbf{u}$ is the vector-valued velocity field, $\boldsymbol{c}$ is also the unknown $\mathbf{u}$ that makes the problem nonlinear, and d is the viscosity parameter. Likewise, the Peclet number for the linear equation (5.1) corresponds to the Reynolds number for the Navier-Stokes equations.
The weak form of the convection-diffusion equation (5.1) is given as follows: *Find $u \in \mathcal{H}^1_E$ such that*

$$a(v,u) = b(v) \qquad \forall v \in \mathcal{H}^1_0, \tag{5.3}$$

where $a(.,.) : \mathcal{H}^1(\Omega) \times \mathcal{H}^1(\Omega) \mapsto \mathbb{R}$ is the bilinear form given by

$$a(v,u) := d \int\limits_{\Omega} \nabla v.\nabla u + \int\limits_{\Omega} (\boldsymbol{c}.\nabla u)v. \tag{5.4}$$

As is obvious $a(v,u) \neq a(u,v)$, therefore, the bilinear form is nonsymmetric. The linear form $b : \mathcal{H}^1(\Omega) \mapsto \mathbb{R}$ on the right side of equation (5.3) has the form

$$b(v) := \int\limits_{\Omega} vf + \int\limits_{\Gamma_N} v\beta. \tag{5.5}$$

Following the standard Galerkin finite element process as described before in chapter 2, the discrete form of the equation (5.3) is as follows:

Find $u_h \in \mathcal{S}^h_E$ such that

$$a(v_h,u_h) = b(v_h) \qquad \forall v_h \in \mathcal{S}^h_0. \tag{5.6}$$

5.1.1 Stabilization of convective term

For a self adjoint (symmetric) operator $\mathcal{L}$, the standard Galerkin method produces *best approximations*, i.e., it minimizes the error in the energy norm. However, the nonsymmetric convective part in convection-diffusion equation deprives the Galerkin method from its best approximation property. In highly convective (large Peclet number) flows, the solution u may exhibit steep gradients, particularly near the boundaries to satisfy the boundary conditions. If these steep gradients are not accurately captured by the numerical scheme used to solve the discrete problem, spurious oscillations, also called wiggles, may occur at the boundary layer. These oscillations then propagate along the streamlines into the domain where the solution u is even smooth, thus spoiling the whole solution, and making the iterative solver unstable.

One approach to handle the problem is to use *locally* refined meshes in the areas where the solution is changing rapidly. In some cases, the areas where the steep gradients arise are apparent. However, in general, it is not always possible to identify all the areas where the solution is nonsmooth, especially when the streamlines are complicated. In such cases, *adaptive local mesh refinement* may be applied: a discrete solution and the corresponding a posteriori error is computed on the initial grid, then the mesh is locally refined in the areas where the error is significant. Nonetheless, this strategy can only be useful, if the errors do not propagate into the regions of the domain where the solution is well behaved.

Several methods have been proposed in the literature to stabilize the discrete solution of the convection-diffusion problem coming from standard Galerkin method. *Streamline diffusion method* (SD), introduced by Hughes and Brooks [76], adds some artificial diffusion along the streamlines that rectifies the oscillatory behavior of the discrete solution. The *Streamline-Upwind Petrov-Galerkin* (SUPG) method, also proposed by Hughes and Brooks [25], interpret the diffusion added in the streamline direction as a modification of the test space and uses this modified test function for all terms of the weak form. This method results in a consistent scheme, where the exact solution of the problem still satisfies the weak form resulting from the stabilization of SUPG method. For more details on SD and SUPG, see [52, 83, 61].

In this thesis, we use another promising stabilization technique called *edge-oriented jump (EOJ) stabilization*, originally proposed by Douglas and Dupont [45]. The working philosophy of the EOJ stabilization is different from SD and SUPG; instead of looking at local Peclet/Reynold numbers, EOJ checks the smoothness of the discrete solution to determine the amount of the stabilization required. The main idea of the technique involves introducing additional interior penalty terms into the weak formulation of the standard Galerkin discretization in a consistent manner. The penalty term controls the jump of the gradient of the discrete solution over the element boundaries. Different jump terms have been proposed in the literature; we use the unified edge-oriented jump term proposed by A. Ouazzi and S. Turek [134], which is of the form

$$\langle \mathbf{S}u_h, v_h \rangle = \sum_{edge\ E} \max(\gamma^* \nu h_E, \gamma h_E^2) \int_E [\nabla u_h][\nabla v_h] d\sigma, \tag{5.7}$$

where ν is the viscosity, h_E is the length of an element edge E, and $[\cdot]$ the jump of function over an edge E. The stabilization parameters γ and γ^* are free constants, and the results are in many cases insensitive to their choice. We have always used $\gamma^* = 0$ in all the numerical experiments. The negative implication of the EOJ stabilization is that the standard sparsity pattern of the Galerkin FEM discretization is destroyed. An extended matrix stencil is required to introduce the extra nonzero entries resulting from stabilization term, which leads to additional memory requirements, as well as an extra computational head for the linear solvers. For more details on EOJ stabilization, see [134].

5.2 Numerical results

We analyze the Multilevel Krylov method numerically for stationary convection-diffusion equation where the associated coefficient matrix is nonsymmetric. MLKM performance is compared with the multigrid method for the said problem on structured as well as unstructured meshes. As a test problem for structured meshes, the convection-diffusion problem is solved on a unit square domain. For performance analysis on the unstructured meshes, the problem is solved on the rectangular domain with a circular obstacle. However, before starting this solver performance comparison, we validate our multilevel Krylov solver implementation by solving the convection-diffusion equation on a unit square domain with some known exact solution.

5.2.1 Code validation

For validation purpose, we consider the following stationary convection-diffusion equation in two dimensions

$$-\frac{1}{\mathrm{Pe}}\Delta u + \begin{pmatrix} 0 \\ 1 \end{pmatrix} \nabla u = f \quad \text{in} \quad \Omega = (0,1)\times(0,1) \tag{5.8}$$

where Pe is the Peclet, and the convection is in vertical direction only. The boundary conditions of Dirichlet type read as follows

$$u(x,y) = \begin{cases} 0 & \text{if } x \ = \ 0 \text{ or } y \ = \ 0, \\ y^3 & \text{if } x \ = \ 1, \\ x^3 & \text{if } y \ = \ 1. \end{cases} \tag{5.9}$$

The right hand side f in equation (5.8) is set to $f = -3xy(-x^2y + 2x^2 + 2y^2)$ to produce the solution $u = x^3y^3$. MLKM (4,2,2) with Jacobi preconditioner is used to solve the above problem with $Pe = 1$. Maximum eigenvalue (λ_{max}) and scaling parameter (ω) in MLKM are set equal to one. The mesh at level 1 consists of one cell, and refined meshes are obtained by performing the uniform mesh refinements, whereby the midpoints of opposite edges of each coarse mesh cell are joined. In all the simulations, level 2 mesh with four quadrilaterals is used as a coarse mesh. Simulations are performed at various mesh refinement levels for both bilinear (Q_1) and biquadratic (Q_2) finite element discretizations, and the corresponding L_2 and H_1 norms of the errors are presented in table 5.1.

The results show that with the grid refinements, the solution error for Q_1 finite element discretization in L_2 and H_1 norms are reduced with factors of 4 and 2, respectively. Similarly, in the case of Q2 discretization, for the first few mesh refinement levels, the L_2 and H_1 errors are reduced by a factor of 8 and 4, respectively. As soon as the L_2 error is reduced to the same order as that of stopping criterion (10^{-06}), these factors are not obtained. If the stopping criterion is further reduced, then in the next levels also L_2 and H_1 errors reduce with the same factors of 8 and 4, respectively. The asymptotic error reduction behavior follows precisely what the theory suggests, thus confirming that our code is bug-free, and we may proceed to compare the performance of MLKM solver with geometric MG solver.

	Q_1		Q_2	
Level	L_2-Error	H_1-Error	L_2-Error	H_1-Error
4	3.89E-03	6.72E-02	3.61E-05	1.87E-03
5	9.76E-04	3.34E-02	4.51E-06	4.67E-04
6	2.44E-04	1.67E-02	5.64E-07	1.167E-04
7	6.10E-05	8.35E-03	8.55E-08	2.92E-05
8	1.52E-05	4.18E-03	3.74E-08	7.78E-06

Table 5.1: Error analysis of MLKM solver for Convection-Diffusion problem with $Pe = 1$

5.2.2 Test Problem 1 (Structured meshes)

We study the effect of convection on the performance of MLKM and compare it with the MG solvers. To this end, we solve the stationary convection-diffusion equation for increasing Peclet numbers and list the number of iterations taken by these solvers to reach the desired accuracy. We begin with solving the two dimensional stationary convection-diffusion problem on a unit square domain with the right-hand side f set equal to zero,

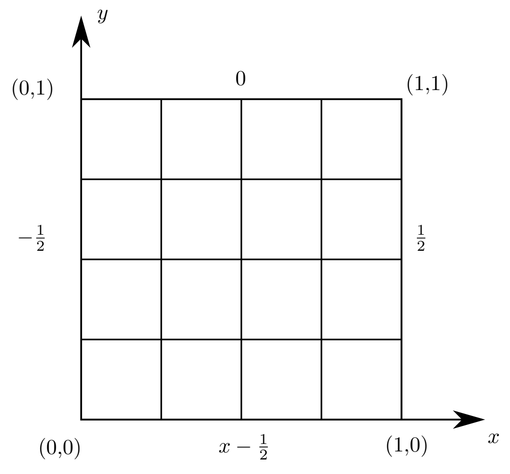
$$-\frac{1}{\mathrm{Pe}}\Delta u + \begin{pmatrix}0\\1\end{pmatrix}\nabla u = 0 \quad \text{in} \quad \Omega = (0,1)\times(0,1), \tag{5.10}$$

and subject to the following Dirichlet boundary conditions

$$\begin{aligned} u(0,y) &= -1/2,\\ u(1,y) &= 1/2,\\ u(x,0) &= x - 1/2,\\ u(x,1) &= 0. \end{aligned}$$

We abbreviate this first test case for the convection-diffusion problem as CD1. The problem features vertical flows with steep gradients near the upper boundary ($y = 1$) to satisfy the boundary conditions. As can be seen in figure 5.1, the boundary values on the vertical walls alter quickly at the upper boundary, from -1/2 to 0 on the left wall and from 1/2 to 0 on the right wall. This dramatic change of the solution u results in *exponential boundary layers* formation near the upper boundary, whose thickness is inversely proportional to the Peclet number. The similar problem has been described in [52] on $\Omega \in (-1,1)^2$ with Dirichlet boundary conditions producing more steeper gradients at upper corners. Erlangga and Nabben have solved this problem in [54] with MKLM solver using finite volume method along with the upwind scheme for flux terms.

We employee quadrilateral finite elements (Q_1 and Q_2) for the discretization of the domain. For the numerical results presented below, the mesh consisting of 16 regular squares with edge length $h = 1/4$ (as shown in figure 5.1), is taken as coarse mesh. The CD1 problem is solved for Peclet numbers $Pe = 20, 50, 100$ and 200 on uniform meshes with $32^2, 64^2, 128^2$ and 256^2 squares.



Level	Grid
3	4^2
4	8^2
5	16^2
6	32^2
7	64^2
8	128^2
9	256^2

Figure 5.1: **CD1:** Coarse mesh with boundary conditions (left); Number of grid cells at each refinement level (right)

Right preconditioned MLKM with (4,2,2) configuration is used on the left preconditioned system $\mathbf{M}^{-1}\mathbf{A} = \mathbf{M}^{-1}\mathbf{b}$. The maximum eigenvalue λ is set equal to one, the shifting parameter ω is varied to improve the guessed eigenvalue, and the results presented are for the optimum value of ω. On the other hand, MG with F-cycle, 4 pre- and post-smoothing steps and UMFPACK as a coarse grid solver is used. For MG with Jacobi smoother, a damping parameter of 0.7 is used. Standard bilinear/biquadratic grid transfer operators from finite element spaces are used as prolongation and restriction operators (deflation subspaces $\mathbf{Z}$ and $\mathbf{Y}$ in case of MLKM solver). The solvers are required to reduce the residual norm by six digits, i.e., the termination criterion for all the simulations is set to $\epsilon = 10^{-6}$ relative.

The first test results (table 5.2) show the number of iterations of the multilevel Krylov solver without any preconditioner ($\mathbf{M} = \mathbf{I}$). As is evident from the results, the MLKM method works fine even without the use of any preconditioner. The convergence rates are independent of the mesh parameter h, even for problems with higher Pe numbers. However, convergence rates of the solver without preconditioner are greatly influenced by the Pe number, with poor convergence rates for larger Pe number problems.

Level/ Pe	**20**	**50**	**100**	**200**
6	23	54	107	231
7	21	44	89	229
8	20	41	80	187
9	20 (0.22)	37 (0.44)	66 (0.86)	137 (2.22)

Table 5.2: **CD1:** MLKM solver without any preconditioner on Q_1 finite element discretization. Time (in seconds) taken by solver at level 9 is also shown.

Next, the numerical tests are performed for different preconditioners/smoothers with bilinear as well as biquadratic finite element approximations. The number of iterations needed for the solvers to reach the convergence criterion for the above numerical tests are shown in the tables 5.3-5.6.

MLKM with point Jacobi preconditioning shows convergence rates that are bounded independent of the discretization parameter h associated with mesh refinements, for both bilinear (table 5.3) as well as biquadratic (table 5.4) finite element discretizations. Moreover, on suitably refined meshes, the convergence rates of MLKM/Jacobi solver are also Peclet number independent, for the range of Peclet numbers considered. On the other hand, multigrid method (with Jacobi smoothing) based on standard Galerkin discretization leads to a divergent method for the convection dominated problems.

Level/ Pe	**20**	**50**	**100**	**200**
		MLKM		
6	9	12	21	47
7	9	9	12	25
8	9	9	9	13
9	9 (0.16)	9 (0.16)	9 (0.16)	9 (0.16)
		MG		
6	4	div	div	div
7	4	div	div	div
8	4	div	div	div
9	4 (0.11)	div	div	div

Table 5.3: **CD1:** Comparison of MLKM/Jacobi ($\omega = 1.1$) and MG/Jacobi on Q_1 finite element discretization. Time taken by solvers at level 9 is also shown.

For the case of point Gauß-Seidel preconditioner/smoother (see tables 5.5 and 5.6), the convergence behavior of solvers is essentially similar to that of Jacobi case. Here again, MLKM exhibits convergence rates that are grid independent and almost Pe number independent (as long as the refinement level is not too small and the Pe number is not too large at the same time), both for Q_1 and

Level/ Pe	**20**	**50**	**100**	**200**
		MLKM		
6	11	12	16	34
7	11	11	12	20
8	11	11	11	14
9	11 (0.83)	11 (0.84)	11 (0.81)	11 (0.81)
		MG		
6	4	div	div	div
7	5	5	div	div
8	5	5	div	div
9	5 (0.73)	5 (0.76)	div	div

Table 5.4: **CD1:** Comparison of MLKM/Jacobi ($\omega = 1.0$) and MG/Jacobi on Q_2 finite element discretization. Time taken by solvers at level 9 is also shown.

Q_2 finite element approximations. Similarly, MG with Gauß-Seidel smoother is not robust and diverges for large Pe numbers.

We want to emphasize here that these problem parameters (h and Pe) independent convergence rates for MLKM are obtained without using any special reordering schemes, without using any stabilization schemes for the convective term, and without using any adaptively refined meshes in the vicinity of the upper boundary.

Figure 5.2 shows contour plots of the solution for Peclet numbers 20 and 200, produced by MLKM/Jacobi method on uniform meshes and without using any stabilization for the convective term. Note that MLKM/Jacobi without stabilization, produces solutions without any nonphysical spurious wiggles in the domain, and the solver nicely captures the exponential layered solution at the top boundary, even at higher Pe numbers. Notice the change in the width of the boundary layer with the change in Peclet number, which complies with the physics of the problem.

MLKM convergence rates are not very sensitive to the choice of the value of ω, and the solver works fine in many cases even if we simply take $\omega = 1.0$. In table 5.7, we show the number of iterations taken by MLKM solver to converge for *optimal* ω used in tables 5.3 and 5.5, and for $\omega = 1.0$; the results presented are at level 9 for $Q1$ finite element approximation. The results show that for both Jacobi and Gauß-Seidel preconditioners, the convergence rates of MLKM are not much altered if we take $\omega = 1.0$.

Level/ Pe	**20**	**50**	**100**	**200**
		MLKM		
6	9	12	21	50
7	8	9	13	22
8	8	8	9	13
9	8 (0.16)	8 (0.16)	8 (0.16)	9 (0.18)
		MG		
6	4	div	div	div
7	3	div	div	div
8	4	div	div	div
9	3 (0.10)	div	div	div

Table 5.5: **CD1:** Comparison of MLKM/Gauß-Seidel ($\omega = 0.7$) and MG/Gauß-Seidel on Q_1 finite element discretization. Time taken by solvers at level 9 is also shown.

Level/ Pe	**20**	**50**	**100**	**200**
		MLKM		
6	9	10	13	24
7	9	9	11	16
8	9	9	9	12
9	9 (0.82)	9 (0.81)	9 (0.80)	10 (0.88)
		MG		
6	3	3	div	div
7	3	3	div	div
8	3	3	div	div
9	3 (0.54)	3 (0.56)	div	div

Table 5.6: **CD1:** Comparison of MLKM/Gauß-Seidel ($\omega = 0.7$) and MG/Gauß-Seidel on Q_2 finite element discretization. Time taken by solvers at level 9 is also shown.

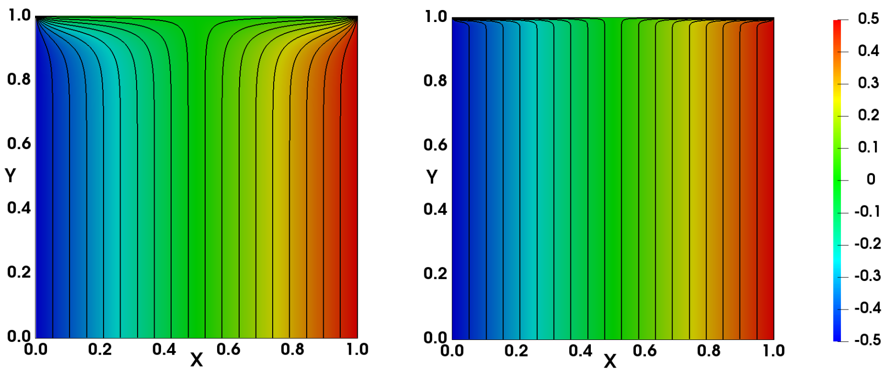


Figure 5.2: **CD1:** Contour plots for the solution of the convection-diffusion problem for *Pe* numbers 20 (left) and 200 (right) on 128^2 uniform mesh. MLKM/Jacobi without stabilization is used as a solver.

ω/Pe	**20**	**50**	**100**	**200**
		Jacobi		
Optimal	9	9	9	9
1.0	9	9	9	10
		Gauß-Seidel		
Optimal	8	8	8	9
1.0	10	10	10	11

Table 5.7: **CD1:** Number of iteration of MLKM for *optimal* ω, and for $\omega = 1.0$.

5.2.3 Test Problem 2 (unstructured meshes)

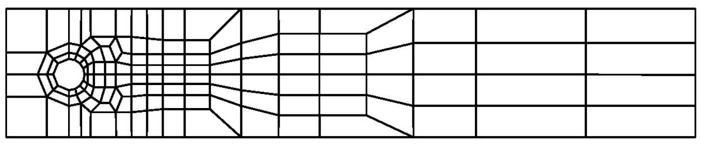
As a second test case, we solve the convection-diffusion equation in a rectangular channel with a circular obstacle. The circular obstacle may represent a heat source placed in the stream of the fluid, and the convection-diffusion problem then depicts the transport of heat in the channel. The mathematical model of the problem along with boundary conditions is as follows

$$-\frac{1}{\mathrm{Pe}}\Delta u + \begin{pmatrix}1\\0\end{pmatrix}\nabla u = 0 \quad \text{in} \quad \Omega, \tag{5.11}$$

with boundary conditions

$$\begin{aligned} u(0,y) &= 0 \\ u(x,0) &= 0 \\ u(x,0.41) &= 0 \\ u(x,y) &= 1 \quad \text{for } (x,y) \text{ on the circle.} \\ \partial_n u(2.2,y) &= 0 \quad \text{natural boundary condition} \end{aligned}$$

We abbreviate this second test case for the convection-diffusion problem as CD2. The computational domain is discretized using unstructured quadrilaterals, and the corresponding coarse level mesh is shown in the figure 5.3.



Level	Elements	Unknowns	
		Q_1	Q_2
5	33280	33696	133952
6	133120	133952	534144
7	532480	534144	2133248
8	2129920	2133248	8526336
9	8519680	8526336	34092032

Figure 5.3: **CD2:** Coarse mesh along with the problem size at each refinement level.

For Q_1 finite element discretization, MLKM with (4,2,2) configuration is used. With Q_2 finite elements, although MLKM (4,2,2) worked fine for lower Pe numbers, however, at higher Pe numbers the solver required more iterations at the coarse level to achieve better convergence rates. Therefore, for Q_2 approximations MLKM (4,2,5) configuration with 5 GMRES iterations at the coarse level are used. In the case of Jacobi preconditioner, the maximum eigenvalue of the preconditioned system is approximated as follows

$$\lambda_{max}(\mathbf{M}^{-1}\mathbf{A}) = \max_{i\in N} \sum_{j\in N} |a_{i,j}/a_{i,i}|,$$

and for Gauß-Seidel preconditioner, it is taken as equal to one. Reverse Cuthill Mckee renumbering is used with Gauß-Seidel preconditioner/smoother, which results in improved convergence rates for both MLKM and MG solvers. All other solvers and numerical settings are same as in the test case one. Similar to the first test case for the convection-diffusion problem, we solve the problem 5.11 for varying Pe numbers $(100, 200, 500, 1000,$ and $5000)$ and study the effect of convection on the performance of MLKM solver and compare it with the MG solver. Numerical results are shown for the solvers without any stabilization, as well as with the edge-oriented jump stabilization to stabilize the convective term in the problems involving higher Peclet numbers, see tables [5.8 - 5.15].

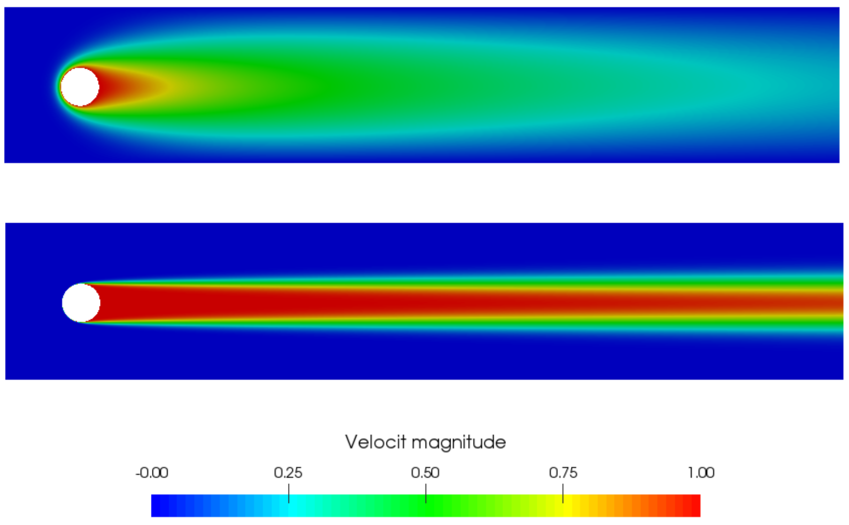


Figure 5.4: **CD2:** Velocity magnitude profiles for $Pe = 100$ (top) and $Pe = 5000$ (bottom).

Numerical results, in the second test case, are similar to the test case one; again for the convection dominated flows, the multilevel Krylov method with Jacobi or Gauß-Seidel preconditioning stands out as a more robust solver than the multigrid method. In this test problem also, MLKM is convergent and produces mesh parameter (h) independent convergence rates, even when no stabilization is used. For highly convective flows $(Pe = 5000)$ also, the method produces h independent convergence rates, without any stabilization. This problem size independent convergence behavior of the method is observed for both bilinear and biquadratic finite element approximations. On the contrary, multigrid is divergent for slightly large Peclet numbers, even when the edge-oriented jump stabilization is used.

The results also show that on appropriately refined meshes, the convergence behavior of the MLKM solver is independent of the Peclet number (for moderately large Peclet numbers). This means that convergence rates and the computational costs are the same for a range of moderate Peclet number problems (upto $Pe = 1000$). However, for very high values of Peclet numbers ($Pe = 5000$) further mesh refinements may produce similar convergence rates. Another important observation is that for highly convective flows, MLKM with diagonal preconditioning performs better than MLKM with Gauß-Seidel preconditioning. Numerical experiments also reveal that convergence rates of both MLKM and MG are not much improved by the use of edge-oriented jump stabilization, for the problem at hand.

Level/ Pe	**100**	**200**	**500**	**1000**	**5000**
			MLKM		
6	18	18	29	60	304
7	17	17	18	32	198
8	16	16	16	19	111
9	15 (37)	15 (37)	15 (37)	15 (37)	57 (166)
			MG		
6	15	Div	Div	Div	Div
7	15	Div	Div	Div	Div
8	17	Div	Div	Div	Div
9	18 (86)	Div	Div	Div	Div

Table 5.8: **CD2:** Comparison of MLKM/Jacobi and MG/Jacobi on Q_1 finite element discretization. Time taken by solvers at level 9 is also shown.

Table 5.16 shows time taken by MLKM and MG solvers to solve the problem 5.11 with $Pe = 100$, at various mesh refinement levels using Q_2 finite elements. With each uniform mesh refinement, the number of unknowns are increased four times. The time taken by both the solvers is also increased with a factor of approximately four at each level, indicating the linear computational complexity of both the solvers.

Level/ Pe	**100**	**200**	**500**	**1000**	**5000**
			MLKM		
6	18	19	31	58	220
7	17	17	20	30	140
8	16	16	16	19	78
9	15(64)	15(64)	15(64)	16(69)	43 (195)
			MG		
6	15	Div	Div	Div	Div
7	15	Div	Div	Div	Div
8	17	Div	Div	Div	Div
9	18 (183)	Div	Div	Div	Div

Table 5.9: **CD2:** Comparison of MLKM/Jacobi and MG/Jacobi on Q_1 finite element discretization with **EOJ stabilization**. Time taken by solvers at level 9 is also shown.

Level/ Pe	**100**	**200**	**500**	**1000**	**5000**
			MLKM		
6	14	14	72	176	not conv.
7	13	13	28	81	not conv.
8	12	12	18	40	not. conv
9	11(35)	11(35)	13 (41)	22(72)	not conv.
			MG		
6	3	Div	Div	Div	Div
7	3	Div	Div	Div	Div
8	3	Div	Div	Div	Div
9	3 (16)	Div	Div	Div	Div

Table 5.10: **CD2:** Comparison of MLKM/Gauß-Seidel and MG/Gauß-Seidel on Q_1 finite element discretization. Time taken by solvers at level 9 is also shown.

Level/ Pe	**100**	**200**	**500**	**1000**	**5000**
			MLKM		
6	14	14	21	38	133
7	13	13	14	25	110
8	13	13	13	17	86
9	12(56)	12(56)	12 (56)	13(61)	59 (301)
			MG		
6	5	9	39	Div	Div
7	4	7	29	Div	Div
8	3	6	43	Div	Div
9	3 (26)	4 (35)	70 (606)	Div	Div

Table 5.11: **CD2:** Comparison of MLKM/Gauß-Seidel and MG/Gauß-Seidel on Q_1 finite element discretization with **EOJ stabilization**. Time taken by solvers at level 9 is also shown.

Level/ Pe	**100**	**200**	**500**	**1000**	**5000**
			MLKM		
5	21	22	33	65	415
6	21	21	23	35	277
7	21	21	21	25	146
8	21 (82)	21 (82)	21 (82)	21 (82)	90 (418)
			MG		
5	7	Div	Div	Div	Div
6	6	Div	Div	Div	Div
7	6	Div	Div	Div	Div
8	6 (48)	Div	Div	Div	Div

Table 5.12: **CD2:** Comparison of MLKM/Jacobi and MG/Jacobi on Q_2 finite element discretization. Time taken by solvers at level 8 is also shown.

Level/ Pe	**100**	**200**	**500**	**1000**	**5000**
			MLKM		
5	21	22	32	64	395
6	21	21	23	35	242
7	21	21	21	27	116
8	21 (193)	21 (193)	21 (193)	21 (194)	76 (739)
			MG		
5	7	Div	Div	Div	Div
6	6	Div	Div	Div	Div
7	6	Div	Div	Div	Div
8	6 (144)	Div	Div	Div	Div

Table 5.13: **CD2:** Comparison of MLKM/Jacobi and MG/Jacobi on Q_2 finite element discretization with **EOJ stabilization**. Time taken by solvers at level 8 is also shown.

Level/ Pe	**100**	**200**	**500**	**1000**	**5000**
			MLKM		
5	21	23	47	367	not conv.
6	21	21	32	130	not conv.
7	21	21	23	85	not conv.
8	21 (93)	21 (93)	21 (93)	51 (237)	not conv.
			MG		
5	4	Div	Div	Div	Div
6	4	Div	Div	Div	Div
7	4	Div	Div	Div	Div
8	4 (30)	Div	Div	Div	Div

Table 5.14: **CD2:** Comparison of MLKM/Gauß-Seidel and MG/Gauß-Seidel on Q_2 finite element discretization. Time taken by solvers at level 8 is also shown.

Level/ Pe	**100**	**200**	**500**	**1000**	**5000**
			MLKM		
5	21	23	40	141	not conv.
6	21	21	30	50	346
7	21	21	23	33	185
8	21 (197)	21 (198)	21 (198)	26 (259)	125 (1252)
			MG		
5	4	Div	Div	Div	Div
6	4	Div	Div	Div	Div
7	4	Div	Div	Div	Div
8	4 (70)	Div	Div	Div	Div

Table 5.15: **CD2:** Comparison of MLKM/Gauß-Seidel and MG/Gauß-Seidel on Q_2 finite element discretization with **EOJ stabilization**. Time taken by solvers at level 8 is also shown.

	MLKM			Multigrid		
DOF	Iter.	Time	Factor	Iter.	Time	Factor
133952	21	1.16	-	7	0.60	-
534144	21	4.51	3.9	6	2.38	4.0
2133248	21	19.38	4.3	6	11.10	4.7
8526336	21	82.12	4.2	6	50.12	4.51

Table 5.16: **CD2:** Time taken by MLKM/Jacobi and MG/Jacobi solvers at various mesh levels, for solving the problem with $Pe = 100$ using Q_2 finite elements discretization.

6 Anisotropic Diffusion Problem

The multigrid method is considered an *optimal* solver for solving the isotropic diffusion problem, with convergence rates oblivious to the problem size. However, it is a well-known fact that these MG features no longer exist as soon as anisotropy is introduced into the problem, either from the differential operator or the underlying mesh of the problem (cf. [70, chapter 10],[4], [131, chapter 5]). The reason for this poor performance is that, contrary to the isotropic case, the error after relaxation is not smooth in every direction and cannot be well represented at the coarse mesh.

Various remedies have been proposed in the literature in an attempt to improve the convergence rates of MG solver for solving such anisotropic problems. One such remedy is to use semi- or directional-coarsening of the meshes instead of full-coarsening [100, 101]. In this approach, the grid is coarsened only in the direction in which the error is smoothed out. Another approach is to modify the relaxation and use line-smoothing (also called block smoothing) on the fully coarsened meshes [19]. Mavriplis [94] has combined directional-coarsening with line-smoothing and reported the improved convergence behavior of multigrid solver against anisotropies. However, both directional-coarsening and line-smoothing have limitations and drawbacks. Directional-coarsening (or semi-coarsening for structured meshes) produce coarse meshes with higher complexity than those from full-coarsening. Full-coarsening reduces mesh complexity between successive meshes four times in 2D case and eight times in 3D problems. However, the directional-coarsening reduce mesh complexity only two times for both 2D and 3D cases. The resulting coarse meshes have much higher computational and memory requirements, particularly for 3D problems. Line-smoothing performs well on structured meshes since in this case, it produces block tridiagonal matrices which can be efficiently solved. However, for unstructured grids, the grid lines do not exist, and it is not straightforward to implement line-smoothing on such grids. Moreover, the techniques mentioned above work well only when the anisotropy is parallel to coordinate axis or aligned with the grid. However, if the anisotropy is not aligned with the grid or coordinate axis, these techniques are not very useful.

In this chapter, we solve the operator-based anisotropic diffusion problem as well as mesh-based anisotropic diffusion problem; however, we do not use any of the remedies mentioned above for handling anisotropies. Instead, we solve the problems using standard full-coarsening and pointwise smoothing and compare how robustly the multilevel solvers, namely MLKM and MG, can handle anisotropies with such standard solver ingredients.

6.1 Operator-based Anisotropy

Directionally dependant or anisotropic diffusion occurs in various science and engineering applications. Composite materials often exhibit anisotropic heat and mass diffusion; for instance, water is absorbed in the wind turbine blades made of composite material through anisotropic diffusion [115]. Polymers with long-chain structures diffuse more easily in the direction of the chain axis than in the transverse direction [85]. A stretched membrane subject to the transverse loading experiences an anisotropic deflection. In microfluidic devices, the diffusion is more dominant along the channel length with a limited diffusion in the direction normal to the channel length [43]. A mathematical model of the anisotropic diffusion problem can be obtained by taking $\mathcal{L} = -\nabla \, . \, (\mathbf{G} \, \nabla)$ in equation (2.1). Assuming homogeneous Dirichlet boundary conditions, the anisotropic diffusion problem is given by

$$-\nabla \cdot (\mathbf{G}\nabla u) = f \qquad \text{in} \qquad \Omega, \tag{6.1a}$$

$$u = 0 \qquad \text{on} \qquad \Gamma, \tag{6.1b}$$

with $u : \Omega \subseteq \mathbb{R}^2 \mapsto \mathbb{R}$, $f \in L^2(\Omega)$, and $\mathbf{G} \in \mathbb{R}^{2\times 2}$ is the diffusion coefficient matrix, which introduces anisotropic diffusion along some vector field $\mathbf{v}$, and is defined as:

$$\mathbf{G} = \alpha \; \mathbf{v}\mathbf{v}^T + \beta \; (I - \mathbf{v}\mathbf{v}^T)$$

For a vector field $\mathbf{v} = (v_1 \; , \; v_2)^T$, anisotropic diffusion matrix can be written as:

$$\mathbf{G} = \begin{pmatrix} (\alpha-\beta) \; v_1 v_1 + \beta & (\alpha-\beta) \; v_1 v_2 \\ (\alpha-\beta) \; v_1 v_2 & (\alpha-\beta) \; v_2 v_2 + \beta \end{pmatrix}.$$

For the existence of the solution of equation (6.1), it is necessary that $\mathbf{G}$ be symmetric positive definite.

The weak form of the anisotropic diffusion equation (6.1) is given as follows: *Find* $u \in \mathcal{H}_0^1$ *such that*

$$a(v,u) = b(v) \qquad \forall v \in \mathcal{H}_0^1, \tag{6.2}$$

where $a(.,.) : \mathcal{H}^1(\Omega) \times \mathcal{H}^1(\Omega) \mapsto \mathbb{R}$ is the bilinear form given by

$$a(v,u) := \int_\Omega \nabla v . \mathbf{G} \nabla u. \tag{6.3}$$

In anisotropic diffusion $a(v,u) = a(u,v)$, therefore bilinear form is symmetric, but the presence of anisotropic coefficient affects the eigenvalues and therefore the condition number of the resulting matrix. The linear form $b : \mathcal{H}^1(\Omega) \mapsto \mathbb{R}$ on the right side of equation (6.2) has the form

$$b(v) := \int_\Omega v f. \tag{6.4}$$

The discrete form of the equation (6.2) is as follows:

Find $u_h \in \mathcal{S}_0^h$ such that

$$a(v_h, u_h) = b(v_h) \qquad \forall v_h \in \mathcal{S}_0^h. \tag{6.5}$$

6.1.1 Numerical results for operator-based anisotropy

We present the performance comparison of multilevel Krylov solver with multigrid solver for the solution of anisotropic-diffusion problem (6.1), with the right-hand side f taken as 1. Domain considered is a square domain, $\Omega = (-1,1) \times (-1,1)$, with homogeneous Dirichlet boundary conditions, i.e., $u = 0$ on Γ. This model problem represents the anisotropic thermal diffusion in a square plate which is uniformly heated (constant source term on the right) with the edges of the plate kept at water-freezing temperature. Bilinear finite elements (Q_1) are used for the spatial discretization, and the coarse level contains 16 uniform quadrilaterals. Solvers are required to achieve six digits accuracy in relative error before termination, i.e., $\epsilon = 10^{-6}$, and the maximum number of iterations to reach this stopping criterion is set to 500. We present the convergence rates (ρ) of the solvers to reach the convergence criterion, with the convergence rate defined as

$$\rho := \left(\frac{\|r_k\|}{\|r_0\|}\right)^{\frac{1}{k}}, \tag{6.6}$$

where $\|r_k\|$ represents the l_2 norm of the residual r_k at the k^{th} iteration. Multilevel Krylov method with the following configuration is used:

- Right Preconditioned MLKM(4,2,2) is used.
- Largest eigenvalue of preconditioned matrix $\lambda_{max} = 1$.
- Relaxation parameter ω is varied and the results presented are for the optimum value of ω.

The following configuration for MG solver is used:

- MG with F-cycle is used
- UMFPACK is used as a coarse grid solver
- Number of pre and post smoothing steps = 4. Jacobi and ILU smoothers are damped with a factor of 0.7.

The anisotropic diffusion matrix $\mathbf{G}$ determines the direction and strength of anisotropy in equation (6.1). Depending on the form of $\mathbf{G}$, the following three cases may arise:

- Isotropic diffusion with no anisotropy, if

$$\mathbf{G} = \mathbf{I}.$$

- Normal or axis parallel anisotropy if

$$\mathbf{G} = \begin{pmatrix} 1 & 0 \\ 0 & \beta \end{pmatrix}.$$

- Rotated anisotropy if

$$\mathbf{G} = \alpha\ \mathbf{v}\mathbf{v}^T + \beta\ (I - \mathbf{v}\mathbf{v}^T).$$

Figure 6.1 shows different anisotropic diffusion cases for $\alpha = 1$ and $\beta = 100$, with an axis parallel anisotropic diffusion on the left, rotated anisotropic diffusion along $\mathbf{v} = (1\ ,\ 1)^T$ in the middle, and rotated anisotropic diffusion along $\mathbf{v} = (1\ ,\ 0.3)^T$ on the right of the figure.

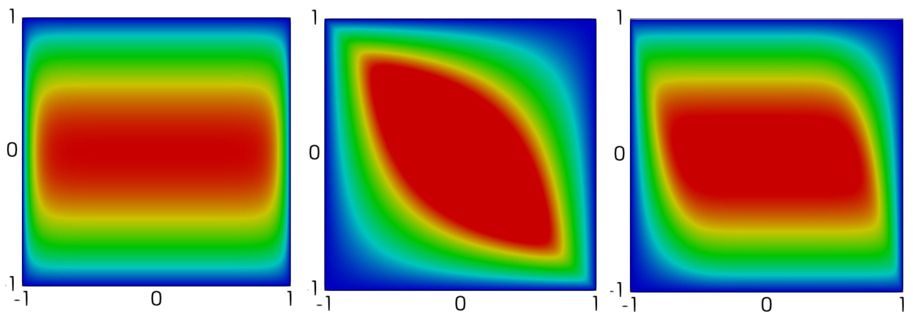


Figure 6.1: Anisotropic diffusion cases: Axis parallel (left), Rotated anisotropy $\mathbf{v} = (1\ ,\ 1)^T$ (middle), Rotated anisotropy $\mathbf{v} = (1\ ,\ 0.3)^T$ (right)

Axis-parallel anisotropy

In the case of axis-parallel anisotropy, the results are presented for $\beta = 50, 100,$ and 500; tables [6.1-6.3] show the results with Jacobi, Gauß-Seidel, and ILU(0) preconditioners/smoothers. With all the preconditioners, MLKM solver produces convergent results for all the values of anisotropic diffusion coefficient β. However, MG is divergent with Jacobi smoother, while convergent with the Gauß-Seidel and ILU(0) smoothing. For larger values of the anisotropic diffusion coefficient($\beta = 500$), MG/Gauß-Seidel has poor convergence rates than MLKM/Gauß-Seidel and in comparison takes four times more computational times. Although, MG/ILU(0) has better convergent rates as compared to that of MLKM/ILU(0), the total CPU times of MLKM/ILU(0) are of the same order as that of MG/ILU(0). Moreover, for the axis-parallel grid anisotropy case, both MLKM and MG (when it converged) solvers produce problem size independent convergent rates.

	MLKM			MG		
Lev/ β	**50**	**100**	**500**	**50**	**100**	**500**
6	0.80	0.85	0.91	div	div	div
7	0.80	0.85	0.93	div	div	div
8	0.80	0.85	0.93	div	div	div
9	0.80	0.85	0.93	div	div	div
Time	5.4	8.2	20.9	-	-	-

Table 6.1: Convergence rates of MLKM/Jacobi and MG/Jacobi for axis-parallel anisotropic diffusion.

	MLKM			MG		
Lev/ β	**50**	**100**	**500**	**50**	**100**	**500**
6	0.63	0.72	0.84	0.56	0.74	0.93
7	0.64	0.73	0.86	0.57	0.75	0.94
8	0.64	0.73	0.86	0.57	0.75	0.94
9	0.64	0.73	0.86	0.57	0.75	0.94
Time	3.1	4.3	10.3	4.1	6.0	37.7

Table 6.2: Convergence rates of MLKM/Gauß-Seidel and MG/Gauß-Seidel for axis-parallel anisotropic diffusion.

	MLKM			MG		
Lev/ β	**50**	**100**	**500**	**50**	**100**	**500**
6	0.16	0.16	0.16	7E-4	4E-4	1E-4
7	0.16	0.16	0.16	1E-3	9E-4	3E-4
8	0.16	0.16	0.16	2E-3	2E-3	6E-4
9	0.16	0.16	0.16	2E-3	2E-3	1E-3
Time	0.95	0.94	0.94	0.58	0.58	0.58

Table 6.3: Convergence rates of MLKM/ILU(0) and MG/ILU(0) for axis-parallel anisotropic diffusion.

Rotated anisotropy

Next, we discuss the case where the anisotropy direction vector $\mathbf{v}$ is not aligned with the grid lines but is at some rotation (angle) to them. In tables [6.4-6.9], we present the results for rotated anisotropic diffusion with the direction vectors $\mathbf{v} = (1\ ,\ 1)^T$ and $\mathbf{v} = (1\ ,\ 0.3)^T$. In axis parallel anisotropy, the diffusion is essentially 1D, and the coefficient matrix becomes tridiagonal for row-wise renumbering. ILU(0) becomes exact in this case, and results in excellent convergence of MG/ILU(0) solver. For the rotated anisotropy case, no renumbering scheme results into an exact ILU(0), and the convergence rates of MG with ILU(0) smoothing are significantly poor for the rotated anisotropy case as compared to the axis parallel anisotropy. This is evident from the results in tables 6.6 and 6.9.

For the rotated anisotropy case, the convergence rates of MLKM are better than MG with all the preconditioners/ smoothers tested. Moreover, MLKM produces (almost) grid size independent convergence rates, even with the weak preconditioner like Jacobi (See Tables 6.4 and6.7). On the contrary, numerical results show that multigrid loses h independent convergence behavior for higher values of anisotropic diffusion coefficient ($\beta > 100$), even when strong smoother like ILU(0) is used (Tables 6.6 and 6.9). Numerical results also depict that the increase in the anisotropy negatively influences the performance of both the solvers. However, MLKM is less sensitive to this change in anisotropy as compared to MG; this can also be seen from the graphs 6.2 and 6.3, which provide a visual comparison of the change in the convergence rates of the two solvers with the change in anisotropy.

	MLKM			**MG**		
Lev/ β	**100**	**500**	**1000**	**100**	**500**	**1000**
6	0.39	0.42	0.43	0.60	0.72	0.73
7	0.39	0.43	0.43	0.62	0.78	0.81
8	0.39	0.43	0.44	0.61	0.82	0.86
9	0.39	0.43	0.45	0.61	0.84	0.88
Time	1.2	1.4	1.5	3.5	9.6	13.5

Table 6.4: Convergence rates of MLKM/Jacobi and MG/Jacobi for anisotropic diffusion along $\mathbf{v} = (1\ ,\ 1)^T$

	MLKM			**MG**		
Lev/ β	**100**	**500**	**1000**	**100**	**500**	**1000**
6	0.41	0.44	0.46	0.45	0.59	0.62
7	0.41	0.46	0.47	0.47	0.68	0.71
8	0.41	0.46	0.48	0.46	0.73	0.78
9	0.41	0.46	0.48	0.46	0.75	0.82
Time	1.6	1.8	1.9	3.5	9.6	13.5

Table 6.5: Convergence rates of MLKM/Gauß-Seidel and MG/Gauß-Seidel for anisotropic diffusion along $\mathbf{v} = (1\ ,\ 1)^T$.

	MLKM			MG		
Lev/ β	**100**	**500**	**1000**	**100**	**500**	**1000**
6	0.22	0.27	0.28	0.21	0.35	0.37
7	0.22	0.29	0.31	0.24	0.46	0.50
8	0.21	0.30	0.32	0.23	0.53	0.60
9	0.21	0.30	0.32	0.23	0.56	0.65
Time	1.1	1.4	1.5	2.1	4.9	6.8

Table 6.6: Convergence rates of MLKM/ILU(0) and MG/ILU(0) for anisotropic diffusion along $\mathbf{v} = (1\ ,\ 1)^T$.

	MLKM			MG		
Lev/ β	**100**	**500**	**1000**	**100**	**500**	**1000**
6	0.69	0.73	0.74	0.64	0.77	0.79
7	0.69	0.74	0.75	0.65	0.82	0.84
8	0.69	0.74	0.76	0.65	0.85	0.88
9	0.69	0.74	0.76	0.65	0.86	0.90
Time	3.0	3.7	4.1	3.8	11.1	16.4

Table 6.7: Convergence rates of MLKM/Jacobi and MG/Jacobi for anisotropic diffusion along $\mathbf{v} = (1\ ,\ 0.3)^T$.

	MLKM			MG		
Lev/ β	**100**	**500**	**1000**	**100**	**500**	**1000**
6	0.63	0.68	0.69	0.42	0.59	0.62
7	0.62	0.68	0.69	0.43	0.67	0.71
8	0.62	0.68	0.69	0.43	0.71	0.77
9	0.62	0.68	0.69	0.43	0.73	0.80
Time	2.7	3.4	4.1	2.5	6.4	9.3

Table 6.8: Convergence rates of MLKM/Gauß-Seidel and MG/Gauß-Seidel for anisotropic diffusion along $\mathbf{v} = (1\ ,\ 0.3)^T$.

	MLKM			**MG**		
Lev/ β	**100**	**500**	**1000**	**100**	**500**	**1000**
6	0.34	0.40	0.41	0.27	0.44	0.46
7	0.34	0.41	0.42	0.28	0.52	0.57
8	0.34	0.41	0.43	0.28	0.57	0.64
9	0.34	0.41	0.43	0.28	0.59	0.69
Time	1.6	1.9	1.9	2.2	5.3	7.5

Table 6.9: Convergence rates of MLKM/ILU(0) and MG/ILU(0) for anisotropic diffusion along $\mathbf{v} = (1\ ,\ 0.3)^T$.

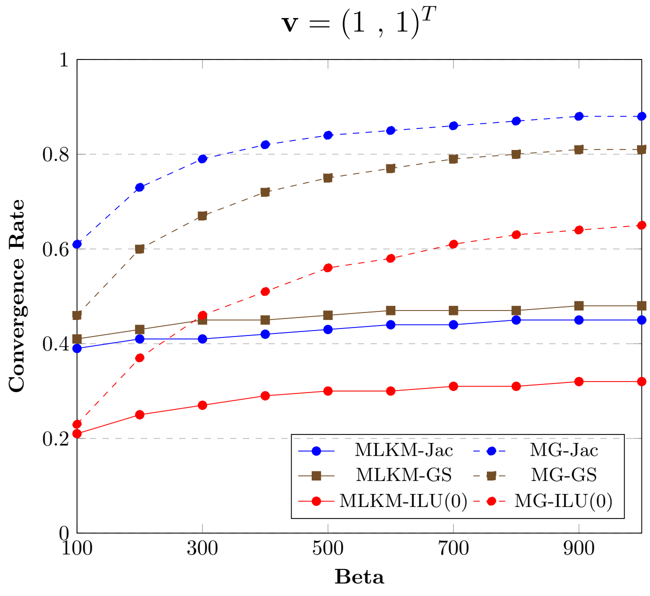


Figure 6.2: Comparison of convergence rates of MLKM and MG solver at level 9, for various values of β in anisotropic diffusion direction vector $\mathbf{v} = (1\ ,\ 1)^T$

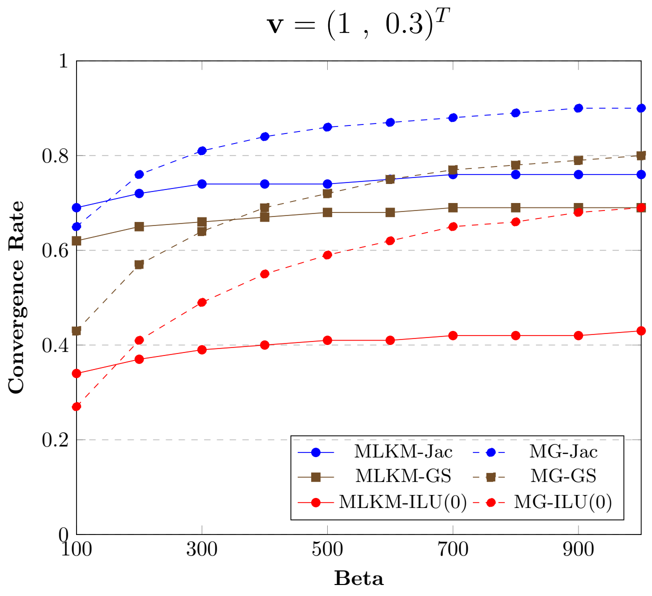


Figure 6.3: Comparison of convergence rates of MLKM and MG solver at level 9, for various values of β in anisotropic diffusion direction vector $\mathbf{v} = (1\ ,\ 0.3)^T$

6.2 Grid-based Anisotropy

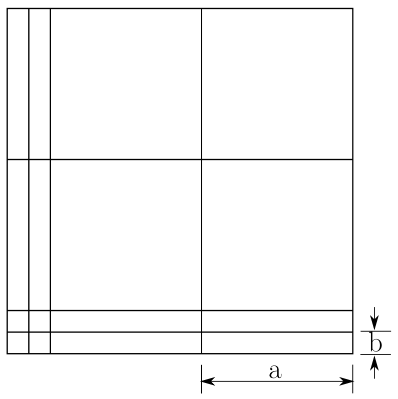
Anisotropic diffusion can also occur due to the underlying anisotropic meshes. These meshes are useful to approximate the functions that have high gradients in a specific direction, such as in boundary layers. In this case, anisotropic meshes have typically a smaller dimension in the direction of high gradients and larger dimension in the orthogonal direction. These meshes are also used to discretize high aspect ratio domains.

6.2.1 Numerical results for grid-based anisotropy

In this section, a performance comparison of the multilevel Krylov method with multigrid solvers is presented for the solution of the diffusion equation

$$-\ \Delta u = f \tag{6.7}$$

on the anisotropic grid shown in figure 6.4. Right-hand side f in equation 6.7 is chosen to produce the solution $u = (x-1)(x+1)(y-1)(y+1)$. Domain considered is a square domain, $\Omega = (-1,\ 1)^2$, with homogeneous Dirichlet boundary conditions, i.e., $u = 0$ on Γ. All the numerical settings and configurations of MLKM and MG solver are kept the same as that for the operator-based anisotropic diffusion case.



Level	DOF (Q_1)
6	4225
7	16641
8	66049
9	263169

Figure 6.4: Anisotropic coarse mesh with aspect ratio a/b (left); Number of degrees of freedom at each refinement level (right)

Results for grid-anisotropy are very similar to the axis-parallel operator-based anisotropy. Here again, MLKM with a weak Jacobi preconditioner results in a convergent solver on anisotropic grids, whereas MG with such a diagonal smoothing is divergent even for small aspect ratios. MLKM/Gauß-Seidel produces better convergent rates and computational times than MG/Gauß-Seidel, particularly on the meshes with larger aspect ratios. For example, on a mesh with an aspect ratio of 31, MLKM/Gauß-Seidel takes five times less time to converge as compared to MG/Gauß-Seidel solver (Table 6.11). Similar to the axis-parallel anisotropy case, with the row-wise reordering, MG/ILU(0) has better convergence rates than MLKM/ILU(0), however, the CPU times of both the solvers are of the same order.

	MLKM			**MG**		
Lev/ AR	**7**	**15**	**31**	**7**	**15**	**31**
6	0.74	0.82	0.84	div	div	div
7	0.75	0.83	0.87	div	div	div
8	0.74	0.84	0.88	div	div	div
9	0.73	0.84	0.88	div	div	div
Time	3.5	6.6	9.5	-	-	-

Table 6.10: Convergence rates of MLKM/Jacobi and MG/Jacobi for solving diffusion equation on anisotropic grids.

	MLKM			**MG**		
Lev/ AR	**7**	**15**	**31**	**7**	**15**	**31**
6	0.56	0.69	0.76	0.41	0.75	0.89
7	0.57	0.72	0.79	0.43	0.78	0.92
8	0.57	0.72	0.80	0.42	0.80	0.93
9	0.57	0.72	0.80	0.42	0.80	0.94
Time	2.1	4.1	5.7	2.3	9.3	29.7

Table 6.11: Convergence rates of MLKM/Gauß-Seidel and MG/Gauß-Seidel for solving diffusion equation on anisotropic grids.

	MLKM			**MG**		
Lev/ AR	**7**	**15**	**31**	**7**	**15**	**31**
6	0.17	0.19	0.17	2.6E-03	4.5E-03	6.1E-03
7	0.17	0.19	0.19	3.9E-03	5.9E-03	9.8E-03
8	0.17	0.18	0.23	4.3E-03	6.2E-03	1.9E-02
9	0.17	0.17	0.23	4.3E-03	5.9E-03	9.9E-03
Time	0.8	0.9	1.0	0.5	0.5	0.6

Table 6.12: Convergence rates of MLKM/ILU(0) and MG/ILU(0) using row-wise renumbering, for solving diffusion equation on anisotropic grids.

7 A monolithic FEM/MLKM solver for Navier-Stokes Equations

The Navier-Stokes equations (NSE) are a set of nonlinear partial differential equations that describe the flow of fluids. Their importance can be understood from the ubiquitous nature of the fluids. These equations are used to study the blood flow inside the human body, to model the weather, to predict the effect of global warming on climate, to design the aircraft, ships, and cars, and to describe many other significant engineering and scientific phenomena. We extend our work form the previous chapters of solving the scalar problems using FEM/MLKM solver, and present in this chapter a new monolithic FEM/multilevel Krylov subspace solver for the numerical solution of stationary incompressible Navier-Stokes equations in a fully coupled way.

We present the stationary incompressible NSE and briefly discuss their finite element discretization. Since the continuity equation does not involve the pressure unknown, the FEM discretization of Navier-Stokes equations is not straightforward. One cannot freely choose the discrete pressure and velocity approximation spaces independent of each other. Their choice must satisfy the compatibility condition, called the *LBB (Ladyzhenskaya-Babuška-Brezzi)* condition, which establishes the relation between velocity and pressure approximations for the well-posedness of the problem. We briefly discuss the LBB condition and our choice of admissible mixed finite elements.

The resulting algebraic system is nonlinear that can only be solved using iterations. We present our new coupled multilevel Krylov method, as an alternative to the coupled geometric multigrid algorithm, to solve the nonlinear algebraic system resulting from the FEM discretization of steady-state incompressible Navier-Stokes equations. The solver involves an outer iteration to handle the nonlinearity; we discuss the Picard and the Newton iteration techniques to linearize the system of nonlinear equations. Linearization results in indefinite saddle point type linear systems with zeros on the diagonal, and solving such linear systems poses a great challenge for the linear solvers. We use MLKM solver discussed in chapter (4) to solve these linear systems in a monolithic way.

7.1 The Navier-Stokes Problem

Consider the steady-state incompressible Navier-Stokes equations governing the flow of a Newtonian, viscous fluid:

$$-\nu\Delta\mathbf{u} + \mathbf{u}.\nabla\mathbf{u} + \nabla p = \mathbf{f} \quad \text{in} \quad \Omega, \tag{7.1a}$$

$$\nabla.\mathbf{u} = 0 \quad \text{in} \quad \Omega, \tag{7.1b}$$

where

$\Omega \subset \mathbb{R}^d$	$-$ is the bounded flow domain with a sufficiently regular boundary Γ, ($d = 2$ or 3 is a spatial dimension),
$\mathbf{u}$	$-$ is the velocity of the fluid,
p	$-$ is the pressure field,
ν	$-$ is the kinematic viscosity constant,
f	$-$ is the forcing term,
Δ	$-$ is the Laplacian operator,
∇	$-$ is the gradient and
$\nabla.$	$-$ is the divergence operator.

The equation (7.1a) is called the *momentum equation* and represents the conservation of momentum of the fluid, while equation (7.1b) is called the *continuity equation* or *incompressibility constraint* and represents the conservation of mass.

Similar to the convection-diffusion equation in chapter 5, it is useful in the study of the Navier-Stokes equations to have some quantitative measure of relative strength of inertial forces of the flow (convection) and the viscous forces in the flow (viscous diffusion). To this end, the Navier-Stokes equations are normalized with respect to the size of domain and velocity magnitude by introducing the dimensionless parameter called *Reynolds number* defined by

$$Re = \frac{LU}{\nu}, \tag{7.2}$$

where L is a characteristic length and U is a characteristic velocity. Using (7.2), the equation (7.1) can be rewritten into the normalized Navier-Stokes equations as follows

$$-\frac{1}{Re}\Delta\mathbf{u} + \mathbf{u}.\nabla\mathbf{u} + \nabla p = \mathbf{f} \quad \text{in} \quad \Omega, \tag{7.3a}$$

$$\nabla.\mathbf{u} = 0 \quad \text{in} \quad \Omega. \tag{7.3b}$$

The equations (7.1) or (7.3) form a system of second order partial differential equations in space and to solve these equations it is necessary to prescribe boundary conditions on the whole boundary of the domain. We consider here the Dirichlet type boundary condition where the velocity field is given on the boundary as

$$\mathbf{u} = \mathbf{g} \quad \text{on} \quad \Gamma_D. \tag{7.4}$$

The special case of

$$\mathbf{u} = \mathbf{0} \iff \mathbf{u}\,.\,\mathbf{n} = 0, \qquad \mathbf{u}\,.\,\mathbf{t} = 0,$$

is called the *no-slip* boundary condition. The velocity component $\mathbf{u}\,.\,\mathbf{n} = 0$ at the boundary means that there is no fluid passing through the boundary and the component $\mathbf{u} \,.\, \mathbf{t} = 0$ states that the fluid does not slip along the wall. If the Dirichlet boundary condition is specified everywhere on the boundary of Ω, then the pressure solution to the Navier-Stokes equations is determined only up to an arbitrary additive constant (since the governing equations, in this case, contain only the gradient of the pressure). To fix the constant, a supplemental condition is imposed requiring the vanishing of the integral mean value pressure

$$\int_{\Omega} p = 0.$$

Moreover, it follows from the integration of the continuity equation over the domain, followed by the application of the divergence theorem that the pure Dirichlet boundary condition should also satisfy the following *compatibility condition*

$$0 = \int_{\Omega} \nabla.\mathbf{u} = \int_{\Gamma} \mathbf{u} \,.\, \mathbf{n} = \int_{\Gamma} \mathbf{g} \,.\, \mathbf{n}, \tag{7.5}$$

or equivalently

$$\int_{\Gamma_{in}} \mathbf{g} \,.\, \mathbf{n} - \int_{\Gamma_{out}} \mathbf{g} \,.\, \mathbf{n} = 0. \tag{7.6}$$

This means that the pure Dirichlet boundary conditions should ensure that the net flow of the fluid through the domain boundaries is equal to zero. This problem can be avoided by replacing the Dirichlet condition at the outflow with a Neumann condition (typically *do nothing* boundary condition) that automatically adjusts $\mathbf{u} \,.\, \mathbf{n}$ at the outflow boundary to satisfy the equation (7.5). For the sake of simplicity in presentation, we next consider the Navier-Stokes equations with homogeneous Dirichlet boundary conditions only, for which case the compatibility condition is naturally satisfied.

Many fluid flows involve small Reynolds numbers ($Re \ll 1$), for instance, flows involving small length scales (fluid flow in MEMs), flows with very small velocities (creeping flow), or flows involving highly viscous fluids (honey). In such low Reynolds number flows, the inertial forces are negligible compared to the viscous forces; this suggests that the convective term can be neglected in Navier-Stokes equations. The resulting system of equations is linear, called the *Stokes equations* and the flow is termed *Stokes flow*

$$\begin{aligned} -\nu\Delta\mathbf{u} + \nabla p &= \mathbf{f} \quad \text{in} \quad \Omega, && (7.7\text{a}) \\ \nabla \,.\, \mathbf{u} &= 0 \quad \text{in} \quad \Omega, && (7.7\text{b}) \\ \mathbf{u} &= 0 \quad \text{on} \quad \Gamma. && (7.7\text{c}) \end{aligned}$$

7.2 Weak Formulation

To obtain the weak form of the Navier-Stokes equations, we need the following usual Lebesgue and Sobolev spaces

$$\begin{aligned} L_0^2(\Omega) &= \big\{q : \quad q \in L^2(\Omega) \quad \text{with} \quad \int_{\Omega} q \;=\; 0\big\}, \\ \mathbf{H}_0^1(\Omega) &= \big\{\mathbf{v} : \quad \mathbf{v} \in H^1(\Omega)^d \;|\; \quad \mathbf{v} \;=\; 0 \text{ on } \Gamma\big\}. \end{aligned}$$

Multiply the momentum equation (7.1a) with the velocity test function $\mathbf{v}$ and the continuity equation with pressure test function q, and integrate the resulting equations over the domain Ω to obtain

$$-\int_\Omega \mathbf{v} \,.\, \nu\Delta\mathbf{u} + \int_\Omega \mathbf{v} \,.\, (\mathbf{u} \,.\, \nabla\mathbf{u}) + \int_\Omega \mathbf{v} \,.\, \nabla p \;=\; \int_\Omega \mathbf{v} \,.\, \mathbf{f}, \tag{7.8a}$$

$$\int_\Omega q\nabla \,.\, \mathbf{u} \;=\; 0. \tag{7.8b}$$

We reduce the strong continuity requirements on the weak solution $(\mathbf{u}, p)$ by shifting the derivatives to the test functions $(\mathbf{v}, q)$. For this let us first consider the second order term and apply integration by parts on it

$$\begin{aligned} -\int_\Omega \mathbf{v} \,.\, \nu\Delta\mathbf{u} \;&=\; -\int_\Gamma \nu(\nabla\mathbf{u} \,.\, \mathbf{v}) \,.\, \mathbf{n} \;+\; \int_\Omega (\nu\nabla\mathbf{u}) : (\nabla\mathbf{v}) \\ &=\; \nu\int_\Omega (\nabla\mathbf{u}) : (\nabla\mathbf{v}). \end{aligned} \tag{7.9}$$

Next take the pressure term, using the product rule we get

$$\begin{aligned} \int_\Omega \mathbf{v} \,.\, \nabla p \;&=\; \int_\Omega \nabla \,.\, (p\mathbf{v}) - p\nabla \,.\, \mathbf{v} \\ &=\; \int_\Gamma (p\mathbf{v}) \,.\, \mathbf{n} - \int_\Omega p\nabla \,.\, \mathbf{v} \qquad \text{using divergence theorem} \\ &=\; -\int_\Omega p\nabla \,.\, \mathbf{v}. \end{aligned} \tag{7.10}$$

Inserting (7.9) and (7.10) in equation (7.8a) results in the standard weak form of the Navier-Stokes equation as follows:

Find $\mathbf{u} \in \mathbf{H} := \mathbf{H}_0^1$ *and* $p \in L := L_0^2(\Omega)$ *such that*

$$\nu\int_\Omega (\nabla\mathbf{u}) : (\nabla\mathbf{v}) + \int_\Omega \mathbf{v} \,.\, (\mathbf{u} \,.\, \nabla\mathbf{u}) - \int_\Omega p\nabla \,.\, \mathbf{v} \;=\; \int_\Omega \mathbf{v} \,.\, \mathbf{f} \qquad \forall\, \mathbf{v} \in \mathbf{H}, \tag{7.11a}$$

$$\int_\Omega q\nabla \,.\, \mathbf{u} \;=\; 0 \qquad \forall\, q \in L. \tag{7.11b}$$

In equations (7.11a) and (7.11b), p and q do not involve any derivatives, and therefore it is sufficient that they are integrable but not necessarily continuous over the element boundaries. On the other hand, $\mathbf{u}$ and $\mathbf{v}$ involve derivatives, and thus not only $\mathbf{u}$ and $\mathbf{v}$ but also their derivatives are required to be integrable, which implies the continuity of $\mathbf{u}$ and $\mathbf{v}$ across the element boundaries. This fact is important in the selection of finite elements for Navier-Stokes equations that will be discussed later. Alternatively, the weak form of steady-state Navier-Stokes equations can be written as:

Find $\mathbf{u} \in \mathbf{H} := \mathbf{H}_0^1$ *and* $p \in L := L_0^2(\Omega)$ *such that*

$$\nu\; a(\mathbf{u}, \mathbf{v}) + c(\mathbf{u}, \mathbf{u}, \mathbf{v}) + b(p, \mathbf{v}) \;=\; (\mathbf{f}, \mathbf{v}) \qquad \forall\, \mathbf{v} \in \mathbf{H}, \tag{7.12}$$

$$b(q, \mathbf{u}) \;=\; 0 \qquad \forall\, q \in L, \tag{7.13}$$

where $a(.,.)$, $b(.,.)$ are the bilinear forms, $(.,.)$ the linear form and $c(.,.,.)$ the trilinear form, defined as follows

$$a(\mathbf{u},\mathbf{v}) := (\nabla\mathbf{u},\nabla\mathbf{v}), \tag{7.14a}$$

$$b(p,\mathbf{v}) := -(p,\nabla.\mathbf{v}), \tag{7.14b}$$

$$(\mathbf{f},\mathbf{v}) := \int_\Omega \mathbf{v}\ .\ \mathbf{f}, \tag{7.14c}$$

$$c(\mathbf{u},\mathbf{u},\mathbf{v}) := ((\mathbf{u}.\nabla)\mathbf{u},\mathbf{v}). \tag{7.14d}$$

7.3 Finite Element Discretization

We define the discretized form of the weak formulation by replacing the infinite dimensional function spaces $\mathbf{H}$ and L with finite dimensional spaces $\mathbf{H}_h$ and L_h respectively. In case of conforming finite elements, $\mathbf{H}_h \subset \mathbf{H}$ and $L_h \subset L$, and therefore the bilinear and trilinear forms in the discretized problem can be used in a similar way to the continuous problem. For the nonconforming finite elements $\mathbf{H}_h \not\subset \mathbf{H}$, one has to work with the bilinear and trilinear forms defined elementwise. Assuming conforming finite elements here (for nonconforming finite elements see [132]), the discrete weak form of Navier-Stokes problem reads as follows:

Find $\mathbf{u}_h \in \mathbf{H}_h$ *and* $p_h \in L_h$ *such that*

$$\nu\int_\Omega(\nabla\mathbf{u}_h):(\nabla\mathbf{v}_h)+\int_\Omega\mathbf{v}_h.(\mathbf{u}_h\ .\ \nabla\mathbf{u}_h)-\int_\Omega p_h\nabla.\mathbf{v}_h = \int_\Omega\mathbf{v}_h.\mathbf{f} \quad \forall\ \mathbf{v}_h\in\mathbf{H}_h, \tag{7.15}$$

$$\int_\Omega q_h\nabla.\mathbf{u}_h = 0 \qquad \forall\ q_h \in L_h. \tag{7.16}$$

Next, we introduce two sets of basis functions, a set of scalar basis $\{\psi_i\}$ for the pressure and a set of vector-valued basis $\{\boldsymbol{\phi}_i\}$ for the velocity vector. Generally, vector-valued basis functions are built from scalar finite element spaces. Given a set of scalar finite element basis functions $\{\phi_j\}_{j=1}^{n_u}$, the velocity basis functions for a two-dimensional problem can be written in the vector form as

$$\{\boldsymbol{\phi}_1,...,\boldsymbol{\phi}_{2n_u}\} = \{(\phi_1,0)^T,...,(\phi_{n_u},0)^T,(0,\phi_1)^T,...,(0,\phi_{n_u})^T\}. \tag{7.17}$$

The approximations of velocity and pressure can be written as

$$\mathbf{u}_h = \sum_{j=1}^{2n_u} u_j\boldsymbol{\phi}_j, \qquad p_h = \sum_{j=1}^{n_p} p_j\psi_j, \tag{7.18}$$

where n_u is the number of unknowns for one velocity component and n_p is the number of pressure unknowns. Letting $\mathbf{v}_h = \phi_i$ and $q_h = \psi_i$ in equations (7.15) and (7.16), we arrive at the standard Galerkin formulation.

Find $\mathbf{u}_h \in \mathbf{H}_h$ *and* $p_h \in L_h$ *such that*

$$\nu \int_\Omega (\nabla \mathbf{u}_h) : (\nabla \phi_i) + \int_\Omega \phi_i.(\mathbf{u}_h.\nabla \mathbf{u}_h) - \int_\Omega p_h \nabla.\phi_i = \int_\Omega \phi_i.\mathbf{f} \quad \text{for } i = 1, ..., 2n_u, \tag{7.19}$$

$$\int_\Omega \psi_i \nabla.\mathbf{u}_h = 0 \qquad \text{for } i = 1, ..., n_p. \tag{7.20}$$

In matrix format, above nonlinear system of equations can be written as

$$\begin{bmatrix} \mathbf{A} + \mathbf{N}(\mathbf{u}) & \mathbf{B}^T \\ \mathbf{B} & \mathbf{0} \end{bmatrix} \begin{bmatrix} \mathbf{U} \\ \mathbf{P} \end{bmatrix} = \begin{bmatrix} \mathbf{F} \\ \mathbf{0} \end{bmatrix}, \tag{7.21}$$

where $\mathbf{U}$ and $\mathbf{P}$ represent the unknown real coefficients $(u_j)_{j=1}^{2n_u}$ and $(p_j)_{j=1}^{n_p}$ respectively. The matrix $\mathbf{A}$ represents the discretization of Laplacian operator $\Delta(.)$, $\mathbf{N}(\mathbf{u})$ the discretization of nonlinear convective operator $\mathbf{u}.\nabla(.)$, $\mathbf{B}^T$ denotes the discretization of the negative of the gradient operator, and $\mathbf{B}$ the divergence operator.

7.4 Conditions for Elements selection

Systems of the form (7.21) are called *saddle point systems*. The zero block in the system comes from the discretization of the continuity equation and reflects the absence of pressure term in the continuity equation. However, we know from equation (7.20), the number of rows in the continuity equation is equal to pressure unknowns. If pressure unknowns are more than the velocity unknowns, the system (7.21) becomes rank deficient, and we have a singular system. So a necessary condition for the unique solution of the above saddle point problem to exist is that the pressure unknowns should never exceed the velocity unknowns ($n_p \leq 2n_u$), irrespective of the grid size. This condition deprives the liberty of choosing the pressure and velocity approximations independent of each other. To satisfy this condition, a rule of thumb is to approximate the pressure with the polynomial basis functions having a degree at least one less than the degree of velocity basis functions. However, this rule does not ensure that pressure unknowns are always less than velocity unknowns on any grid. It can be shown [125] that even if the pressure and velocity elements are chosen such that pressure has a lower degree polynomial approximation as compared with velocity, still the resulting coefficient matrix is singular.

A sufficient condition that elements must satisfy to ensure the well posedness of the saddle point problem (7.21) is a well-known compatibility condition between pressure and velocity ansatz functions called the inf-sup or *LBB (Ladyžhenskaya-Babuška-Brezzi)* condition [11, 22], given in discrete form as

$$\inf_{q_h \in L_h} \sup_{\mathbf{v}_h \in H_h} \frac{(q_h, \nabla.\mathbf{v}_h)}{\|q_h\|_{L_h} \|\mathbf{v}_h\|_{H_h}} \geq \gamma > 0, \tag{7.22}$$

where γ is a mesh independent parameter. The elements satisfying the above condition are called admissible elements. See [35] for the derivation of an exact

LBB condition. From a practical viewpoint, it is difficult to verify directly if the LBB condition (7.22) is satisfied by an element or not. In [59], Fortin has provided a more practical criterion to check the LBB condition, which states that:
Assume that the continuous LBB condition is satisfied and assume that there exists a continuous operator $\pi_h : \mathbf{H} \to \mathbf{H}_h$ satisfying:

$$\begin{cases} (\nabla \cdot (\mathbf{u} - \pi_h \mathbf{u}), q_h) = 0 & \forall\, q_h \in Q_h, \\ \|\pi_h \mathbf{u}\|_{\mathbf{H}_h} \le C \|\mathbf{u}\|_{\mathbf{H}} & \forall\, \mathbf{u} \in \mathbf{H}, \end{cases} \tag{7.23}$$

then the discrete LBB condition (7.22) is satisfied [23]. In [35], it is shown how to check the condition (7.23) on various elements.

7.5 Our choice of admissible elements

Finite elements used for the approximation of Navier-Stokes equations are categorized into two families; *Taylor-Hood* family with continuous pressure approximation and *Crouzeix-Raviart* family with discontinuous pressure approximation. Our choice of LBB-stable quadrilateral finite elements used in this thesis, namely conforming Q_2/P_1^{disc} (see [113, 66]) and nonconforming $\tilde{Q}_1/Q_0$ [112], belong to the Crouzeix-Raviart family. The mesh cell oriented Vanka smoothing/ preconditioning can be conveniently applied on such approximations with discontinuous pressure [80].

7.5.1 Nonconforming $\tilde{Q}_1/Q_0$ Element

The nonconforming $\tilde{Q}_1/Q_0$ finite element pair, also called *Rannacher–Turek element*, uses rotated multilinear (bilinear in 2D and trilinear in 3D) polynomial shape functions for velocity approximations in combination with piecewise constants for the pressure. The element was introduced by Rannacher and Turek in [112] and can be considered as a quadrilateral counterpart of the famous triangular Crouzeix-Raviart element [66]. Let us define a bilinear transformation $\psi_T : \hat{T} \to T$ from the reference element $\hat{T} = [-1, 1]^2$ to each element $T \in \mathcal{T}_h$. So, the rotated bilinear element is defined as

$$\tilde{Q}_1(T) := \{q \circ \psi_T^{-1} \; : q \in span\langle 1, x, y, x^2 - y^2 \rangle\}, \tag{7.24}$$

with four degrees of freedom at the midpoints of edges. The degrees of freedom are determined by either of the nodal functionals

$$F_E^{(a)} := |E|^{-1} \int_E v d\gamma, \qquad F_E^{(b)} := v(m_E), \tag{7.25}$$

where $E \subset \partial\mathcal{T}_h$ is the cell edge and m_E its midpoint. The related *parametric* finite element space for velocity approximation is given by

$$\begin{aligned} \mathbf{H}_h^{(a,b)} &:= S_h^{(a,b)} \times S_h^{(a,b)} \\ S_h^{(a,b)} &:= \{v_h \in L^2(\Omega) \mid v_{h|T} \in \tilde{Q}_1(T),\ \forall T \in \mathcal{T}_h, v_h \text{ continuous w.r.t. nodal} \\ &\qquad \text{functionals } F_{E_{ij}}^{(a,b)}(.),\ \forall E_{ij},\ \text{and } F_{E_{i0}}^{(a,b)}(v_h) = 0,\ \forall E_{i0}\}, \end{aligned} \tag{7.26}$$

with E_{ij} all inner edges shared by the elements i and j and E_{i0} the boundary edges. The pressure is approximated using Q_0 element defined by the piecewise constant functions from the space

$$L_h := \{q_h \in L_0^2(\Omega),\ q_{h|T} = constant,\ \forall T \in \mathcal{T}_h\}. \tag{7.27}$$

In [112], authors have mentioned that the stability and approximation properties of above defined parametric version of $\tilde{Q}_1/Q_0$ element deteriorate on highly distorted meshes. As an alternative, *non-parametric* version of the element can be used for which the reference space $\tilde{Q}_1(T) := \{q \in span\langle 1, \xi, \eta, \xi^2 - \eta^2\rangle\}$ is defined on each physical element T independently using the local coordinate system (ξ, η) obtained by joining the midpoints of T [112, 132]. Hence, the shape functions are defined directly on the physical element instead of the reference element. Although this ansatz is computationally more expensive than the parametric counterpart, it shows better performance on the grids with large anisotropies. In this thesis, we use the non-parametric version of $\tilde{Q}_1/Q_0$ with degrees of freedom corresponding to the mean values over the edges. See also [82] for details on the non-parametric $\tilde{Q}_1/Q_0$ element.

7.5.2 Conforming Q_2/P_1^{disc} Element

Let us define the finite element spaces $\mathbf{H}_h$ for the velocity and L_h for the pressure as follows

$$\mathbf{H}_h := \{\mathbf{v}_h \in [H_0^1(\Omega)]^2,\ \mathbf{v}_{h|T} \in [Q_2(T)]^2\ \forall T \in \mathcal{T}_h,\ \mathbf{v}_h = 0 \text{ on } \partial\Omega\}, \tag{7.28}$$

$$L_h := \{p_h \in L^2(\Omega),\ p_{h|T} \in P_1(T)\ \forall T \in \mathcal{T}_h\}, \tag{7.29}$$

with $Q_2(T)$ the biquadratic and $P_1(T)$ the linear space on the quadrilateral element T. Let us again define the bilinear transformation $\psi_T : \hat{T} \to T$ from the reference element $\hat{T} = [-1, 1]^2$ to arbitrary element T. The space Q_2 is defined on the physical element as

$$Q_2(T) := \{q \circ \psi_T^{-1}\ : q \in span\langle 1, x, y, xy, x^2, y^2, x^2y, xy^2, x^2y^2\rangle\}, \tag{7.30}$$

with nine nodes located at vertices, edge mid-points and centroid of quadrilateral. The space $P_1^{disc}(T)$ consists of linear polynomials discontinuous across the element boundaries and zero outside the element, defined by

$$P_1^{disc}(T) := \{q \circ \psi_T^{-1}\ : q \in span\langle 1, x, y\rangle\}. \tag{7.31}$$

The approximation comprises three degrees of freedom, corresponding to the function value and its two partial derivatives, located at the centroid of the quadrilateral. A problem with the above presented formulation is that when bilinear transformation ψ_T is applied to the linear function on the reference element, the resulting basis functions on the $\hat{T}$ are not full bilinear. As a consequence, the method is accurate only up to first order on general meshes, obtained for instance from certain mesh adaptation (see [5, 113])

$$\|p - p_h\| = \mathcal{O}(h).$$

To fix this problem, the reference space is defined on each physical element individually as

$$P_1(T) := \{1, \xi, \eta\},$$

by using the local coordinate system (ξ, η) obtained by joining the midpoints of opposite sides of T [5]. This non-parametric approximation satisfies the LBB condition and the second order approximation for the pressure is also recovered [113, 66]

$$\|p - p_h\| = \mathcal{O}(h^2).$$

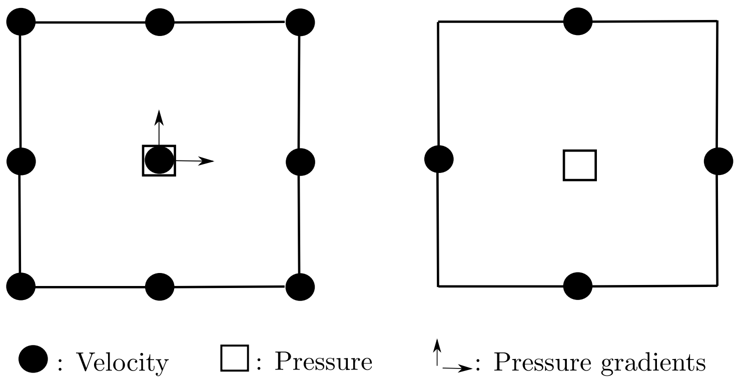


Figure 7.1: Nodal points for Q_2/P_1^{disc} (left) and $\tilde{Q}_1/Q_0$ finite elements

7.6 Monolithic Multilevel Krylov Subspace Solver

Although the selection of LBB stable finite elements ensures the well posedness of the discrete Navier-Stokes problem, the resulting saddle point type problem is non-symmetric and indefinite, and poses great challenges for the development of efficient and robust numerical solvers. Many iterative solution methods have been proposed in the literature to solve such saddle point problems that can be broadly categorized as *segregated or operator splitting* methods and *coupled* methods. Segregated methods split the linear system into smaller reduced systems, and solve the unknown variables, velocity and pressure, separately. On the other hand coupled methods treat the linear system as a whole, and solve both the pressure and velocity unknowns simultaneously. Both classes of methods have their own problem-dependent advantages and shortcomings, and one cannot say in general which one is better than the other. For instance, segregated methods work fine for unsteady flows with large Reynolds numbers that involve small time steps. However, for steady or unsteady flows with low Reynolds numbers, they face considerable problems, and the coupled methods are the preferred choice in this case[132, 136].

Previously, many authors have used multigrid as a multilevel solver to solve Stokes or Navier-Stokes equations in a coupled way (see [18, 141, 138, 148, 17, 16, 132, 146]). In this section, we present our monolithic multilevel Krylov subspace solver for the solution of steady incompressible Navier-Stokes equations in a coupled way, which can be considered as an alternative to the coupled multigrid method.

The presence of the convective term in the Navier-Stokes equations makes the algebraic system (7.21) nonlinear, and to handle such a non-linearity, the solver consists of an outer nonlinear iterative procedure. To derive this nonlinear iterative procedure, we start with the continuous form of the Navier-Stokes equations (7.1) and write them in the form of a nonlinear function

$$\mathbf{d}(\mathbf{u},p) \;=\; \begin{pmatrix} -\nu\Delta\mathbf{u} \;+\; \mathbf{u}.\nabla\mathbf{u} \;+\; \nabla p - \mathbf{f} \\ \nabla.\mathbf{u} \end{pmatrix}. \tag{7.32}$$

$\mathbf{d}(\mathbf{u},p)$ is a nonlinear defect function whose root is the solution to Navier-Stokes equations (7.1). Let $\mathbf{x}(\mathbf{u},p)$ be the root of $\mathbf{d}(\mathbf{u},p)$, the nonlinear defect correction iteration can be written as:

$$\mathbf{x}^{k+1} = \mathbf{x}^k + \tilde{\omega}^k(\mathbf{T}^{-1}(\mathbf{x}^k))\mathbf{d}(\mathbf{x}^k), \qquad k \in \mathbb{N}, \tag{7.33}$$

where $\mathbf{x}^{k+1}$ is the approximate solution at the $k+1$ iteration, $\mathbf{x}^k$ is the solution from the previous iteration, $\mathbf{T}(\mathbf{x}^k)$ is the preconditioner evaluated at the $\mathbf{x}^k$, and $\tilde{\omega}^k$ is the damping parameter.

The nonlinear iteration loop (7.33) is carried out in three steps as shown in the algorithm (7.1).

Algorithm 7.1 Nonlinear defect correction iteration

```
1: Given $\mathbf{x}^0, \mathbf{f}$
2: $k \leftarrow 0$
3: while ($\mathbf{x}^k$ not converged) do
4:     Calculate the nonlinear defect $\mathbf{d}(\mathbf{x}^k)$ using equation (7.32).
5:     Evaluate the correction term $\delta\mathbf{x}^k$ by a linear solve:
           $\delta\mathbf{x}^k := \mathbf{T}^{-1}(\mathbf{x}^k)\mathbf{d}(\mathbf{x}^k) \iff \mathbf{T}(\mathbf{x}^k)\delta\mathbf{x}^k = \mathbf{d}(\mathbf{x}^k).$   (7.34)
6:     Update the defect:
           $\mathbf{x}^{k+1} = \mathbf{x}^k + \tilde{\omega}^k\delta\mathbf{x}^k.$   (7.35)
7:     $k \leftarrow k+1$
8: end while
```

Linearization in equation (7.34) is commonly performed using either *Newton* or *fixed point* linearization methods. Next, we discuss both of these methods.

7.6.1 Newton linearization

In Newton method, the preconditioning operator $\mathbf{T}(\mathbf{x}^k)$ is the exact Fréchet-derivative ($\nabla\mathbf{d}(\mathbf{x}^k)$) of the continuous Navier-Stokes equations evaluated at the $\mathbf{x}^k$. Thus the correction term in equation (7.34) can be evaluated by solving the system

$$\nabla\mathbf{d}(\mathbf{x}^k)\delta\mathbf{x}^k = \mathbf{d}(\mathbf{x}^k). \tag{7.36}$$

The left-hand side in the above equation represents the directional derivative of the defect function $\mathbf{d}(\mathbf{x})$ at $\mathbf{x}^k$ along $\delta\mathbf{x}^k$. Using the definition of directional derivative, we can write

$$\nabla \mathbf{d}(\mathbf{u}^k, p^k)(\delta\mathbf{u}^k, \delta p^k) = \lim_{t\to 0} \frac{1}{t}(\mathbf{d}(\mathbf{u}^k + t\delta\mathbf{u}^k, p^k + t\nabla\delta p^k) - \mathbf{d}(\mathbf{u}^k, p^k)).$$

Applying Taylor expansion and neglecting second order and higher terms, we get

$$\nabla \mathbf{d}(\mathbf{u}^k, p^k)(\delta\mathbf{u}^k, \delta p^k) = \begin{pmatrix} -\nu\Delta\delta\mathbf{u}^k + \mathbf{u}^k.\nabla\delta\mathbf{u}^k + \delta\mathbf{u}^k.\nabla\mathbf{u}^k + \nabla\delta p^k \\ \nabla.\delta\mathbf{u}^k \end{pmatrix}. \tag{7.37}$$

The resulting linearized Newton correction system is then given by

$$-\nu\Delta\delta\mathbf{u}^k + \mathbf{u}^k.\nabla\delta\mathbf{u}^k + \delta\mathbf{u}^k.\nabla\mathbf{u}^k + \nabla\delta p^k = \mathbf{f}_u, \tag{7.38a}$$

$$\nabla.\delta\mathbf{u}^k = \mathbf{f}_p, \tag{7.38b}$$

where the right-hand side, $\mathbf{f}_u = -\nu\Delta\mathbf{u}^k + \mathbf{u}^k.\nabla\mathbf{u}^k + \nabla p^k - \mathbf{f}$ and $\mathbf{f}_p = \nabla.\mathbf{u}^k$, is the defect calculated based on the information from the previous iteration. Following the finite element discretization procedure described previously, and using the following basis functions representation for the correction term

$$\delta\mathbf{u}_h = \sum_{j=1}^{2n_u} \boldsymbol{\Delta}\mathbf{u}_j\boldsymbol{\phi}_j, \qquad \delta p_h = \sum_{j=1}^{n_p} \boldsymbol{\Delta}\mathbf{p}_j\psi_j, \tag{7.39}$$

the linear algebraic system corresponding to the Newton correction problem turns out to be as follows:

$$\begin{bmatrix} \mathbf{A}+\mathbf{N}+\mathbf{W} & \mathbf{B}^T \\ \mathbf{B} & \mathbf{0} \end{bmatrix} \begin{bmatrix} \boldsymbol{\Delta}\mathbf{u} \\ \boldsymbol{\Delta}\mathbf{p} \end{bmatrix} = \begin{bmatrix} \mathbf{F} \\ \mathbf{g} \end{bmatrix}. \tag{7.40}$$

Using velocity splitting (7.17), above algebraic system can also be written as:

$$\begin{bmatrix} A_{11}+N_{11}+W_{11} & W_{12} & B_1^T \\ W_{21} & A_{22}+N_{22}+W_{22} & B_2^T \\ B_1 & B_2 & 0 \end{bmatrix} \begin{bmatrix} \boldsymbol{\Delta}\mathbf{u}_1 \\ \boldsymbol{\Delta}\mathbf{u}_2 \\ \boldsymbol{\Delta}\mathbf{p} \end{bmatrix} = \begin{bmatrix} \mathbf{F}_1 \\ \mathbf{F}_2 \\ \mathbf{g} \end{bmatrix}. \tag{7.41}$$

The Newton iteration is known to converge quadratically if the initial guess is close to the solution. However, if the initial guess is far from the solution, then the Newton iteration converges slowly, or it may diverge. The Newton correction system (7.38) contains a reaction term $\delta\mathbf{u}^k.\nabla\mathbf{u}^k$ that is additional to the usual Navier-Stokes system. Discretization of this reaction term involves the calculation and storage of additional velocity blocks, thus increasing the storage requirements and assembly costs of the Newton iteration. Moreover, as can be seen from equation (7.41), the block matrix $\mathbf{W}$ corresponding to the reaction term also influences the sparsity structure of the linear system. Consequently, the computational cost of each linear solver iteration also increases. Another problem with the Newton method is its lack of robustness for the large Reynolds number problems. As the Reynolds number is increased, even better initial guesses are needed to ensure the convergence of Newton iteration. To avoid these problems, the reaction term in the Jacobian is dropped which results in the fixed point scheme discussed next.

7.6.2 Fixed point linearization

Dropping out the "bad" reaction term in equation (7.38) gives the following linearized fixed point system

$$-\nu\Delta\delta\mathbf{u}^k + \mathbf{u}^k.\nabla\delta\mathbf{u}^k + \nabla\delta p^k = \mathbf{f}_u, \tag{7.42a}$$

$$\nabla.\delta\mathbf{u}^k = \mathbf{f}_p. \tag{7.42b}$$

This is equivalent to linearizing the convective term in the Navier-Stokes equation with the velocity from the previous iteration. The related linear algebraic system for the fixed point iteration reads as

$$\begin{bmatrix} A_{11}+N_{11} & 0 & B_1^T \\ 0 & A_{22}+N_{22} & B_2^T \\ B_1 & B_2 & 0 \end{bmatrix} \begin{bmatrix} \boldsymbol{\Delta}\mathbf{u}_1 \\ \boldsymbol{\Delta}\mathbf{u}_2 \\ \boldsymbol{\Delta}\mathbf{p} \end{bmatrix} = \begin{bmatrix} \mathbf{F}_1 \\ \mathbf{F}_2 \\ \mathbf{g} \end{bmatrix}. \tag{7.43}$$

The computational costs and storage requirements of the fixed point iteration are less than Newton iteration. Another advantage of the fixed point method is that it is more robust, particularly at high Reynolds numbers, and has a larger radius of convergence as compared to the Newton method. However, in contrast to the Newton method, the scheme has only linear convergence rates.

Efficient solution strategy can be built by combining both fixed point and Newton iterations, which works well particularly for higher Reynolds number flows. The strategy involves starting with some initial guess (e.g., the solution of Stokes problem), perform few fixed point iterations to reach near the final solution. Next provide this solution as an initial guess to the Newton iteration, which is expected to converge fast (hopefully quadratically) as the iteration is in the neighborhood of the final solution. An alternative approach to solving the large Reynolds number flows is to start with the low Reynolds number problem and gradually increase the Reynolds number. The low *Re* problem is solved, and the solution is used as an initial guess to Newton iteration for the next higher *Re* number. This method is called a *continuation method*.

7.6.3 MLKM as a coupled linear solver

Each outer nonlinear defect correction iteration described above involves the solution of a nonsymmetric and indefinite linear system of the type (7.41) or (7.43). Let's rewrite this linear system in the standard form as

$$\mathbf{A}\mathbf{x} \;=\; \mathbf{b}. \tag{7.44}$$

Solving such a linear system is the computationally most intensive part of the whole numerical process, and developing an efficient and robust linear solver is of utmost importance. In our monolithic multilevel Krylov subspace solver, we use MLKM solver described in chapter (4) to solve this linear system as a whole. From our experience of solving the scalar problems with the MLKM solver, we expect that our coupled MLKM solver will also converge independently of the mesh size.

As already discussed in the chapter (4), MLKM solver requires various ingredients for its functioning, such as a hierarchy of system matrices, deflation subspaces, and the preconditioner at each mesh level.

- The hierarchy of system matrices $\mathbf{A}^l$ is obtained by discretizing the infinite dimensional Navier-Stokes equations on a hierarchy of spatial meshes Ω^l ($l = 1, ..., L$ with Ω^1 the coarsest mesh and Ω^L the finest mesh where we seek the solution to our problem), using the Galerkin finite element method. The right-hand side $\mathbf{b}^L$ at the finest level is specified while the right-hand side at all other levels is generated during the MLKM run. The outer nonlinear iteration works on the finest level, and the inner linear iteration solves at each mesh level the linear system of the form

$$\mathbf{A}^l\mathbf{x}^l = \mathbf{b}^l. \tag{7.45}$$

- Standard multigrid prolongation and restriction inter-grid transfer operators are taken as deflation subspaces $\mathbf{Z}$ and $\mathbf{Y}^T$, respectively. These grid transfer operators have already been discussed in section (3.2.3) for the Q_1 element. The elementwise prolongation for the nonconforming rotated bilinear finite element functions and piecewise constant finite element functions is shown schematically in figure (7.2). On the coarse mesh are shown the weights of the corresponding d.o.f's that are used to evaluate the d.o.f on the fine mesh. In the top row, full prolongation in $\tilde{Q}_1$ element (with mean values on edges as d.o.f's) is shown only for two nodes; remaining nodes are calculated analogously. For nodes on edges, take the average of the two adjacent macro-elements. For the piecewise constant Q_0 element (bottom row), the constant value of the function at the coarse cell is transferred to all the children cells in the fine mesh. See [132] for more details. The restriction operator is obtained as an adjoint of the prolongation operator. See [70, 77] for the Q_2/P_1^{disc} element.

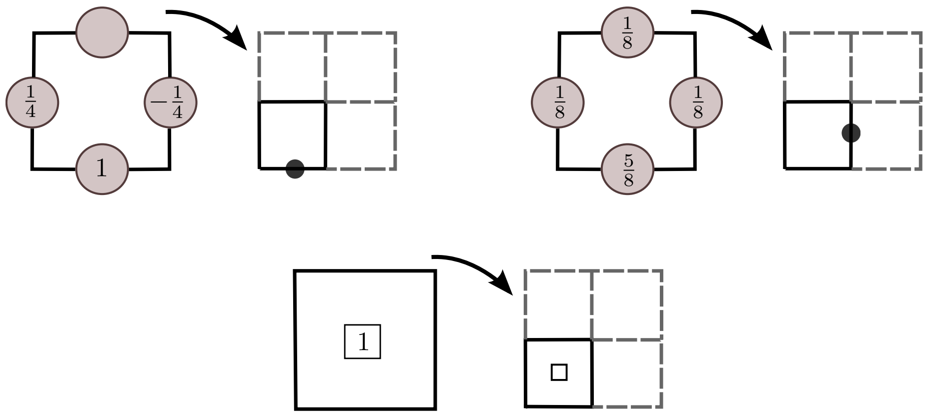


Figure 7.2: Schematic representation of prolongation in rotated bilinear $\tilde{Q}_1$ element (with mean values on edges as DOFs) using full interpolation [top], and in Q_0 element using piecewise constant interpolation [bottom].

Local pressure Schur complement(LPSC) preconditioner

Our experience of solving the scalar problems with MLKM method in the previous chapters tells us that preconditioning is vital for the performance of MLKM method. It would be convenient from the practical viewpoint, if the basic iterative schemes used as preconditioners for the scalar problems, could also be used as preconditioners for the solution of Navier-Stokes equations. The zero diagonal block in the saddle point problem unfortunately makes it impossible to use the point Jacobi and point Gauss-Seidel as preconditioners to the MLKM method. Similarly, direct application of ILU(0) preconditioner may also fail due to the zeros on the diagonal. Although various numerical strategies (such as using special reorderings and /or pivoting) have been proposed by the scientific community to avoid ILU breakdowns, still incomplete factorization failure rates are high for the nonsymmetric and indefinite matrices arising from the discretization of Navier-Stokes equations. In coupled MLKM solver, we use *local pressure Schur complement* preconditioner at each mesh level. The idea was originally introduced by Vanka [138] to solve the Navier-Stokes equations discretized with finite difference method on staggered grids.

The main idea of the LPSC approach is to divide the domain in small patches Ω_i, solve the local subsystem on each patch exactly by treating all the variables (velocity and pressure) associated with the patch monolithically and update the local degrees of freedom in a blockwise Jacobi or Gauss-Seidel manner. The patch may constitute one cell or many neighboring cells. In case of one cell, the preconditioner is called *cell-oriented LPSC* preconditioner. In this thesis, we have used the cell-oriented LPSC preconditioner embedded in a block Gauss-Seidel iteration.

To better illustrate the functioning of the LPSC preconditioner, following [82] we introduce few terminologies. Let $I(K)$ be the index set containing the list of all degrees of freedom associated with the element K. Through this index set, let us define the rectangular matrix $\mathbf{A}_K$ containing only those rows from the global matrix $\mathbf{A}$ that correspond to the index set $I(K)$. Similarly, $\mathbf{x}_K$ and $\mathbf{b}_K$ are the subvectors of $\mathbf{x}$ and $\mathbf{b}$ respectively, restricted to the element K. Furthermore, we define the square matrix $\mathbf{A}_{K,K}$ by dropping out all the columns from the matrix $\mathbf{A}_K$ that do not correspond to the index set $I(K)$. Using these notations, the LPSC preconditioner can be realized by applying the defect correction elementwise as

$$\mathbf{x}_K^{j+1} = \mathbf{x}_K^j - \overline{\omega}^j \mathbf{C}_K^{-1}(\mathbf{A}_K \mathbf{x}^j - \mathbf{b}_K), \tag{7.46}$$

where $\overline{\omega} > 0$ is a damping parameter, and $\mathbf{C}_K$ is an appropriate preconditioner yet to be defined. In practice, above iteration is carried out in three steps:

1. Calculate the defect for the element K

$$def\mathbf{x}_K = (\mathbf{A}_K \mathbf{x}^j - \mathbf{b}_K). \tag{7.47}$$

2. Solve the local system for $\mathbf{y}_K$

$$\mathbf{C}_K \mathbf{y}_K = def\mathbf{x}_K. \tag{7.48}$$

   Depending on the choice of the matrix $\mathbf{C}_K$, two types of the LPSC preconditioners are defined

   (a) *Diagonal LPSC preconditioner*

$$\mathbf{C}_K := diag(\mathbf{A}_{K,K}).$$

(b) *Full LPSC preconditioner*

$$\mathbf{C}_K := \mathbf{A}_{K,K}.$$

The diagonal version is cheaper and faster, however, the full version is more stable and robust. In both the cases, the resulting local problem is of saddle point type and small; in 2D case, the system has 9 unknowns for the $\tilde{Q}_1/Q_0$ space and 21 unknowns for the Q_2/P_1^{disc} space. These systems can be feasibly solved either directly (e.g., with the LAPACK package [3]) or using the Schur complement decomposition [132].

3. Find the new iterate using the damping factor $\overline{\omega}$

$$\mathbf{x}_K^{j+1} = \mathbf{x}_K^j - \overline{\omega}^j \mathbf{y}_K.$$

The calculated local degrees of freedom are updated successively in a Gauss-Seidel manner, i.e., as soon as the local degrees of freedom are calculated in the current patch, their new updated values are used for the calculations in the next patch.

7.6.4 Adaptive step length control

After the solution of the linear system in nonlinear defect correction iteration (line 5 in algorithm 7.1), the next step is to choose the appropriate damping factor $\tilde{\omega}^k$ to ensure the stabilization and acceleration of the numerical solver. This optimal damping parameter is determined such that the error between the new iterate $\mathbf{x}^{k+1}$ and the exact solution is minimized in the Euclidean norm. This means that $\tilde{\omega}^k$ should satisfy

$$\tilde{\omega}^k = \min_{\omega} \|T(\mathbf{x}^k + \omega\delta\mathbf{x}^k)(\mathbf{x}^k + \omega\delta\mathbf{x}^k) - \mathbf{f}\|_E. \tag{7.49}$$

Above equation is a one-dimensional nonlinear optimization problem in variable ω; to solve it we linearize the fully nonlinear operator $T(\mathbf{x}^k + \omega\delta\mathbf{x}^k)$ using $\tilde{\omega}^{k-1}$ from the previous iteration and solve the resulting linear minimization problem as follows:

1. On the finest level, build the global matrix $\overline{T}$ at point $\mathbf{x}^k + \tilde{\omega}^{k-1}\delta\mathbf{x}^k$, i.e.,

$$\overline{T}(\mathbf{x}^k + \tilde{\omega}^{k-1}\delta\mathbf{x}^k) = \begin{pmatrix} \mathbf{S}^k & \mathbf{B}^T \\ \mathbf{B} & 0 \end{pmatrix},$$

where $\mathbf{S}^k = \mathbf{A} + N(\mathbf{u}^k + \tilde{\omega}^{k-1}\delta\mathbf{u}^k)$.

2. Calculate *optimal* damping parameter $\tilde{\omega}^k$ as follows:

$$\tilde{\omega}^k = \left\langle \frac{\overline{T}\delta\mathbf{x}^k, \mathbf{f} - \overline{T}\mathbf{x}^k}{\overline{T}\delta\mathbf{x}^k, \overline{T}\delta\mathbf{x}^k} \right\rangle_E,$$

where $\langle .,. \rangle_E$ is the Euclidean scalar product.

3. Calculate the new iterate using equation (7.35)

$$\mathbf{x}^{k+1} = \mathbf{x}^k + \tilde{\omega}^k\delta\mathbf{x}^k.$$

This completes the description of the main components of the coupled MLKM solver. Of course, other several possible ingredients can be added to the solver, however, they have been omitted for the sake of simplicity. For example, a proper stabilization is crucial for the Navier-Stokes equations with higher Reynolds numbers, since pure Galerkin FEM discretization, in this case, would result in numerical oscillations thus leading to wrong solutions and convergence problems for the numerical solver. Edge oriented jump stabilization, introduced in the chapter (5) and used for the scalar convection dominated flows, will be used for the Navier-Stokes problems involving higher Reynolds numbers. In the next chapter, we present the numerical analysis of the proposed coupled MLKM solver and study its efficiency and robustness for solving the Navier-Stokes equations.

8 Numerical results for stationary incompressible Navier-Stokes problem

Chapters 5 and 6 have presented the numerical description and analysis of the MLKM solver for the scalar model problems. These scalar problems pose challenges for the numerical solvers that also arise in the case of more complicated Navier-Stokes equations. For instance, the numerical instabilities observed in a convection-diffusion problem with high Peclet numbers, also exist in Navier-Stokes equations with high Reynolds numbers. Similarly, the convergence issues faced by numerical solvers while solving scalar problems on anisotropic meshes, are also exhibited when Navier-Stokes equations are solved on such meshes. In this chapter, we numerically analyze the coupled multilevel Krylov subspace solver, presented in the previous chapter, for the solution of Navier-Stokes equations, and see how robustly it can handle these numerical challenges and compare its performance with other coupled solvers in FEATFLOW software.

We have seen from the scalar elliptic problems that in hard numerical setups (highly convective or highly anisotropic situations), MLKM solver performed better than multigrid solver when Jacobi or Gauss-Seidel are used as preconditioner/ smoother. Whereas, multigrid with ILU(0) smoothing has shown better convergence rates than MLKM/ILU(0) solver in some cases. However, the incomplete factorization is known to have high failure rates when applied to the indefinite systems arising from the discretization of Navier-Stokes equations [120]. The presence of zeros on the diagonal of these indefinite systems makes the incomplete factorization nonexistence in many cases. Even if the incomplete LU factorization does exist, the triangular solves are quite frequently highly unstable or too inaccurate to produce satisfactory convergence rates. The LPSC preconditioner, which always exists for such indefinite systems, can be considered as a local block Jacobi or block Gauss-Seidel preconditioner. Based on our experience from the scalar problems, we expect that the coupled MLKM solver preconditioned with LPSC preconditioner will be more robust for the solution of Navier-Stokes equations as compared to the coupled multigrid solver with LPSC as smoother.

Following the precedent set by the scalar test problem cases, we first validate our monolithic coupled MLKM solver by solving the Navier-Stokes equations on standard benchmark problems. The solver is tested using both conforming and nonconforming finite elements, and the numerical results are compared with the already published results in the literature. Next, we perform numerical experiments to see which parameters of our linear solver influence the convergence behavior of the monolithic coupled MLKM solver and choose optimal values for these parameters.

Finally, the benchmark problems are solved for a range of Reynolds numbers on the meshes with increasing aspect ratios, to analyze the robustness of the proposed solver numerically. The solver robustness is checked and compared with other solvers like multigrid and UMFPACK, with respect to the following two key points:

- How does the solver behave, in comparison with other solvers, in view of the increasing nonlinearity (occurring due to the increasing Reynolds number) in the problem?
- How does the increase in anisotropy in the spatial mesh influences the solver convergence behavior in comparison with the other methods?

We also compare the time taken by the solvers to reach the stopping criterion, as an indication of the overall efficiency comparison of the solvers. These tests are performed for both conforming and nonconforming finite elements and using the fixed point as well as Newton nonlinear iterations.

8.1 Numerical Validation

This section is dedicated to the validation of the code, which is a crucial step in the development of new numerical methods. To validate our code, we have chosen two well-known benchmark problems from two different classes of flow problems, namely *enclosed flows* and *inflow/outflow type flows*. Enclosed flows have $\mathbf{u}.\mathbf{n} = 0$ everywhere on the boundary Γ, i.e., the flow does not cross the boundaries of the domain. The nontrivial flow is generated by imposing a nonzero tangential velocity at some part of the boundary. On the other hand, inflow/outflow type flows are characterized by specifying $\int_\Gamma \mathbf{u}.\mathbf{n} \neq 0$.

8.1.1 Lid-driven cavity flow benchmark

Lid-driven cavity (LDC) flow is a well known enclosed flow type benchmark problem, used by the scientific community to test and validate new numerical techniques. Highly accurate numerical results are available in literature [65, 55, 26, 135, 37] for moderately high Reynolds numbers for the benchmark comparisons. Despite its geometric simplicity, the physics of the driven cavity fluid flow involves recirculations, counter-rotating vortices formation, and point singularities, which pose enormous challenges for numerical methods. A standard setup of the problem involves producing a shear flow in the cavity by moving the upper wall at a constant speed. This flow configuration is akin to many industrial applications such as coater for producing high-grade paper and photographic films [1].

The problem domain in our numerical experiments consists of a unit square $\Omega = (0,1)^2$, with the Dirichlet boundary conditions prescribed on the whole boundary as follows: the top wall or lid is moved from left to right with a constant velocity $u_x = 1$, and a no-slip boundary condition is set on the side and bottom walls of the cavity. The coarse grid consists of 16 quadrilaterals obtained by the uniform refinement of a unit square. Table 8.1 shows the problem size at various levels for both $\tilde{Q}_1/Q_0$ and Q_2/P_1^{disc} finite elements. The simulations are

		DOF	
Level	Cells	$\tilde{Q}_1/Q_0$	Q_2/P_1^{disc}
4	1024	5248	11522
5	4096	20736	45570
6	16384	82432	181250
7	65536	328704	722946
8	262144	1312768	2887682
9	1048576	5246976	11542530

Table 8.1: **LDC problem:** Problem size at various mesh levels.

run for the constant viscosities of $\nu = 10^{-3}$ and $\nu = 2\times10^{-4}$ which correspond to $Re = 1000$ and $Re = 5000$ respectively. Streamline contours for the simulations on mesh level 8 are presented in Figure 8.1. These streamline patterns exhibit the physics of the flow inside the cavity, with the large primary vortex in the middle and small counter-rotating secondary and tertiary vortices developing at the corners as the inertia of the flow increases. At Reynolds number 1000, two secondary vortices are visible at the two bottom corners of the cavity. Moreover, if we look carefully at the contours for $Re = 1000$, the tertiary vortices are also visible at the bottom corners, as also reported by Kumar et al. [123] at fine grid resolutions. At $Re = 5000$, the primary vortex moves more towards the geometric center, and the smaller vortices in the bottom corners grow in size due to the increased inertia in the flow. It also exhibits a new secondary vortex near the upper left corner.

Benchmark Numerical Quantities

As a quantitative measure for the comparison of our numerical results, we calculate the total kinetic energy at each mesh level as

$$E = \frac{1}{2}\|\mathbf{u}\|_{0,\Omega}^2, \tag{8.1}$$

and compare our results with the other published results in the literature for both nonconforming $\tilde{Q}_1/Q_0$ and conforming Q_2/P_1^{disc} finite elements. These kinetic energy comparisons are presented in Tables 8.2 and 8.3, and as can be seen, our solver results are consistent with the published results. The results also reveal the fact that for highly nonlinear flows ($Re = 5000$), higher order finite elements are necessary for obtaining the mesh converged solutions on reasonably refined meshes. Figure 8.2 also shows that with higher order Q_2/P_1^{disc} finite elements, the grid independent velocity profile is obtained on mesh refinement

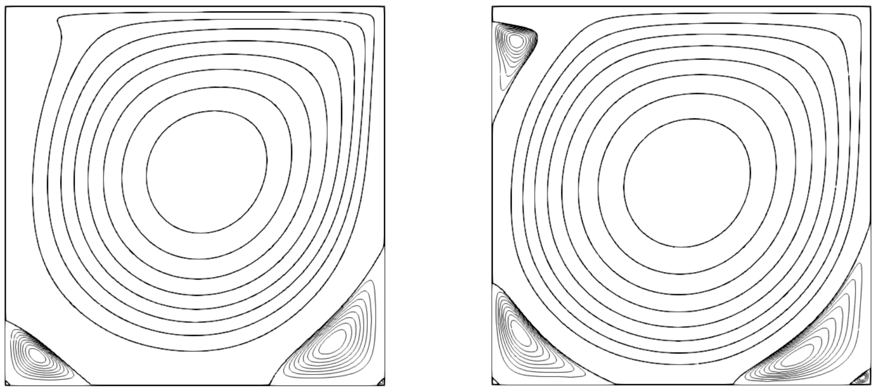


Figure 8.1: **LDC problem:** Streamlines for $RE = 1000$ (left) and $RE = 5000$ (right).

level as low as level 5, whereas, for the lower order $\tilde{Q}_1/Q_0$ finite elements the mesh convergence is not achieved even at the mesh refinement level 9.

	Re=1000		Re=5000	
cells	Present	Ref. [135]	Present	Ref. [135]
4	4.2555E-02	4.2554E-02	4.7442E-02	4.7442E-02
5	4.2928E-02	4.2927E-02	4.5427E-02	4.5427E-02
6	4.3545E-02	4.3545E-02	4.3942E-02	4.3942E-02
7	4.4091E-02	4.4090E-02	4.4621E-02	4.4622E-02
8	4.4368E-02	4.4367E-02	4.5896E-02	4.5896E-02
9	4.4473E-02	4.4472E-02	4.6772E-02	4.6771E-02

Table 8.2: **LDC problem:** Comparison of kinetic energy with other published results for $\tilde{Q}_1/Q_0$ finite element

Finally in addition to comparing the global quantity (total kinetic energy), we also compare the local quantities (velocity component profiles) along the cutlines passing through the geometric center of the cavity. $u - velocity$ profile along the vertical line and $v - velocity$ profile along the horizontal line passing through the geometric center of the cavity are plotted in figures 8.3 and 8.4, respectively. Here also these profiles are in excellent agreement with that of the Ghia et al.[65] and Erturk et al.[55].

	Re=1000		Re=5000	
cells	Present	Ref. [37]	Present	Ref. [37]
4	4.5395E-02	4.8095E-02	5.9387E-02	6.1149E-02
5	4.4605E-02	4.4590E-02	4.9229E-02	4.9571E-02
6	4.4525E-02	4.4524E-02	4.7684E-02	4.7691E-02
7	4.4519E-02	4.4519E-02	4.7446E-02	4.7465E-02
8	4.4518E-02	4.4518E-02	4.7429E-02	4.7430E-02

Table 8.3: **LDC problem:** Comparison of kinetic energy with other published results for Q_2/P_1^{disc} finite element

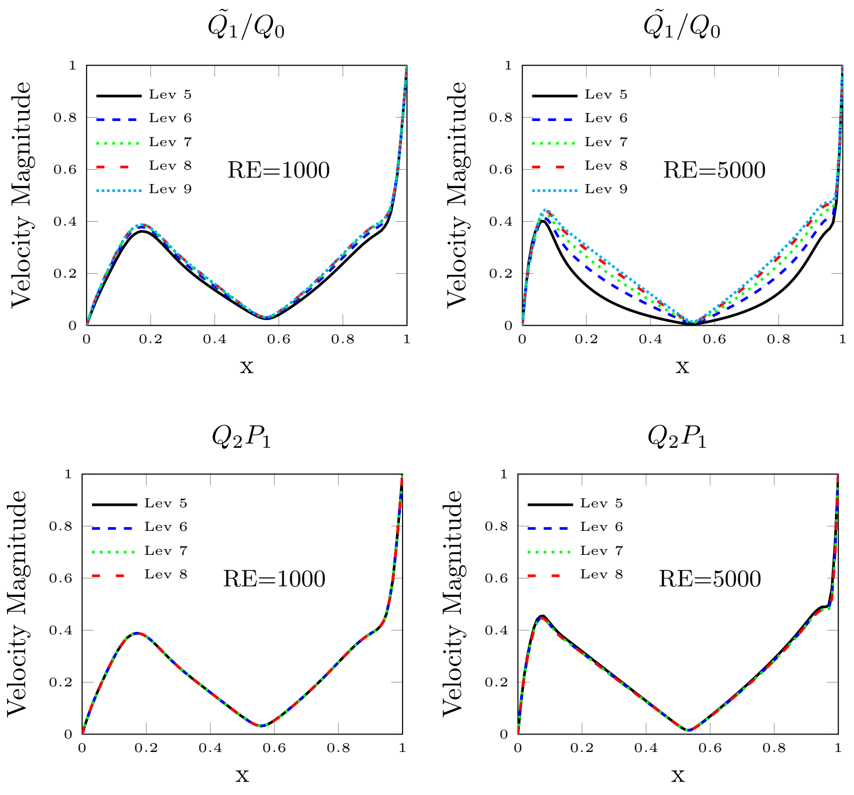


Figure 8.2: **LDC problem:** Velocity magnitude profiles at a horizontal line passing through the geometric center.

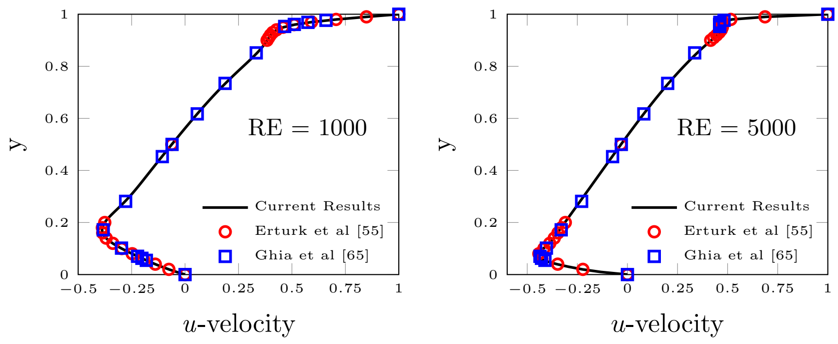


Figure 8.3: **LDC problem:** u-velocity profiles computed at a vertical line passing through the geometric center

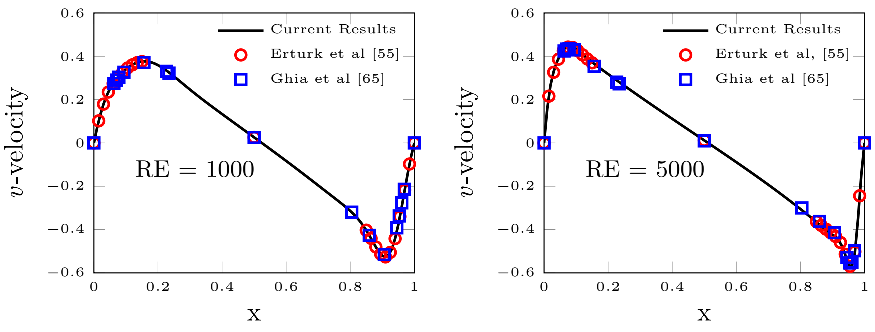


Figure 8.4: **LDC problem:** v-velocity profile computed at a horizontal line passing through the geometric center

8.1.2 Flow around cylinder benchmark

The flow around a cylinder (FAC) is a benchmark based on the *1995 DFG Priority Research Programme:* "Flow Simulation on High Performance Computers" [124, 132]. The problem consists of simulating and analyzing the flow behavior of an incompressible Newtonian fluid past the circular obstacle placed at right angle to the oncoming fluid in a rectangular channel. The Kinematic viscosity of the fluid is taken as $\nu = 10^{-3}\ m^2/s$ and the density of the fluid as $\rho = 1.0\ kg/m^3$. The problem domain consists of the rectangular channel of height $H = 0.41\ m$ and length $L = 2.2\ m$, with a circular cylinder of radius $r = 0.05\ m$ placed at the position (0.2,0.2), as shown in figure 8.5. The domain

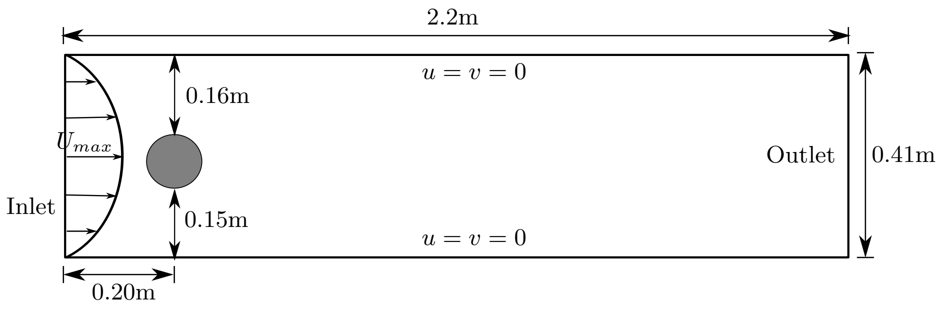


Figure 8.5: Geometry of flow around a cylinder problem

is subject to the following boundary conditions:

- No slip boundary condition ($u = v = 0$) is imposed on the upper wall (at $y = 0.41\ m$), lower wall (at $y = 0.0\ m$) and on the boundary S of the circular cylinder.
- Parabolic inflow boundary profile is imposed on the left wall as follows:

$$u(0, y) = 4U_{max}\ y(H - y)/H^2, \qquad v(0, y) = 0$$

- The right wall is an outflow boundary and is subject to the "do nothing" natural boundary condition

$$\nu \frac{\partial \mathbf{u}}{\partial n} - pn = 0$$

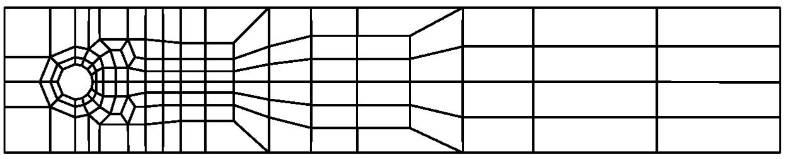
If the maximum velocity is taken as $U_{max} = 0.3m/s$, then the mean velocity of the parabolic profile, which is also the characteristic velocity of the flow, comes out to be

$$U_{mean} = \frac{2}{3} U_{max} = 0.2.$$

The characteristic length L here is the diameter of the obstacle normal to the flow direction, i.e., $L = 2 \times 0.05 = 0.1m$. The resulting Reynolds number of the flow is given by

$$Re = \frac{U_{mean} L}{\nu} = \frac{0.2 \times 0.1}{10^{-3}} = 20,$$

which corresponds to the laminar and stationary flow past the cylinder. The domain is discretized using quadrilateral finite elements, and the hierarchy of fine meshes are obtained by the uniform mesh refinements, i.e., by joining the midpoints of the opposite sides of each cell of the immediate coarse level mesh. Figure 8.6 shows coarse mesh at level 1, and the number of cells and degrees of freedom at various mesh levels for both $\tilde{Q}_1/Q_0$ and Q_2/P_1^{disc} finite elements.



		Degrees of freedom	
Level	Cells	$\tilde{Q}_1/Q_0$	Q_2/P_1^{disc}
1	130	702	1534
2	520	2704	5928
3	2080	10608	23296
4	8320	42016	92352
5	33280	167232	367744
6	133120	667264	1467648
7	532480	2665728	5863936
8	2129920	10656256	23442432

Figure 8.6: **FAC problem:** Coarse mesh (top); Problem size at various mesh levels (bottom).

Benchmark Numerical Quantities

We calculate drag and lift coefficients at the surface S of the cylinder as benchmark quantities, which are important quantities of interest in many real life engineering applications. For instance, calculation of such quantities is critical for the design of aircraft, automobiles, high-rise buildings and offshore structures, etc. Drag and lift coefficients are the dimensionless quantities defined as

$$C_D = \frac{2}{U_{mean}^2 L}\int_S (\rho\nu\frac{\partial \mathbf{u}_t}{\partial \mathbf{n}}n_y - pn_x)dS, \qquad D_L = \frac{2}{U_{mean}^2 L}\int_S (\rho\nu\frac{\partial \mathbf{u}_t}{\partial \mathbf{n}}n_x + pn_y)dS, \tag{8.2}$$

where $\mathbf{u}_t$ is the tangential velocity and $\mathbf{n} = (n_x, n_y)^T$ is the unit normal on S. Nabh [107] has computed highly accurate results for C_D and C_L using high order spectral methods, which are taken as reference results:

- Drag coefficient: $C_D = 5.57953523384$
- Lift coefficient: $C_L = 0.010618948146$

The problem is solved with coupled MLKM solver on various mesh refinement levels for both $\tilde{Q}_1/Q_0$ and Q_2/P_1^{disc} finite elements. For $\tilde{Q}_1/Q_0$ discretization, edge oriented jump stabilization (stabilization parameter $\gamma = 0.01$) is used, while higher order Q_2/P_1^{disc} finite element discretization solves the problem without any stabilization. Figure 8.7 shows the velocity, pressure and stream function contours for the solution at level 6 using Q_2/P_1^{disc} finite elements. Drag and lift coefficient values at various mesh levels are compared with the published results in the literature, see Tables 8.4 and 8.5. As can be seen, our results at each level are in good agreement with the published results for both the finite element discretizations used. Also, for both the discretizations, the solution converges to the reference solution with the increase in the mesh refinements.

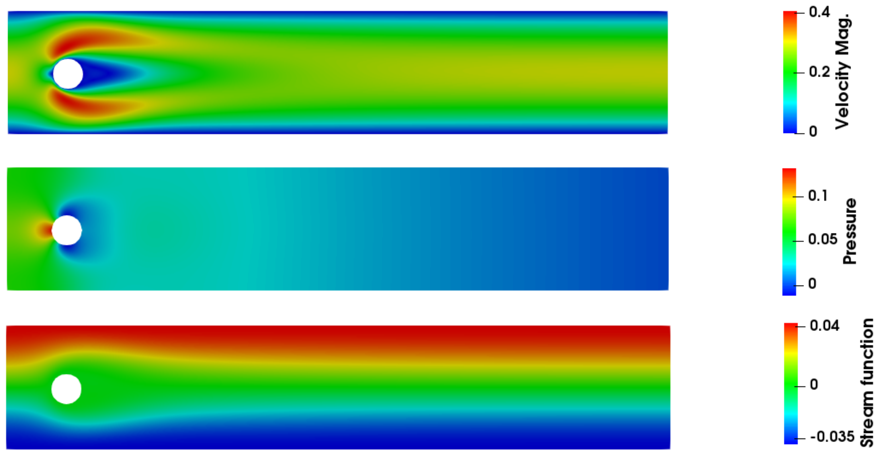


Figure 8.7: **FAC problem:** Velocity, pressure and stream function contours for stationary flow at $Re = 20$.

	Drag/ Lift	
Lev.	Present	Ref.[109]
3	5.6102/ 0.008590	5.6186/ 0.008874
4	5.5935/ 0.009619	5.5901/ 0.010022
5	5.5841/ 0.010271	5.5823/ 0.010436
6	5.5808/ 0.010508	5.5803/ 0.010566
7	5.5799/ 0.010585	-
8	5.5796/ 0.010608	-

Table 8.4: **FAC problem:** Results for $\tilde{Q}_1/Q_0$ finite elements.

	Drag/ Lift	
Lev.	Present	Ref.[37]
2	5.5456/ 0.00904	5.5424/ 0.00945
3	5.5671/ 0.01043	5.5672/ 0.01047
4	5.5761/ 0.01057	5.5761/ 0.01057
5	5.5786/ 0.01059	5.5786/ 0.01060
6	5.5793/ 0.01061	-
7	5.5795/ 0.01062	-

Table 8.5: **FAC problem:** Results for Q_2/P_1^{disc} finite elements.

8.2 Influence of solver parameters on the performance of coupled MLKM solver.

Coupled MLKM solver presented in the previous chapter involves some solver parameters. In this section, we study the effect of these parameters on the convergence behavior of the solver. Numerical experiments are performed by fixing all the parameters of the solver except one parameter, and then simulations are run for a range of values of this parameter. The purpose of these experiments is two-fold: first to see how sensitive our coupled MLKM solver performance is to the change in values of a certain parameter, and second to choose the optimal value of this parameter (optimal in a sense that the overall cost of the coupled MLKM solver is minimized). As a test problem for these experiments, we have considered a lid-driven cavity flow problem with $Re = 1000$, discretized using $\tilde{Q}_1/Q_0$ finite elements. Coupled MLKM solver with a damped full Vanka or full LPSC preconditioner ($\bar{\omega} = 0.7$) is used to solve this test problem. In all the numerical results presented in this section, the absolute stopping criterion for the outer nonlinear iteration is set to $\epsilon_{NL} = 10^{-10}$, where ϵ_{NL} is the norm of the nonlinear defect vector. Moreover, the shift scale parameter ω of the MLKM solver is set equal to 1.0.

8.2.1 Number of FGMRES iterations at each level

As mentioned previously in chapter 4, one iteration of MLKM solver is configured with three numeric letters in the brackets. If l_{max} is the finest mesh refinement level on which the problem is solved and l_{min} is the minimum mesh refinement level, then MLKM(x, y, z) means one iteration of MLKM solver involves:

- x number of FGMRES iterations at level $(l_{max} - 1)$.
- y number of FGMRES iterations at all levels between $(l_{max} - 1)$ and l_{min}.
- z number of FGMRES iterations at the coarsest level l_{min}.

The test problem is solved using various FGMRES iterations configurations of the MLKM solver. Fixed point is used as a nonlinear iteration, and the linear solver is set to gain one digit at each outer nonlinear iteration. Approximate maximum eigenvalue in MLKM solver is taken as $\lambda_{max} = 1.0$. Figure 8.8 shows the convergence behavior of the solver for various FGMRES iterations configurations at the mesh refinement level 7.

The top-left graph in the Figure 8.8 with title MLKM(x,2,2) shows the effect of changing the number of FGMRES iterations at level $(l_{max} - 1)$ while keeping the number of FGMRES iterations at all the other coarse levels fixed. As can be seen, for the MLKM(2,2,2) configuration, the convergence of solver is slightly deteriorated, indicating that with only two FGMRES iterations at level $(l_{max} - 1)$, the related Galerkin system solve is not accurate enough. Doubling these iterations improves the convergence rate of the solver, however, increasing the iterations beyond four at this level produces no more improvement in the convergence rates. The other two graphs in this figure (top-right and bottom) show that convergence of solver is not affected by the accuracy of the solutions at the rest of the coarse levels. Although the convergence behavior of MLKM(4,2,2) and MLKM(4,2,5) is almost identical (see also Table 8.6), we choose MLKM(4,2,5) configuration as our optimal choice because of its slightly improved convergence at fine meshes and almost similar computational costs as that of MLKM(4,2,2).

This convergence behavior of coupled MLKM is very much similar to the one reported by Erlangga and Nabben in [54] for the scalar Poisson problem. However, in their results, the grid levels are fixed to five, with the minimum level increasing with the increase in the finest mesh level. In our case, the coarsest mesh level is always fixed. Moreover, they solve the Galerkin problem at the coarsest grid exactly, whereas, in our case, this system is solved with few FGMRES iterations.

	MLKM(4,2,2)		MLKM(4,2,5)	
Lev.	NL/MLKM$_{\text{total}}$	Time	NL/MLKM$_{\text{total}}$	Time
5	27/109	5.9	26/105	5.8
6	20/79	17.9	20/79	18
7	17/67	69.5	16/63	65.9

Table 8.6: Performance comparison of two MLKM configurations for solving LDC problem at $Re = 1000$

8.2.2 Approximate maximum eigenvalue

In this subsection, we solve the test problem using various values of approximate maximum eigenvalue (λ_{max}) to see the effect of λ_{max} on the convergence behavior of the solver. The following MLKM configuration is used for these numerical experiments: MLKM(4,2,5), fixed point as a nonlinear iteration, and $\epsilon_{lin=10^{-1}}$ as a stopping criterion for the linear iteration. Table 8.7 depicts the

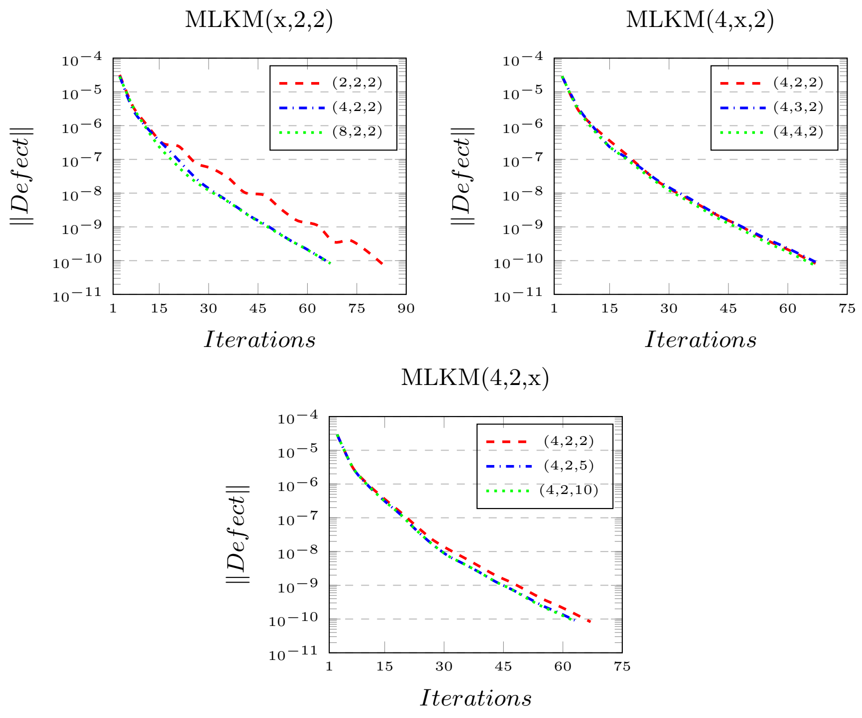


Figure 8.8: Convergence behavior of the solver for various MLKM configurations to solve LDC problem ($Re = 1000$) at level 7. Total linear solver iterations on the horizontal axis and nonlinear defect with logarithmic scale on vertical axis.

convergence behavior of the solver by presenting the number of iterations required for convergence for different values of λ_{max}. Here NL represents the total number of nonlinear iterations and MLKM$_{\text{avg.}}$ represents the average linear iterations taken per nonlinear iteration. The results show that the solver is not very sensitive to the value of λ_{max} and mesh size independent convergence is achieved for a range of values of λ_{max} $(0.8 - 1.4)$. We take $\lambda_{max} = 1.0$ as our optimal choice of the approximated maximum eigenvalue.

8.2.3 Linear solver stopping criterion

Next, we study the effect of the linear solver stopping criterion(ϵ_{lin}) on the convergence properties and efficiency of the fixed point and Newton nonlinear iterations. To this end, we employ MLKM(4,2,5) with $\lambda_{max} = 1.0$ as a linear solver and solve the test problem on mesh level 7 for different values of the ϵ_{lin}. In figure 8.9, the norm of the defect is plotted at each nonlinear iteration to see the convergence behavior of the nonlinear solvers. With the fixed point nonlinear iteration (left), all the defect reduction curves for the coupled MLKM solver are almost identical, which means that the global convergence of the solver in this case is not influenced by the linear solver stopping criterion. Moreover, as expected the defect reduces linearly for the fixed point iteration. The conver-

	Iterations (NL/MLKM$_{\text{avg.}}$)				
Lev./λ_{max}	**0.6**	**0.8**	**1.0**	**1.2**	**1.4**
5	27/13	26/5	26/4	27/4	27/5
6	21/20	21/4	20/4	22/4	23/4
7	18/13	18/4	16/4	17/4	18/4

Table 8.7: Effect of approximate maximum eigenvalue on the convergence behavior of the coupled MLKM solver for LDC problem at $Re = 1000$.

gence of the Newton iteration (right) is quadratic for the case $\epsilon_{lin} = 10^{-10}$, however, it becomes linear for $\epsilon_{lin} = 10^{-1}$. For $\epsilon_{lin} = 10^{-2}$, the convergence is quadratic in the beginning, and becomes linear in the last two iterations, since the residual becomes very small in the last two iterations and reaches the region of linear convergence.

In Table 8.8, the number of iterations and the time (in seconds) taken by the solvers to reach the convergence are compared for the above numerical experiments. Here NL means the total number of nonlinear iterations and MLKM$_{\text{Total}}$ the total number of linear iterations. The table reveals the fact that for fixed point iteration, gaining only one digit in linear iteration is computationally more economic than gaining two or more digits. Hence, for fixed point iteration, we take $\epsilon_{lin} = 10^{-1}$ as our optimal choice of linear solver stopping criterion. In case of Newton solver, although $\epsilon_{lin} = 10^{-10}$ produces quadratic convergence, however, linear solver requires much more iterations to satisfy the stopping criterion, and as a consequence, the CPU times are very high. We take $\epsilon_{lin} = 10^{-2}$ as our optimal choice due to the fast convergence of the linear solver in this case. Although $\epsilon_{lin} = 10^{-1}$ results in minimum linear solver iterations for this problem, for higher nonlinear flows the linear solver with such weak stopping criterion will not result in minimum iterations, or it may even lead to the divergence of nonlinear Newton solver.

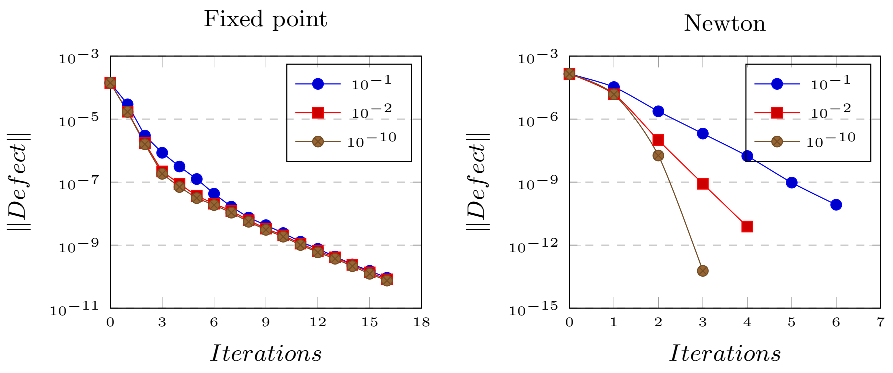


Figure 8.9: Convergence behavior of coupled MLKM solver for different linear solver stopping criteria. Standard fixed point iteration(left) and Newton iteration (right).

	Fixed Point			Newton		
ϵ_{lin}	NL	$\text{MLKM}_{\text{Total}}$	Time	NL	$\text{MLKM}_{\text{Total}}$	Time
10^{-1}	16	63	66	6	25	51
10^{-2}	16	120	89	4	31	51
10^{-10}	16	554	266	3	104	128

Table 8.8: Number of nonlinear iterations and CPU times (in seconds) of coupled MLKM solver using fixed point and Newton nonlinear iteration, for various linear solver stopping criteria.

We conclude this section by presenting the "optimal" configuration of the coupled MLKM as follows:

- MLKM(4,2,5)
- Shift scale parameter, $\omega = 1.0$.
- Approximate maximum eigenvalues $\lambda_{max} = 1.0$.
- $\epsilon_{lin} = 10^{-1}$ (for fixed point iteration), $\epsilon_{lin} = 10^{-2}$ (for Newton iteration).

This configuration of coupled MLKM solver is used in the numerical experiments presented in the later sections of this chapter unless otherwise mentioned.

8.3 Numerical comparison of monolithic MLKM solver with other monolithic solvers

In this section, we present the numerical comparison of our monolithic MLKM solver with other monolithic solvers available in FEATFLOW software, namely multigrid and UMFPACK solvers. For these comparative studies, once again we choose two benchmark problems, one from the enclosed flow category (lid-driven cavity flow problem) and other from the inflow/ outflow category (flow around a square obstacle). We make these problems numerically harder and more challenging by solving them for a range of Reynolds numbers on spatial meshes with increasingly higher aspect ratios. We compare the robustness of the solvers by looking at their convergence behavior against:

- The increase in the nonlinearities in the flow.
- The increase in the anisotropies of the spatial mesh.

We show the number of iterations to convergence (total nonlinear iterations/ average linear iterations per nonlinear iteration) as an indicator of the convergence behavior of the solvers. We also present the total CPU time elapsed as an indicator of the overall efficiency of these solvers. This total CPU time includes all the times taken during pre-processing, matrix generation, solution phase and the post-processing phase. All tests for such CPU time measurement and comparison are performed on Intel Xeon E5-26700 processor with 2.6 GHz frequency.

The solvers are required to satisfy the following stopping criteria on the nonlinear and linear iterations:
Nonlinear solver stopping criteria:

- Absolute residual for the momentum equation $= 10^{-8}$
- Absolute residual for the continuity equation $= 10^{-8}$
- Maximum relative changes for the velocity $\mathbf{u} = 10^{-5}$
- Maximum relative changes for the pressure $p = 10^{-3}$

Linear solver stopping criteria:

- The relative change in the residual $\epsilon_{lin} = 10^{-1}$ (for fixed point iteration) and $\epsilon_{lin} = 10^{-2}$ (for Newton iteration).

8.3.1 Numerical results for lid-driven cavity flow

Lid-driven cavity flow problem as described in section (8.1.1) is solved on the anisotropic meshes. The coarsest level mesh used by the multilevel solvers is shown in figure 8.10. Anisotropy in the mesh is increased by moving the horizontal and vertical lines neighboring the boundary, further closer to the boundary. Mesh at the next refinement level is achieved by joining the midpoints of the opposite sides of each cell in the coarse mesh. The domain is discretized using the nonparametric finite elements quadrilaterals, once using the nonconforming $\tilde{Q}_1/Q_0$ finite elements and once using the conforming Q_2/P_1^{disc} finite elements. Edge-oriented jump stabilization is used to stabilize less smooth problems. MLKM solver with full Vanka preconditioner and *optimal* configuration is used to solve the problem. Multigrid with F-cycle, 4 pre- and post-smoothing steps of full Vanka smoother, and UMFPACK as a coarse grid solver is used. Results are shown for the solvers using both fixed point and Newton methods.

Tables 8.9-8.12 compare the performance of solvers for $\tilde{Q}_1/Q_0$ discretization by presenting the number of solver iterations to convergence. Table 8.9 shows the results on a regular isotropic mesh with an aspect ratio (AR) equals one. On such a regular mesh, both MLKM and multigrid solvers show mesh width independent convergence behavior. Furthermore, both coupled solvers also show Reynolds number independent convergence on such meshes. However, as soon as the anisotropy is introduced into the mesh, the multigrid solver shows serious convergence issues, and the solver breaks down. Even with the higher number of smoothing steps, we could not find any configuration of the multigrid solver that can solve the problem on anisotropic meshes.

On the other hand, MLKM solver can robustly handle the mesh anisotropies and produce convergent results for both the fixed point method and the Newton method. Although the convergence behavior of the MLKM solver is adversely affected by the increase in the grid anisotropy, the solver produces grid size

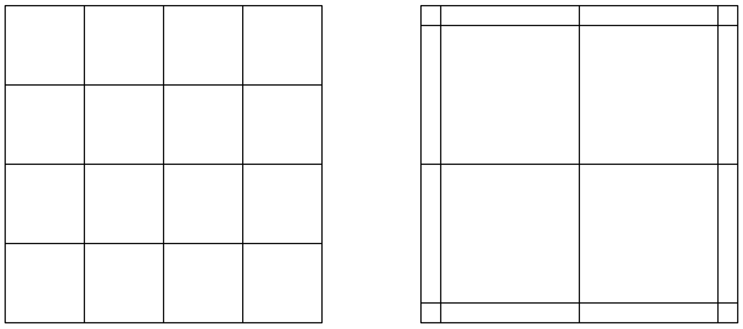


Figure 8.10: **LDC flow:** Mesh at the coarse level. "Regular mesh" with aspect ratio = 1 (left), anisotropic mesh with aspect ratio = 6 (right).

independent convergence rates, even on meshes with large anisotropies. Another interesting observation is that the convergence rate of the MLKM solver is in fact improved with the increase in the mesh refinement levels.

Tables 8.13-8.16 present results for the Q_2/P_1^{disc} discretization. Here again on a regular mesh (Table 8.13), all the solvers produce mesh width independent convergence rates. Nevertheless, multigrid with Newton as a nonlinear iteration fails to converge for $Re = 2000$, whereas, MLKM solver for a similar numerical situation not only converges but also produces convergence rates independent of the mesh size. This shows the superior robust behavior of the MLKM solver over the multigrid solver. On anisotropic meshes again multigrid solver diverges, while MLKM solver produces converged results.

Results also reveal the fact that on anisotropic grids $\tilde{Q}_1/Q_0$ discretization is more efficient as compared to the higher order Q_2/P_1^{disc} discretization, when solving low Reynolds number flows on highly stretched meshes (see Tables 8.12 and 8.16). The reason for this behavior is that for the case of $\tilde{Q}_1$ element we are using nonparametric version of the finite element. In nonparameteric finite element, the shape functions are defined independently on each physical element instead of the reference element. Such ansatz are robust with repect to the shape of the element, and hence the effect of the mesh anisotropy is minimal in this case. On the other hand, for the biquadratic Q_2 finite element, the nonparametric shape function is quartic along an edge that is not aligned with the coordinate axis, and therfore, cannot be uniquely defined with its value at three points on the edge. This will result into a broken element which is not continuous with the adjacent element. To circumvent this difficulty, the parametric finite elements with the shape functions defined on the reference element are used. The shape functions are then mapped back to the general elements using isoparamteric transformations. These mapped shape functions are quadratic along an element edge, and connect continuously to the adjacent element. However, the transformations from the reference element to the highly anisotropic meshes result in ill-conditioned matrices, which explains the poor convergence of MLKM solver for the case of conforming Q_2/P_1 finite elements. Many important engineering applications require solution of such low Reynolds number flows on anisotropic meshes, such as modeling of fluid flow in microelectromechanical systems (MEMS), and analysis of hydraulic performance of ventricular assist devices[110]. Our numerical experience suggests that lower order discretization such as $\tilde{Q}_1/Q_0$ should be the preferred choice for such applications, with coupled MLKM/fixed point as a numerical solver.

Another observation is that the effect of anisotropy is more pronounced in the case of low Reynolds number flows, while for the high Reynolds number flows the effect of anisotropy is not that significant. We can see from the results for fixed point linearization in Table 8.16, MLKM takes much less linear solver iterations for $Re = 2000$ than the number of linear solver iterations for $Re = 1$. The reason for this behavior is that anisotropy hits diffusion term harder than the convection term; since diffusion is more dominant in low Reynolds number flows, the discretized system in this case is more ill-conditioned than that for the highly convective flows. The high convection seems to stabilize the MLKM solver. Moreover, with the mesh refinements the 2nd order 'elliptic' diffusion operator becomes more dominant, we can observe the convergence rates of the MLKM solver become worse with mesh refinements.

The performance of the MLKM/Newton solver for solving high Reynolds number flows on isotropic grids is as expected, with quadratic convergence in the nonlinear iterations while having similar convergence rates of the linear solver as that for the fixed point iteration case. Therefore, the MLKM/Newton solver takes much less total linear solver iterations than the MLKM/fixed point solver. However, the performance of the MLKM/Newton solver for solving highly convective flows on anisotropic grids is not very convincing. Hence, for solving flow problems involving high Reynolds numbers and highly anisotropic meshes, we recommend $Q_2/P1$ discretization over $\tilde{Q}_1/Q_0$ and MLKM/fixed point solver over MLKM/Newton.

One question may arise that since the nonlinearity in the problem depends only on the Reynolds number, why in the results number of nonlinear iterations increase with the increase in the mesh anisotropy for a fixed Reynolds number? The answer to this question is that the anisotropic mesh is taken arbitrarily to make the problem harder for the numerical solvers, and is not adapted to the underlying solution. When the resulting linear system is solved with just one digit gain, it does not provide adequate correction to the nonlinear solution, and nonlinear solver has to do more work (more iterations) to reach to the convergence. If the linear solver stopping criterion is made more strict on the anisotropic grids, the number of nonlinear iterations to convergence are reduced.

Moreover, numerical results also depict that the convergence rates of Umfpack solver are not affected by either the mesh size, the mesh anisotropy, or the choice of finite element discretization. However, the huge memory requirements and computational costs of the solver make it unsuitable for the problems involving a higher number of unknowns, which is the case for most real life simulations.

	Umfpack		Multigrid		MLKM	
Lev	Fixed Pt.	Newton	Fixed Pt.	Newton	Fixed Pt.	Newton
			Re = 1			
5	3/1	3/1	6/1	3/2	8/4	4/7
6	3/1	3/1	5/1	3/2	8/4	4/7
7	**	**	5/1	3/2	7/4	4/7
8	**	**	5/1	3/2	7/4	4/7
			Re=1000			
5	18/1	4/1	18/2	5/5	18/4	4/8
6	**	**	15/2	5/4	14/4	4/8
7	**	**	12/1	4/3	12/4	4/8
8	**	**	10/1	5/3	11/4	4/8
			Re=2000			
5	33/1	5/1	33/2	5/6	28/5	5/13
6	**	**	22/2	4/7	21/5	4/9
7	**	**	17/2	5/7	17/4	4/8
8	**	**	14/1	5/6	16/4	4/8

**: Out of memory; *: Solver diverged

Table 8.9: **LDC flow:** Performance comparison of solvers on a grid with AR= **1** and $\tilde{Q}_1/Q_0$ discretization.

	Umfpack		Multigrid		MLKM	
Lev	Fixed Pt.	Newton	Fixed Pt.	Newton	Fixed Pt.	Newton
			Re = 1			
5	3/1	3/1	7/86	5/159	9/4	5/8
6	3/1	3/1	6/67	5/120	8/4	4/8
7	**	**	6/67	4/100	8/4	5/8
8	**	**	6/100	4/100	8/5	4/9
			Re=1000			
5	19/1	5/1	*	*	18/5	5/13
6	**	**	*	*	16/5	4/12
7	**	**	*	*	14/5	4/10
8	**	**	*	*	14/5	4/9
			Re=2000			
5	25/1	6/1	*	*	24/7	6/19
6	**	**	*	*	20/6	4/16
7	**	**	*	*	16/5	4/13
8	**	**	*	*	17/5	4/11

**: Out of memory; *: Solver diverged

Table 8.10: **LDC flow:** Performance comparison of solvers on a grid with AR= **6** and $\tilde{Q}_1/Q_0$ discretization.

	Umfpack		Multigrid		MLKM	
Lev	Fixed Pt.	Newton	Fixed Pt.	Newton	Fixed Pt.	Newton
			Re = 1			
5	3/1	3/1	*	*	10/7	5/12
6	3/1	2/1	*	*	10/7	5/13
7	**	**	*	*	10/7	5/14
8	**	**	*	*	10/7	5/15
			Re=1000			
5	21/1	5/1	*	*	27/7	6/23
6	**	**	*	*	19/7	5/22
7	**	**	*	*	14/7	4/19
8	**	**	*	*	13/7	5/17
			Re=2000			
5	21/1	6/1	*	*	34/9	6/36
6	**	**	*	*	23/10	5/34
7	**	**	*	*	20/8	5/25
8	**	**	*	*	21/6	5/20

**: Out of memory; *: Solver diverged

Table 8.11: **LDC flow:** Performance comparison of solvers on a grid with AR= **12** and $\tilde{Q}_1/Q_0$ discretization.

	Umfpack		Multigrid		MLKM	
Lev	Fixed Pt.	Newton	Fixed Pt.	Newton	Fixed Pt.	Newton
			Re = 1			
5	3/1	3/1	*	*	10/14	5/30
6	3/1	3/1	*	*	11/17	5/32
7	**	**	*	*	11/15	5/30
8	**	**	*	*	10/16	5/32
			Re=1000			
5	24/1	6/1	*	*	25/11	6/51
6	**	**	*	*	20/10	5/41
7	**	**	*	*	17/11	5/37
8	**	**	*	*	17/11	5/36
			Re=2000			
5	28/1	12/1	*	*	36/18	12/119
6	**	**	*	*	25/18	6/79
7	**	**	*	*	25/12	6/70
8	**	**	*	*	25/13	6/42

**: Out of memory; *: Solver diverged

Table 8.12: **LDC flow:** Performance comparison of solvers on a grid with AR= **24** and $\tilde{Q}_1/Q_0$ discretization.

	Umfpack		Multigrid		MLKM	
Lev	Fixed Pt.	Newton	Fixed Pt.	Newton	Fixed Pt.	Newton
			Re = 1			
5	4/1	3/1	5/1	3/2	7/4	4/7
6	3/1	**	5/1	3/2	7/4	4/7
7	**	**	5/1	3/2	8/4	4/7
			Re=1000			
5	12/1	4/1	12/2	4/5	14/6	4/12
6	**	**	10/1	4/3	13/5	4/9
7	**	**	7/1	4/2	12/5	4/7
			Re=2000			
5	19/1	4/1	19/3	*	19/8	5/19
6	**	**	15/2	*	16/5	4/13
7	**	**	11/2	*	16/5	4/9

**: Out of memory; *: Solver diverged

Table 8.13: **LDC flow:** Performance comparison of solvers on a grid with AR= **1** and Q_2/P_1^{disc} discretization.

	Umfpack		Multigrid		MLKM	
Lev	Fixed Pt.	Newton	Fixed Pt.	Newton	Fixed Pt.	Newton
			Re = 1			
5	3/1	3/1	7/87	4/199	9/9	5/18
6	3/1	**	*	*	9/10	5/19
7	**	**	*	*	9/10	5/20
			Re=1000			
5	8/1	4/1	*	*	16/6	5/23
6	**	**	*	*	12/6	4/16
7	**	**	*	*	12/6	4/18
			Re=2000			
5	15/1	4/1	*	*	47/8	7/52
6	**	**	*	*	30/6	5/28
7	**	**	*	*	16/5	5/18

**: Out of memory; *: Solver diverged

Table 8.14: **LDC flow:** Performance comparison of solvers on a grid with AR= **6** and Q_2/P_1^{disc} discretization.

	Umfpack		Multigrid		MLKM	
Lev	Fixed Pt.	Newton	Fixed Pt.	Newton	Fixed Pt.	Newton
			Re = 1			
5	3/1	2/1	*	*	10/17	5/35
6	**	**	*	*	10/21	5/45
7	**	**	*	*	10/24	5/49
			Re=1000			
5	14/1	3/1	*	*	15/9	5/41
6	**	**	*	*	14/9	5/47
7	**	**	*	*	14/10	4/49
			Re=2000			
5	14/1	4/1	*	*	not conv.	5/88
6	**	**	*	*	40/7	5/59
7	**	**	*	*	19/8	5/42

**: Out of memory; *: Solver diverged

Table 8.15: **LDC flow:** Performance comparison of solvers on a grid with AR= **12** and Q_2/P_1^{disc} discretization.

	Umfpack		Multigrid		MLKM	
Lev	Fixed Pt.	Newton	Fixed Pt.	Newton	Fixed Pt.	Newton
			Re = 1			
5	3/1	2/1	*	*	10/28	5/54
6	**	**	*	*	10/42	5/80
7	**	**	*	*	11/51	5/100
			Re=1000			
5	11/1	3/1	*	*	15/19	6/58
6	**	**	*	*	14/21	5/77
7	**	**	*	*	14/22	5/100
			Re=2000			
5	19/1	4/1	*	*	55/16	8/141
6	**	**	*	*	40/11	6/92
7	**	**	*	*	22/14	6/109

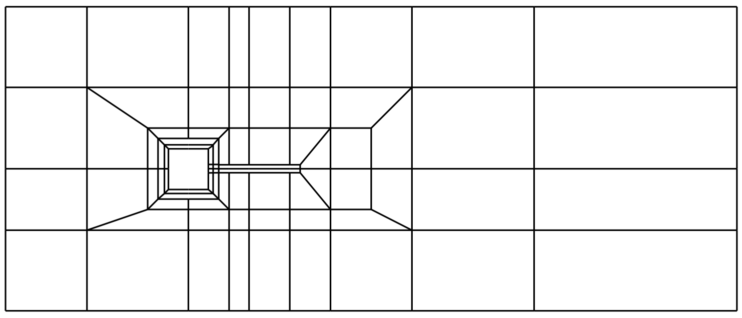
**: Out of memory; *: Solver diverged

Table 8.16: **LDC flow:** Performance comparison of solvers on a grid with AR= **24** and Q_2/P_1^{disc} discretization.

8.3.2 Stationary channel flow around a square

This numerical test is very similar to the DFG benchmark flow around cylinder problem, where the circular obstacle in the channel is replaced with the square one. This enables us to easily introduce anisotropies in the mesh, with the aim to study the robustness behavior of various monolithic Navier-Stokes solvers. The problem domain comprises a rectangular channel of height $H = 0.75\ m$ and length $L = 1.8\ m$, with a square obstacle of side $s = 0.1\ m$ placed at right angle to the flow inside the channel. The square is positioned with its center at the point (0.45, 0.35). No slip boundary conditions are imposed on the upper and lower walls of the channel and the boundary of the square obstacle. A parabolic inflow profile with a maximum velocity $U_{max} = 1.0\ m/s$ is provided at the left boundary, and the right wall is taken as an outflow boundary imposed with do nothing natural boundary condition. We consider an incompressible Newtonian fluid with constant density $\rho = 1.0\ kg/m^3$, and the numerical tests are run for various values of Kinematic viscosity parameter $\nu = \{1/5, 1/50, 1/500\}$. All these numerical tests produce laminar and stationary flow past the square cylinder.

$\tilde{Q}_1/Q_0$ finite element quadrilaterals are used for the discretization of the domain, along with the edge oriented jump stabilization for the unsmooth problems. Figure 8.11 shows the coarse mesh at level 1 with moderately anisotropic mesh of aspect ratio 10; the meshes at the refined levels are obtained by joining the opposite midpoints of each cell of the immediate coarse mesh. Meshes with higher anisotropies are obtained by increasing the aspect ratio of the innermost elements around the square and keeping all the other elements in the mesh unchanged. This problem has also been solved in [132], and our calculated drag and lift values are in good agreement with this work (see Table 8.17).



Level	Cells	Degrees of freedom
1	86	466
3	1,376	7,024
4	5,504	27,808
5	22,016	110,656
6	88,064	441,472

Figure 8.11: **FAS problem:** Coarse mesh with $AR \sim 10$ (top); Problem size at various mesh levels (bottom).

In Tables 8.18-8.20, we show the results of monolithic MLKM and MG solvers with fixed point method as nonlinear iteration, for solving stationary Navier-Stokes equation on meshes with aspect ratios $AR = \{10, 20, 40\}$. Our numerical results have shown that MLKM (2,2,5) is the best configuration for these problems, and an increase in the number of FGMRES iterations at any level does not improve the convergence behavior of the solver. All the other numerical settings of the MLKM solver remain the same as described in the LDC flow problem. Similarly, all the numerical configurations of the MG solver are the same as that used in the LDC flow problem, except for the number of smoothing steps. Unlike LDC flow problem, in the case of flow around a square (FAS) problem, increasing the number of smoothing steps produces converged results for the MG solver on the stretched grids.

The results show that on a moderately anisotropic mesh with aspect ratio 10, both linear solvers produce (almost) mesh width and Reynolds number independent convergence behavior. On meshes with higher aspect ratios, MG with 4 smoothing steps breaks down. Our numerical experience showed that if the aspect ratio in the grid is doubled, smoothing steps need to be increased 4 times to get the converged results with the MG solver. Even with such an increase in smoothing steps, we can see from the Table 8.18, MG with 64 smoothing steps on a grid with aspect ratio 40, the solver diverged for Reynolds number 500 on mesh refinement level 6. On the other hand, MLKM(2,2,5) successfully produces converged results on the grids with higher aspect ratios. Although the ideally desired mesh size independent convergence behavior of MLKM solver is lost, the total CPU time of the MLKM solver is still less than that of MG solver, for most cases.

Lev	MLKM	Ref.[132]
	Re = 5	
3	5.0462+1 / 0.1297+1	4.8239+1 / 0.1241+1
4	5.1685+1 / 0.1329+1	5.0098+1 / 0.1292+1
5	5.2100+1 / 0.1341+1	5.1009+1 / 0.1316+1
6	5.2250+1 / 0.1345+1	5.1507+1 / 0.1330+1
	Re = 50	
3	6.0601+0 / 0.2029-0	6.0175+0 / 0.1904-0
4	6.1880+0 / 0.2070-0	6.0820+0 / 0.2000-0
5	6.2306+0 / 0.2091-0	6.1274+0 / 0.2047-0
6	6.2461+0 / 0.2100-0	6.1651+0 / 0.2072-0
	Re = 500	
3	1.7955+0 / -0.1235-1	1.7795+0 / -0.7125-2
4	1.7279+0 / -0.6910-2	1.7195+0 / -0.5231-2
5	1.6974+0 / -0.4840-2	1.6843+0 / -0.4375-2
6	1.6886+0 / -0.4090-2	1.6733+0 / -0.4459-2

Table 8.17: **FAS problem:** Comparison of the calculated drag and lift values on a mesh with $AR = 10$ with the reference results.

	Multigrid (4)*		MLKM (2,2,5)	
Lev	Iterations	Time	Iterations	Time
		Re = 5		
3	5/2	0.31	7/5	0.30
4	5/2	0.74	7/5	0.99
5	6/2	3.15	6/7	3.74
6	5/2	12.38	6/7	16.24
		Re= 50		
3	7/1	0.33	8/6	0.35
4	6/2	0.83	7/6	1.08
5	5/2	2.81	6/6	3.78
6	4/2	10.78	6/7	16.60
		Re= 500		
3	15/2	1.50	15/7	1.31
4	12/2	4.72	13/6	4.43
5	12/2	19.89	11/6	14.98
6	11/2	77.50	10/7	62.58

*: Multigrid with 4 pre and post smoothing steps.

Table 8.18: **FAS problem:** Performance comparison of solvers on a grid with $AR = 10$ and $\tilde{Q}_1/Q_0$ discretization.

	Multigrid (16)*		MLKM (2,2,5)	
Lev	Iterations	Time	Iterations	Time
		Re = 5		
3	5/1	0.34	7/6	0.32
4	5/1	1.07	7/8	1.19
5	5/2	5.12	7/10	5.30
6	5/2	28.51	7/12	25.77
		Re= 50		
3	7/1	0.43	7/6	0.32
4	6/1	1.09	7/8	1.27
5	5/2	4.65	7/10	5.28
6	5/2	28.48	6/11	22.52
		Re= 500		
3	15/1	1.58	15/8	1.43
4	13/2	6.80	15/9	5.96
5	11/2	23.87	13/12	24.69
6	10/2	103.69	11/19	121.07

*: Multigrid with 16 pre and post smoothing steps.

Table 8.19: **FAS problem:** Performance comparison of solvers on a grid with $AR = 20$ and $\tilde{Q}_1/Q_0$ discretization.

	Multigrid (64)*		MLKM (2,2,5)	
Lev	Iterations	Time	Iterations	Time
		Re = 5		
3	4/1	0.62	8/6	0.36
4	4/1	2.04	8/12	1.71
5	5/2	15.38	8/17	8.72
6	5/3	157.54	8/31	65.80
		Re= 50		
3	6/1	0.89	8/6	0.36
4	5/1	3.02	7/12	1.54
5	5/1	14.19	7/17	7.64
6	5/3	136.54	7/29	53.84
		Re= 500		
3	15/1	3.03	17/10	1.95
4	13/5	52.40	15/20	11.40
5	11/2	86.80	14/33	67.21
6	**	-	13/67	442.29

*: Multigrid with 64 pre and post smoothing steps.
**: Solver diverged.

Table 8.20: **FAS problem:** Performance comparison of solvers on a grid with $AR = 40$ and $\tilde{Q}_1/Q_0$ discretization.

9 Summary and outlook

We close this thesis by briefly summarizing the presented work and drawing some conclusions from the conducted studies, and finally proposing possible future extensions in work.

9.1 Summary and Conclusions

The focus of this work has been on the development, implementation and numerical analysis of a new class of robust and efficient monolithic finite-element multilevel Krylov subspace solvers for the numerical solution of stationary incompressible Navier-Stokes equations, using FEATFLOW software library. Developing such a solution algorithm that possesses both the highly sort-after properties, namely robustness and efficiency, is not an easy task. Two most widely used classes of iterative methods for the solution of incompressible Navier-Stokes equations are preconditioned Krylov subspace solvers and multigrid methods. Krylov subspace solvers are robust but in general not so fast; their convergence rates deteriorate as the problem size increases due to the mesh refinements. Multigrid methods are fast with the convergence rates independent of the problem size; however, they are not very robust and often face convergence issues. An *ideal* solver for the Navier-Stokes equations is the one that is robust and fast and produces convergence rates that are independent of the problem size, mesh shape, and Reynolds number.

We have proposed a new monolithic multilevel Krylov subspace algorithm to solve stationary incompressible Navier-Stokes equations using finite element methods. Our solver is based on the multilevel projection-based Krylov subspace method proposed by Nabben and Erlangga [54] to solve the scalar partial differential equations. In chapter 4, the working principle and the construction of this method have been discussed in detail. The idea of this projection method is to accelerate the convergence rate of the Krylov subspace solver by clustering the small eigenvalues of system matrix, responsible for the slow convergence of Krylov subspace solvers, around the largest one. Galerkin system related to the linear system contains the information about these problematic eigenvalues. In [54], authors have used Galerkin coarse grid projection to represent the Galerkin system, i.e., $\hat{\mathbf{E}} = \mathbf{Y}^T(\mathbf{A}\mathbf{M}^{-1})\mathbf{Z}$. A disadvantage of this approach is that it requires explicit calculation of the inverse preconditioner matrix $\mathbf{M}^{-1}$ at every mesh level in the initialization phase of the algorithm. Such matrices are in general dense, except for the Jacobi preconditioning; however the diagonal preconditioning is not very efficient in most practical problems, and stronger preconditioners are often required. Moreover, in the particular case of Navier-Stokes equations, diagonal preconditioning is not applicable due to the presence of zero diagonal block, and one has to look for other preconditioning possibilities. Calculating and storing the dense $\mathbf{M}^{-1}$ matrices is very expensive, thus making it impossible to use preconditioners other than Jacobi preconditioners in the MLKM solver. Our implementation of MLKM solver in the FEATFLOW software is different from the MLKM algorithm in [Erlangga2008]; it uses discretization coarse grid approximation to setup Galerkin matrices at the coarse levels. The benefit of this approach is that the explicit calculation of the $\mathbf{M}^{-1}$ matrix is not required. Implementation of the MLKM solver in the context of FEATFLOW software has been explained in detail in chapter 4 and its pseudocode is given in algorithm 4.3.

This implementation of MLKM algorithm gives the freedom of using any iterative method as a preconditioner in the MLKM solver. The solver becomes more flexible in a sense that it can use weaker but faster preconditioners for simple problems, and stronger but complicated preconditioners for more harder and ill-conditioned problems. Jacobi, Gauss-Seidel, and ILU have been successfully used as preconditioners to MLKM solver in this thesis to solve the convection-diffusion and anisotropic diffusion problems. Similarly, *local pressure Schur complement techniques* in combination with MLKM has resulted in a new class of efficient and robust monolithic multilevel Krylov subspace solvers for the stationary incompressible Navier-Stokes equations. Our monolithic MLKM solver can be considered as an alternative to the already implemented monolithic multigrid method in the FEATFLOW software.

MLKM combines the ideas of Krylov subspace solvers and MG methods. Like Krylov subspace solvers, it extracts the solution by making projections on to the Krylov subspaces; moreover, it uses grid hierarchies similar to the multigrid methods, during the course of its action. Consequently, it inherits properties of both the solver classes, i.e., robustness from the Krylov subspace solvers and efficiency from the multigrid methods. The solver has been tested for the solution of convection-diffusion problems, anisotropic diffusion problems, and incompressible Navier-Stokes problems. Numerical studies presented in the thesis have shown that MLKM solver produces level independent convergence rates, a typical trait of the MG methods. Numerical results have also revealed that MLKM solver is more robust than the MG method towards handling the anisotropies in the problem and solving the flows involving higher Peclet/ Reynolds numbers.

Numerical results for the convection-diffusion problem are presented in chapter 5. The problem is solved for various flow configurations, and the results show that MLKM solver is robust with respect to the underlying flow configuration. Numerical tests also show that MLKM solver worked equally fine with bilinear as well as biquadratic finite elements, structured as well as unstructured grids. High inertia flow simulations produce instabilities in the numerical solution, leading to wrong solutions and convergence issues for numerical solvers; this can be circumvented by using numerical stabilization techniques. In this work, we have used edge-oriented jump stabilization to stabilize the convective terms, which has worked successfully with the MLKM solver. Numerical experiments have also revealed that in comparison with the multigrid solver, MLKM solver can more robustly solve the convection dominated problems. With the tested smoothers/ preconditioners, MG is divergent for high Pe numbers, while MLKM not only converged but also produced mesh-level and Peclet number independent convergence rates.

Anisotropic diffusion problem results are shown in chapter 6, where the solver is tested for operator-based as well as grid-based anisotropies. When the anisotropy is aligned with the grid, MG/Jacobi solver is divergent whereas MLKM/Jacobi is always convergent. MG/ILU(0) with suitable renumbering scheme produced better convergence rates than MLKM/ILU(0) for grid aligned anisotropies; however, the computational costs of MLKM are of the same order as that of MG solver, in this case. When the anisotropy is at an angle to the grid lines, MLKM solver always produced convergence rates superior to that of the MG solver.

Finally, we presented the numerical results of our monolithic MLKM solver for the solution of Navier-Stokes equations in chapter 8. The solution technique is validated through a couple of well-known benchmark problems. For the lid-driven cavity flow benchmark, the total kinetic energy and velocity component profiles are calculated. In the second benchmark, drag and lift values are calculated for the flow around a cylinder at $Re = 20$. These validation studies are carried out using nonconforming $\tilde{Q}_1/Q_0$ and conforming Q_2/P_1 mixed finite element formulations. The results of benchmark studies for both the formulations are found in excellent agreement with the previously published results in the literature.

Further, numerical studies are conducted on a model problem to see the effect of various solver parameters on the overall convergence behavior of the solver. Based on these experiments, an optimal configuration of the coupled MLKM solver is proposed. This optimal configuration of the MLKM solver is used for the performance comparison studies with other monolithic solvers. To compare the robustness of our monolithic MLKM solver with other existing monolithic solvers in FEATFLOW, lid-driven cavity flow and flow around a square problems are solved for a range of Reynolds numbers and grid anisotropies.

Numerical results for the lid-driven cavity flow have shown that on isotropic meshes, both multilevel coupled solvers produced mesh parameter (h) and Re independent convergence rates. Thus, MLKM solver is perfectly scalable on isotropic meshes, i.e., the number of iterations does not scale with the increase in the number of degrees of freedom. On anisotropic meshes MG is divergent, and we could not find any converging MG configuration. On the other hand, monolithic MLKM solver is always convergent and produced mesh size independent convergence behaviors in most cases. In the case of flow around a square problem, although convergence rates of monolithic MLKM solver are not mesh-level independent, the overall CPU times to convergence of the MLKM solver are still better than that for the MG solver.

9.2 Outlook

We have established through code validations and performance comparison studies with the MG solver, that the new coupled MLKM solver is an efficient and robust solution algorithm. We end this thesis by making some recommendations for the possible future extensions of the presented work.

Application to non-Newtonian and non-isothermal problems The monolithic multilevel Krylov subspace solver with local pressure Schur complement preconditioner provides a framework that can be applied to any problem of the saddle point type. One possible extension of the present work can be to apply the solver to more complex non-Newtonian or non-isothermal flows.

Governing equations for non-Newtonian flows are very similar to the standard Navier-Stokes equations 7.1, with the exception that now they have additional nonlinearity in the diffusion operator, due to the nonconstant viscosity. The resulting algebraic system obtained after linearization will be more complex and involve more terms (see [109]). This requires that the inner linear solver in the monolithic solution algorithm should be robust. We hope that our monolithic MLKM solver with robust inner linear solver in combination with the optimal damping in the outer nonlinear iteration will result in an efficient and robust overall solution technique for non-Newtonian fluids.

Non-isothermal flows involve temperatures that are not constant. This brings into play an additional energy equation in the standard Navier-Stokes equations. Since fluid flow transports heat, a change in the flow field will affect the temperature field as well. Thus momentum, continuity and energy equations form a coupled system of equations, and the monolithic MLKM solver can be used to solve non-isothermal flows in a coupled way. The introduction of the temperature equation, however, changes the structure of the resulting local system, and the local pressure Schur complement method has to be adapted accordingly.

Parallel implementation Cell-oriented Vanka preconditioner acts locally on a single mesh cell at a time. This local procedure can be parallelized in a straightforward way by embedding it into an outer block Jacobi iteration. From our experience with the scalar problems, we have seen that MG with Jacobi smoother fails to produce a convergent solution scheme at higher Peclet numbers. However, MLKM solver with Jacobi preconditioner produced convergence rates which are equally as good as with the Gauss-Seidel preconditioner (see Tables 5.9, 5.11,5.13, 5.15); in fact the convergence rates with Jacobi preconditioner are even better than Gauss-Seidel preconditioner for large Pe numbers. Based on this experience, we hope that Vanka preconditioning with an outer block Jacobi iteration, instead of block Gauss-Seidel iteration, will not adversely affect the overall convergence behavior of the solver. MLKM algorithm involves two preconditioner applications in every iteration, and this preconditioning task is one of the most computationally intensive operations of the algorithm. A parallel implementation of this task will improve the efficiency of the solver considerably.

Moreover, a significant advantage of MLKM method over the MG method is the liberty in the choice of inter-grid transfer operators (deflation subspaces in the context of MLKM solver). In MLKM algorithm, the only condition on these deflation subspaces is that they should be full rank, and they are not required to do accurate interpolations like that required in MG. Exploiting this freedom, one can select such deflation subspaces that facilitate the parallel implementation of the algorithm. Nabben and Erlangga in [54] have used subdomain deflation, which is nothing but piecewise constant interpolations. Frank and Viuk [60] have given an efficient parallel implementation of the subdomain deflation on the distributed memory system with small communication overhead.

Patch based LPSC When there are large jumps in the size/aspect ratio of the two neighboring cells in the triangulation, the convergence rates of cell-oriented LPSC with Jacobi/Gauss-Seidel iteration deteriorate significantly. Turek [132] has used patch-based local pressure Schur complement smoother to increase the robustness of the coupled MG solver. The idea of the patch based LPSC is to collect all elements of nearly the same size and shape into a patch. These patches hide the anisotropies, and the resulting small ill-conditioned matrices are solved exactly. The outer global convergence behavior of LPSC becomes independent of the grid anisotropies. This patch based LPSC preconditioner can also be applied to monolithic MLKM solver, and the resulting solver is expected to produce convergence rates independent of the mesh anisotropies.

3D Extension The most important feature of the monolithic MLKM solver is its numerical scalability. We have seen that the computational complexity of the MLKM solver is linear, i.e., the computational work of the solver increases linearly with the number of unknowns. Moreover, the convergence rates of the solver are independent of the mesh refinement level. These features make monolithic MLKM solver a potential solver candidate for the solution of 3D CFD simulations that involve refined meshes due to high accuracy requirements.

Code optimization In this work, we have made comparisons with the monolithic multigrid solver being implemented in FEATFLOW software. This implementation of the multigrid solver is highly optimized. Our implementation of the monolithic MLKM solver is not optimal, and a careful and optimized implementation can further improve the CPU times of the solver.

Bibliography

[1] C. K. Aidun, T. N. Triantafillopoulos, and J. D. Benson. “Global stability of a lid-driven cavity with throughflow: Flow visualization studies”. In: *Physics of Fluids A: Fluid Dynamics* 3.9 (1991), pp. 2081–2091.

[2] P.K. Amestoy et al. “Encyclopedia of Parallel Computing - Multifrontal Method”. In: Boston, MA: Springer US, 2011, pp. 1209–1216.

[3] E. Anderson et al. “LAPACK: A portable linear algebra library for high-performance computers”. In: *Proceedings of the 1990 ACM/IEEE conference on Supercomputing.* IEEE Computer Society Press. 1990, pp. 2–11.

[4] T. Apel. *Anisotropic finite elements: local estimates and applications.* Vol. 3. Citeseer, 1999.

[5] D. Arnold, D. Boffi, and R. Falk. “Approximation by quadrilateral finite elements”. In: *Mathematics of computation* 71.239 (2002), pp. 909–922.

[6] O. Axelsson. “A generalized SSOR method”. In: *BIT Numerical Mathematics* 12.4 (1972), pp. 443–467.

[7] O. Axelsson. *Iterative solution methods.* Cambridge university press, 1996.

[8] O. Axelsson and M. Neytcheva. “Preconditioning methods for linear systems arising in constrained optimization problems”. In: *Numerical linear algebra with applications* 10.1-2 (2003), pp. 3–31.

[9] I. Babushka. *Accuracy estimates and adaptive refinements in finite element computations.* John Wiley & Sons, 1986.

[10] I. Babushka, J. Chandra, and J. E. Flaherty. *Adaptive computational methods for partial differential equations.* Vol. 16. Siam, 1983.

[11] I. Babuška. “The finite element method with Lagrangian multipliers”. In: *Numerische Mathematik* 20.3 (1973), pp. 179–192.

[12] R. Barrett et al. *Templates for the solution of linear systems: Building blocks for iterative methods.* Philadelphia: SIAM, 1994.

[13] M. Benzi. “Preconditioning Techniques for Large Linear Systems: A Survey”. In: *Journal of Computational Physics* 182.2 (2002), pp. 418–477.

[14] M. Benzi, G. H. Golub, and J. Liesen. “Numerical solution of saddle point problems”. In: *Acta numerica* 14 (2005), pp. 1–137.

[15] J. Bey and G. Wittum. “Downwind numbering: Robust multigrid for convection-diffusion problems”. In: *Applied Numerical Mathematics* 23.1 (1997). Multilevel Methods, pp. 177–192. ISSN: 0168-9274. DOI: `https://doi.org/10.1016/S0168-9274(96)00067-0`. URL: `http://www.sciencedirect.com/science/article/pii/S0168927496000670`.

[16] D. Braess and W. Dahmen. “A cascadic multigrid algorithm for the Stokes equations”. In: *Numerische Mathematik* 82.2 (1999), pp. 179–191.

[17] D. Braess and R. Sarazin. “An efficient smoother for the Stokes problem”. In: *Applied Numerical Mathematics* 23.1 (1997), pp. 3–19.

[18] A. Brandt and N. Dinar. “Multigrid solutions to elliptic flow problems”. In: *Numerical methods for partial differential equations.* Elsevier, 1979, pp. 53–147.

[19] A. Brandt and O. E. Livne. *Multigrid techniques: 1984 guide with applications to fluid dynamics.* Vol. 67. SIAM, 2011.

[20] A. Brandt, S. F. McCormick, and J. W. Ruge. "Algebraic multigrid (AMG) for sparse matrix equations." In: *In: Evans, D. J. eds. Sparsity and its application* (1984), pp. 257–284.

[21] S. Brenner and R. Scott. *The mathematical theory of finite element methods.* Vol. 15. Springer Science & Business Media, 2007.

[22] F. Brezzi. "On the existence, uniqueness and approximation of saddle-point problems arising from Lagrangian multipliers". In: *Revue française d'automatique, informatique, recherche opérationnelle. Analyse numérique* 8.R2 (1974), pp. 129–151.

[23] Franco Brezzi and Michel Fortin. *Mixed and Hybrid Finite Element Methods.* Berlin, Heidelberg: Springer-Verlag, 1991. ISBN: 0-387-97582-9.

[24] W. L. Briggs, V. E. Henson, and S. F. McCormick. *A Multigrid Tutorial.* SIAM, 2000.

[25] A. N. Brooks and T. Hughes. "Streamline upwind/Petrov-Galerkin formulations for convection dominated flows with particular emphasis on the incompressible Navier-Stokes equations". In: *Computer methods in applied mechanics and engineering* 32.1-3 (1982), pp. 199–259.

[26] C. H. Bruneau and M. Saad. "The 2D lid-driven cavity problem revisited". In: *Computers & Fluids* 35.3 (2006), pp. 326–348.

[27] N. I. Buleev. "A numerical method for solving two-dimensional diffusion equations". In: *Atomic Energy* 6.3 (1960), pp. 222–224.

[28] G. F. Carey, K. C. Wang, and W. D. Joubert. "Performance of iterative methods for Newtonian and generalized Newtonian flows". In: *International Journal for Numerical Methods in Fluids* 9.2 (Feb. 1989), pp. 127–150. ISSN: 1097-0363. DOI: 10.1002/fld.1650090202. URL: http:https://doi.org/10.1002/fld.1650090202.

[29] Edmond. Chow and Y. Saad. "Experimental study of ILU preconditioners for indefinite matrices". In: *Journal of Computational and Applied Mathematics* 86.2 (1997), pp. 387–414.

[30] B. A. Cipra. "The Best of the 20th Century: Editors Name Top 10 Algorithms". In: *SIAM News* 33.4 (2000).

[31] P. Concus, G. H. Golub, and D. P. O. Leary. "A Generalized Conjugate Gradient Method for the Numerical Solution of Elliptic Partial Differential Equations". In: *in Sparse Matrix Computations, J. Bunch and D. Rose, eds.* (1976), pp. 309–332.

[32] E. Cuthill. "Several Strategies for Reducing the Bandwidth of Matrices". In: *Sparse Matrices and their Applications: Proceedings of a Symposium on Sparse Matrices and Their Applications.* Ed. by D. J. Rose and R. A. Willoughby. Boston, MA: Springer US, 1972, pp. 157–166. ISBN: 978-1-4615-8675-3. DOI: 10.1007/978-1-4615-8675-3_14. URL: https://doi.org/10.1007/978-1-4615-8675-3_14.

[33] E. Cuthill and J. McKee. "Reducing the Bandwidth of Sparse Symmetric Matrices". In: *Proceedings of the 1969 24th National Conference*. ACM '69. New York, NY, USA: ACM, 1969, pp. 157–172. DOI: `10.1145/800195.805928`. URL: `http://doi.acm.org/10.1145/800195.805928`.

[34] C. Cuvelier, A. Segal, and A. A. van Steenhoven. *Finite element methods and Navier-Stokes equations*. Vol. 22. Springer Science & Business Media, 1986.

[35] C. Cuvelier, A. Segal, and A. A. van Steenhoven. *Introduction to the Finite Element Method*. D. Reidel Publishing Company, 1986.

[36] O. Dahl and S. Ø Wille. "An ILU preconditioner with coupled node fill-in for iterative solution of the mixed finite element formulation of the 2D and 3D Navier-Stokes equations". In: *International Journal for Numerical Methods in Fluids* 15.5 (1992), pp. 525–544. ISSN: 1097-0363. DOI: `10.1002/fld.1650150503`. URL: `http://dx.doi.org/10.1002/fld.1650150503`.

[37] H. Damanik. "FEM Simulation of Non–isothermal Viscoelastic Fluids". PhD thesis. TU Dortmund, May 2011.

[38] T. A. Davis. "Algorithm 832: UMFPACK V4.3–an unsymmetric-pattern multifrontal method". In: *ACM Trans. Math. Softw.* 30.2 (2004), pp. 196–199.

[39] T. A. Davis. *Direct methods for sparse linear system*. Philadelphia, PA: SIAM, 2006.

[40] T. A. Davis and I. S. Duff. "An unsymmetric-pattern multifrontal method for sparse LU factorization". In: *SIAM Journal on Matrix Analysis and Applications* 18.1 (1997), pp. 140–158.

[41] T. A. Davis and Y. Hu. "The University of Florida sparse matrix collection". In: *ACM Transactions on Mathematical Software (TOMS)* 38.1 (2011), p. 1.

[42] T. A. Davis, S. Rajamanickam, and W. M. Sid-Lakhdar. "A survey of direct methods for sparse linear systems". In: *Acta Numerica* 25 (2016), pp. 383–566. DOI: `10.1017/S0962492916000076`.

[43] S. L. Dettmer et al. "Anisotropic diffusion of spherical particles in closely confining microchannels". In: *Physical Review E* 89.6 (2014).

[44] J. Donea and A. Huerta. *Finite element methods for flow problems*. John Wiley & Sons, 2003.

[45] J. Douglas and T. Dupont. "Interior penalty procedures for elliptic and parabolic Galerkin methods". In: *Computing methods in applied sciences* (1976), pp. 207–216.

[46] M. Dryja and O. B. Widlund. "Schwarz methods of Neumann-Neumann type for three-dimensional elliptic finite element problems". In: *Communications on pure and applied mathematics* 48.2 (1995), pp. 121–155.

[47] I. S. Duff, I. Erisman, and J. Reid. *Direct methods for sparse matrices*. London, England: Oxford University Press, 1986.

[48] I. S. Duff and J. K. Reid. "The Multifrontal Solution of Indefinite Sparse Symmetric Linear". In: *ACM Trans. Math. Softw.* 9.3 (Sept. 1983), pp. 302–325.

[49] T. Dupont, R. P. Kendall, and H.H. Rachford Jr. “An approximate factorization procedure for solving self-adjoint elliptic difference equations”. In: *SIAM Journal on Numerical Analysis* 5.3 (1968), pp. 559–573.

[50] L. C. Dutto. “The effect of ordering on preconditioned GMRES algorithm, for solving the compressible Navier-Stokes equations”. In: *International Journal for Numerical Methods in Engineering* 36.3 (1993), pp. 457–497.

[51] H. C. Elman and E. Agrón. “Ordering techniques for the preconditioned conjugate gradient method on parallel computers”. In: *Computer Physics Communications* 53.1 (1989), pp. 253–269. ISSN: 0010-4655. DOI: `https://doi.org/10.1016/0010-4655(89)90164-1`. URL: `http://www.sciencedirect.com/science/article/pii/0010465589901641`.

[52] H. C. Elman, D. J. Silvester, and A. J. Wathen. *Finite elements and fast iterative solvers: with applications in incompressible fluid dynamics.* Numerical Mathematics & Scientific Computation, 2014.

[53] Y. A. Erlangga and R. Nabben. “Deflation and Balancing Preconditioners for Krylov Subspace Methods Applied to Nonsymmetric Matrices”. In: *SIAM Journal on Matrix Analysis and Applications* 30.2 (2008), pp. 684–699.

[54] Y. A. Erlangga and R. Nabben. “Multilevel Projection-Based Nested Krylov Iteration for Boundary Value Problems”. In: *SIAM J. Sci. Comput.* 30.3 (2008), pp. 1572–1595.

[55] E. Erturk, T. C. Corke, and C. Gockol. “Numerical solutions of 2-D steady incompressible driven cavity flow at high Reynolds numbers”. In: *International Journal for Numerical Methods in Fluids* 48.7 (2005), pp. 747–774. ISSN: 1097-0363. DOI: `10.1002/fld.953`. URL: `http://dx.doi.org/10.1002/fld.953`.

[56] V. Faber and T. Manteuffel. “Necessary and Sufficient Conditions for the Existence of a Conjugate Gradient Method”. In: *SIAM Journal on Numerical Analysis* 21.2 (1984), pp. 352–362. DOI: `10.1137/0721026`. eprint: `https://doi.org/10.1137/0721026`. URL: `https://doi.org/10.1137/0721026`.

[57] J. E. Flaherty. *Lecture Notes: Finite element analysis CSCI-6860 / MATH-6860.* `http://www.cs.rpi.edu/~flaherje/feaframe.html`.

[58] R. Fletcher. “Conjugate Gradient Methods for Indefinite Systems”. In: *In: Watson, G., Ed., Numerical Analysis Dundee 1975, Lecture Notes in Mathematics* 506 (1976), pp. 73–89.

[59] A. Fortin. “Old and new finite finite elements for incompressible flows”. In: *Int. J. for Numer. Meth. in Fluids.* 1 (1981), pp. 347–364.

[60] J. Frank and C. Vuik. “On the construction of deflation-based preconditioners”. In: *SIAM Journal on Scientific Computing* 23.2 (2001), pp. 442–462.

[61] T. P. Fries, H. G. Matthies, et al. “A review of Petrov–Galerkin stabilization approaches and an extension to meshfree methods”. In: *Technische Universitat Braunschweig, Brunswick* (2004).

[62] A. George. “Nested Dissection of a Regular Finite Element Mesh”. In: *SIAM Journal on Numerical Analysis* 10.2 (1973), pp. 345–363. DOI: 10.1137/0710032. eprint: https://doi.org/10.1137/0710032. URL: https://doi.org/10.1137/0710032.

[63] A George and D. R. McIntyre. “On the Application of the Minimum Degree Algorithm to Finite Element Systems”. In: *SIAM Journal on Numerical Analysis* 15.1 (1978), pp. 90–112. DOI: 10.1137/0715006. eprint: https://doi.org/10.1137/0715006. URL: https://doi.org/10.1137/0715006.

[64] J. A. George. “Computer Implementation of the Finite Element Method”. AAI7205916. PhD thesis. Stanford, CA, USA: Stanford University, 1971.

[65] U. Ghia, K. N. Ghia, and C. T. Shin. “High-Resolutions for incompressible flows Using the Navier-Stokes equations and a multigrid method”. In: *Journal of Computational Physics* 48 (1982), pp. 387–411.

[66] V. Girault and P. A. Raviart. *Finite element methods for Navier-Stokes equations: theory and algorithms.* Vol. 5. Springer Science & Business Media, 2012.

[67] A. Greenbaum, V Ptak, and Z. Strakos. “Any Nonincreasing Convergence Curve is Possible for GMRES”. In: *SIAM Journal on Matrix Analysis and Applications* 17.3 (1996), pp. 465–469. DOI: 10.1137/S0895479894275030. eprint: https://doi.org/10.1137/S0895479894275030. URL: https://doi.org/10.1137/S0895479894275030.

[68] I. Gustafsson. “A class of first order factorization methods”. In: *BIT Numerical Mathematics* 18.2 (1978), pp. 142–156.

[69] W. Hackbusch. *Iterative solution of large sparse systems of equations.* Vol. 40. Springer, 1994.

[70] W. Hackbusch. *Multi-Grid Methods and Applications.* ISBN: 0-387-12761-5. Berlin: Springer-Verlag, 1985.

[71] W. Hackbusch. *Multi-Grid Methods and Applications.* Springer, 2003.

[72] W. Hackbusch and T. Probst. “Downwind Gauß-Seidel Smoothing for Convection Dominated Problems”. In: *Numerical Linear Algebra with Applications* 4.2 (1997), pp. 85–102.

[73] M. R. Hestenes and E. Stiefel. “Methods of Conjugate Gradients for Solving Linear Systems”. In: *Journal of Research of the National Bureau of Standards* 49.6 (1952), pp. 409–436.

[74] R. A. Horn and C. R. Johnson. *Matrix analysis.* Cambridge university press, 1990.

[75] Y. Huang and H. A. van der Vorst. *Some Observations on the Convergence Behavior of GMRES.* Tech. rep. Delft Univ. Technology, 1989.

[76] T. Hughes and A.N. Brooks. “A multidimensional upwinding scheme with no crosswind diffusion”. In: *Finite Element Methods for Convection Dominated Flows* 34 (1979).

[77] S. Hussain. “Numerical analysis of new class of higher order Galerkin time discretization schemes for nonstationary incompressible flow problems”. PhD thesis. Universitätsbibliothek Dortmund, 2012.

[78] I. C. F. Ipsen and C. D. Meyer. "The Idea behind Krylov Methods". In: *The American Mathematical Monthly* 105.10 (1998), pp. 889–899. ISSN: 00029890, 19300972. URL: `http://www.jstor.org/stable/2589281`.

[79] B. M. Irons. "A frontal solution program for finite element analysis". In: *International Journal for Numerical Methods in Engineering* 2.1 (1970), pp. 5–32. ISSN: 1097-0207. DOI: `10.1002/nme.1620020104`. URL: `http://dx.doi.org/10.1002/nme.1620020104`.

[80] V. John. "Higher order finite element methods and multigrid solvers in a benchmark problem for the 3D Navier–Stokes equations". In: *International Journal for Numerical Methods in Fluids* 40.6 (2002), pp. 775–798.

[81] C. Johnson. *Numerical solution of partial differential equations by the finite element method*. Cambridge Uni. Press, 1994.

[82] M. Köster. "A hierarchical flow solver for optimisation with PDE constraints". PhD thesis. Universitätsbibliothek Dortmund, 2011.

[83] D. Kuzmin. "A guide to numerical methods for transport equations". In: *University Erlangen-Nuremberg* (2010).

[84] C. Lanczos. "Solution of systems of linear equations by minimized iterations". In: *Journal of Research of the National Bureau of Standards* 49 (1952), pp. 33–53.

[85] S. Link et al. "Anisotropic diffusion of elongated and aligned polymer chains in a nematic solvent". In: *The Journal of Physical Chemistry B* 110.40 (2006), pp. 19799–19803.

[86] J. W. H. Liu. "The Multifrontal Method for Sparse Matrix Solution: Theory and Practice". In: *SIAM Review* 34.1 (1992), pp. 82–109.

[87] J. W. H. Liu and A. H. Sherman. "Comparative Analysis of the Cuthill-McKee and the Reverse Cuthill-McKee Ordering Algorithms for Sparse Matrices". In: *SIAM Journal on Numerical Analysis* 13.2 (1976), pp. 198–213. ISSN: 00361429. URL: `http://www.jstor.org/stable/2156087`.

[88] R. Löhner. *Applied computational fluid dynamics techniques: an introduction based on finite element methods*. John Wiley & Sons, 2008.

[89] D. G. Luenberger. *Introduction to Linear and Nonlinear Programming.* Newyork: Addison-Wesley, 1973.

[90] J. Mandel. "Balancing domain decomposition". In: *International Journal for Numerical Methods in Biomedical Engineering* 9.3 (1993), pp. 233–241.

[91] J. Mandel and M. Brezina. "Balancing domain decomposition for problems with large jumps in coefficients". In: *Mathematics of Computation of the American Mathematical Society* 65.216 (1996), pp. 1387–1401.

[92] J. Mandel and C. R. Dohrmann. "Convergence of a balancing domain decomposition by constraints and energy minimization". In: *Numerical linear algebra with applications* 10.7 (2003), pp. 639–659.

[93] T. A. Manteuffel. "An incomplete factorization technique for positive definite linear systems". In: *Mathematics of computation* 34.150 (1980), pp. 473–497.

[94] D. J. Mavriplis. “Multigrid strategies for viscous flow solvers on anisotropic unstructured meshes”. In: *Journal of Computational Physics* 145.1 (1998), pp. 141–165.

[95] J. Meijerink and H. A. van der Vorst. “An iterative solution method for linear systems of which the coefficient matrix is a symmetric M-matrix”. In: *Mathematics of computation* 31.137 (1977), pp. 148–162.

[96] A. Meister and C. Vömel. “Efficient preconditioning of linear systems arising from the discretization of hyperbolic conservation laws”. In: *Advances in Computational Mathematics* 14.1 (Jan. 2001), pp. 49–73. ISSN: 1572-9044. DOI: `10.1023/A:1016645505973`. URL: `https://doi.org/10.1023/A:1016645505973`.

[97] R. B. Morgan. “A restarted GMRES method augmented with eigenvectors”. In: *SIAM Journal on Matrix Analysis and Applications* 16.4 (1995), pp. 1154–1171.

[98] R. B. Morgan. “GMRES with deflated restarting”. In: *SIAM Journal on Scientific Computing* 24.1 (2002), pp. 20–37.

[99] R. B. Morgan. “Implicitly restarted GMRES and Arnoldi methods for nonsymmetric systems of equations”. In: *SIAM Journal on Matrix Analysis and Applications* 21.4 (2000), pp. 1112–1135.

[100] W. A. Mulder. “A high-resolution Euler solver based on multigrid, semi-coarsening, and defect correction”. In: *Journal of Computational Physics* 100.1 (1992), pp. 91–104.

[101] W. A. Mulder. “A new multigrid approach to convection problems”. In: *Journal of Computational Physics* 83.2 (1989), pp. 303–323.

[102] M. Müller. “Adaptive high-resolution finite element schemes.” PhD thesis. Technische Universität Dortmund, 2010.

[103] M. Müller. *Lecture Notes: Mathematical and practical aspects of finite elements*. 2010.

[104] N. Munksgaard. “Solving sparse symmetric sets of linear equations by preconditioned conjugate gradients”. In: *ACM Transactions on Mathematical Software (TOMS)* 6.2 (1980), pp. 206–219.

[105] R. Nabben and C. Vuik. “A comparison of deflation and coarse grid correction applied to porous media flow”. In: *SIAM Journal on Numerical Analysis* 42.4 (2004), pp. 1631–1647.

[106] R. Nabben and C. Vuik. “A Comparison of Deflation and the Balancing Preconditioner”. In: *SIAM Journal on Scientific Computing* 27.5 (2006), pp. 1742–1759.

[107] G. Nabh. *On high order methods for the stationary incompressible Navier–Stokes equations*. Interdisziplinäres Zentrum für Wiss. Rechnen der Univ. Heidelberg, 1998.

[108] R. A. Nicolaides. “Deflation of conjugate gradients with applications to boundary value problems”. In: *SIAM Journal on Numerical Analysis* 24.2 (1987), pp. 355–365.

[109] A. Ouazzi. “Finite element simulation of nonlinear fluids with application to granular material and powder”. PhD thesis. TU Dortmund, Dec. 2005.

[110] L. Pauli and M. Behr. “On stabilized space-time FEM for anisotropic meshes: Incompressible Navier–Stokes equations and applications to blood flow in medical devices”. In: *International Journal for Numerical Methods in Fluids* 85.3 (2017), pp. 189–209.

[111] A. Quarteroni and A. Valli. *Domain decomposition methods for partial differential equations.* Oxford University Press, 1999.

[112] R. Rannacher and S. Turek. “Simple nonconforming quadrilateral Stokes element”. In: *Numerical Methods for Partial Differential Equations* 8.2 (1992), pp. 97–111. ISSN: 1098-2426. DOI: `10.1002/num.1690080202`. URL: `http://dx.doi.org/10.1002/num.1690080202`.

[113] D. Boffi and L. Gastaldi. “On the Quadrilateral Q_2P_1 Element for the Stokes Problem”. In: *Int. J. Numer. Meth. Fluids.* 39 (2002), pp. 1001–1011.

[114] M. Renardy and R. C. Rogers. *An introduction to partial differential equations.* Vol. 13. Springer Science & Business Media, 2006.

[115] I. Rocha et al. “Experimental/numerical study of anisotropic water diffusion in glass/epoxy composites”. In: *IOP Conference Series: Materials Science and Engineering.* Vol. 139. IOP Publishing. 2016.

[116] J. W. Ruge and K. Stüben. “Algebraic Multigrid”. In: *Multigrid Methods.* Philadelphia: SIAM, 1987. Chap. 4, pp. 73–130. DOI: `10.1137/1.9781611971057.ch4`. eprint: `http://locus.siam.org/doi/pdf/10.1137/1.9781611971057.ch4`. URL: `http://locus.siam.org/doi/abs/10.1137/1.9781611971057.ch4`.

[117] Y. Saad. “ILUT: A dual threshold incomplete LU factorization”. In: *Numerical linear algebra with applications* 1.4 (1994), pp. 387–402.

[118] Y. Saad. *Iterative Methods for Sparse Linear Systems.* 2nd. SIAM, 2003.

[119] Y. Saad. *Numerical methods for large eigenvalue problems.* Manchester University Press, 1992.

[120] Y. Saad. “Preconditioned Krylov subspace methods for CFD applications”. In: *Solution Techniques for Large-Scale CFD Problems* (1995), pp. 139–158.

[121] Y. Saad and M. H. Schultz. “GMRES: A Generalized Minimal Residual Algorithm for Solving Nonsymmetric Linear Systems”. In: *SIAM Journal on Scientific and Statistical Computing* 7.3 (1986), pp. 856–869. DOI: `10.1137/0907058`. URL: `https://doi.org/10.1137/0907058`.

[122] Y. Saad and H. A. van der Vorst. “Iterative solution of linear systems in the 20th century”. In: *Numerical Analysis: Historical Developments in the 20th Century.* Elsevier, 2001, pp. 175–207.

[123] K. D. Santhosh, K. K. Suresh, and D. M. Kumar. “A fine grid solution for a lid-driven cavity flow using multigrid method”. In: *Engineering Applications of Computational Fluid Mechanics* 3.3 (2009), pp. 336–354.

[124] M. Schäfer et al. “Benchmark computations of laminar flow around a cylinder”. In: *Flow simulation with high-performance computers II.* Springer, 1996, pp. 547–566.

[125] Ir. A. Segal. *Lecture notes in Finite element methods for the incompressible Navier-Stokes equations. J.M. Burgerscentrum, Delft Institute of Applied Mathematics, TU Delft.* URL: `http://ta.twi.tudelft.nl/nw/users/vermolen/SpecialTopics/fem_notes.pdf`. Last visited on 09.03.2018. 2017.

[126] P. Sonneveld. "CGS, A Fast Lanczos-Type Solver for Nonsymmetric Linear systems". In: *SIAM Journal on Scientific and Statistical Computing* 10.1 (1989), pp. 36–52. DOI: `10.1137/0910004`. eprint: `https://doi.org/10.1137/0910004`. URL: `https://doi.org/10.1137/0910004`.

[127] G. Strang and G. J. Fix. *An analysis of the finite element method.* Vol. 212. Prentice-hall Englewood Cliffs, NJ, 1973.

[128] K. Stüben. "A review of algebraic multigrid". In: *Journal of Computational and Applied Mathematics* 128.1 (2001). Numerical Analysis 2000. Vol. VII: Partial Differential Equations, pp. 281–309. ISSN: 0377-0427. DOI: `https://doi.org/10.1016/S0377-0427(00)00516-1`. URL: `http://www.sciencedirect.com/science/article/pii/S0377042700005161`.

[129] A. Toselli and O. B. Widlund. *Domain decomposition methods: algorithms and theory.* Vol. 34. Springer, 2005.

[130] L. N. Trefethen and D. Bau. *Numerical Linear Algebra.* SIAM, 1997.

[131] U. Trottenberg, C. W. Oosterlee, and A. Schuller. *Multigrid.* Academic press, 2000.

[132] S. Turek. *Efficient Solvers for Incompressible Flow Problems: An Algorithmic and Computational Approach.* Springer-Verlag, 1999.

[133] S. Turek. "On ordering strategies in a multigrid algorithm". In: *Proc. 8th GAMM-Seminar.* Vol. 41. Notes on Numerical Fluid Mechanics. Vieweg, 1992.

[134] S. Turek and A. Ouazzi. "Unified edge-oriented stabilization of nonconforming FEM for incompressible flow problems: Numerical investigations". In: *Journal of Numerical Mathematics* 15.4 (2007), pp. 299–322.

[135] S. Turek, A. Ouazzi, and J. Hron. "On pressure separation algorithms (PSEPA) for improving the accuracy of incompressible flow simulations". In: *International Journal for Numerical Methods in Fluids* 59.4 (2008), pp. 387–403.

[136] S. Turek and R. Schmachtel. "Fully coupled and operator-splitting approaches for natural convection flows in enclosures". In: *International Journal for Numerical Methods in Fluids* 40 (2002), pp. 1109–1119.

[137] A. Van der Sluis and H. A. van der Vorst. "The rate of convergence of conjugate gradients". In: *Numerische Mathematik* 48.5 (1986), pp. 543–560.

[138] S.P. Vanka. "Block-implicit Multigrid Solutions of Navier-Stokes Equations in Primitive Variables". In: *J. of Comp. Phys.* 65 (1986), pp. 138–158.

[139] R. S. Varga. *Factorization and normalized iterative methods.* Tech. rep. Westinghouse Electric Corp. Bettis Plant, Pittsburgh, 1959.

[140] R. S. Varga. *Matrix iterative analysis.* Prentice Hall, Engelwood Cliffs NJ., 1962.

[141] R. Verfürth. "A multilevel algorithm for mixed problems". In: *SIAM journal on numerical analysis* 21.2 (1984), pp. 264–271.

[142] R. Verfürth. *A review of a posteriori error estimation and adaptive mesh-refinement techniques.* John Wiley & Sons Inc, 1996.

[143] H. A. van der Vorst. "Bi-CGSTAB: A Fast and Smoothly Converging Variant of Bi-CG for the Solution of Nonsymmetric Linear Systems". In: *SIAM Journal on Scientific and Statistical Computing* 13.2 (1992), pp. 631–644. DOI: `10.1137/0913035`. eprint: `https://doi.org/10.1137/0913035`. URL: `https://doi.org/10.1137/0913035`.

[144] H. A. van der Vorst and C. Vuik. "The superlinear convergence behaviour of GMRES". In: *Journal of Computational and Applied Mathematics* 48.3 (1993), pp. 327–341. DOI: `http://dx.doi.org/10.1016/0377-0427(93)90028-A`. URL: `http://www.sciencedirect.com/science/article/pii/037704279390028A`.

[145] P. Wesseling. *An introduction to multigrid methods.* Pure and applied mathematics. John Wiley & Sons Australia, Limited, 1992. ISBN: 9780471930839. URL: `https://books.google.de/books?id=MznvAAAAMAAJ`.

[146] P. Wesseling and C. W. Oosterlee. "Geometric multigrid with applications to computational dynamics". In: *Journal of Computational and Applied Mathematics* 128.2 (2001), pp. 311–334.

[147] J. H. Wilkinson. *The algebraic eigenvalue problem.* Vol. 87. Clarendon Press Oxford, 1965.

[148] G. Wittum. "Multi-grid methods for Stokes and Navier-Stokes equations". In: *Numerische Mathematik* 54.5 (1989), pp. 543–563.

[149] D. M. Young. *Iterative solution of large linear systems.* Academic Press, New York, 1971.

[150] W. Zulehner. "Analysis of iterative methods for saddle point problems: a unified approach". In: *Mathematics of computation* 71.238 (2002), pp. 479–505.