EFFICIENT "BLACK-BOX" MULTIGRID SOLVERS FOR CONVECTION-DOMINATED PROBLEMS

A thesis submitted to the University of Manchester
for the degree of Doctor of Philosophy
in the Faculty of Engineering and Physical Science

2011

Glyn Owen Rees
School of Computer Science

Contents

(Word count: 43,408)

List of Tables

List of Figures

Efficient "black-box" multigrid solvers for convection-dominated problems,

by Glyn Owen Rees.

A thesis submitted to the University of Manchester for the degree of Doctor of Philosophy,

August 2011.

The main objective of this project is to develop a "black-box" multigrid preconditioner for the iterative solution of finite element discretisations of the convection-diffusion equation with dominant convection. This equation can be considered a stand alone scalar problem or as part of a more complex system of partial differential equations, such as the Navier-Stokes equations. The project will focus on the stand alone scalar problem.

Multigrid is considered an optimal preconditioner for scalar elliptic problems. This strategy can also be used for convection-diffusion problems, however an appropriate robust smoother needs to be developed to achieve mesh-independent convergence. The focus of the thesis is on the development of such a smoother. In this context a novel smoother is developed referred to as truncated incomplete factorisation (tILU) smoother. In terms of computational complexity and memory requirements, the smoother is considerably less expensive than the standard ILU(0) smoother. At the same time, it exhibits the same robustness as ILU(0) with respect to the problem and discretisation parameters. The new smoother significantly outperforms the standard damped Jacobi smoother and is a competitor to the Gauss-Seidel smoother (and in a number of important cases tILU outperforms the Gauss-Seidel smoother). The new smoother depends on a single parameter (the truncation ratio). The project obtains a default value for this parameter and demonstrated the robust performance of the smoother on a broad range of problems. Therefore, the new smoothing method can be regarded as "black-box". Furthermore, the new smoother does not require any particular ordering of the nodes, which is a prerequisite for many robust smoothers developed for convection-dominated convection-diffusion problems.

To test the effectiveness of the preconditioning methodology, we consider a number of model problems (in both 2D and 3D) including uniform and complex (recirculating) convection fields discretised by uniform, stretched and adaptively refined grids. The new multigrid preconditioner within block preconditioning of the Navier-Stokes equations was also tested. The numerical results gained during the investigation confirm that tILU is a scalable, robust smoother for both geometric and algebraic multigrid. Also, comprehensive tests show that the tILU smoother is a competitive method.

Declaration

No portion of the work referred to in this thesis has been submitted in support of an application for another degree or qualification of this or any other university or other institute of learning.

Copyright

Acknowledgement

I would like to thank my principal supervisor Milan Mihajlović for his continual advice, support and tolerating my spelling of (scaler/scalar). I would also like to thank my second supervisor David Silvester for his assistance and guidance during the project. Matthias Heil and Andrew Hazil for providing the software framework "Oomph-lib" as a test-bed for developing and testing the new solution strategies. I would also like to thank the "HEC" EPSRC, for funding the research.

A special thanks must go to my partner Rachel Ryder for her love and support, putting up with the inconsistent working hours as well as the duration of the research. Also, Peter Ryder for all of his hard work and effort. Finally, my mother and father for the love and support they have given to me all through my life.

Chapter 1

Introduction

The convection-diffusion equation is used in flow modelling. Ocean modelling, flow through a porous media, ground water pollution, air pollution, river dynamics or sediment transport [42, p.133] are some examples where the convection-diffusion equation provides an important part of the mathematical model for these types of flows. For fluid mechanical engineering Wesseling [91] suggests that static diffusion, anisotropic diffusion and convection-diffusion are key test problems.

In the project we use the finite element (FE) method to discretise the convection-diffusion equation. The resulting sparse system of linear equations is then iteratively solved by a Krylov subspace method preconditioned by multigrid. The main focus in the thesis is on the development of an efficient multigrid preconditioner for the discrete problem.

The convection-diffusion equation involves two parameters that can be manipulated to form a plethora of interesting and physically relevant instances (cases/ scenarios). The parameters that influence the convection-diffusion problem are the relative strength of convection and the structure of the convection field (including uni-directional, recirculating (vortex) and multi-vortices). An increase in the spacial dimension of the problem also has an impact to the properties of the resulting discrete problem and thus the solution methods. These are referred to as problem parameters. A further consideration is that the sparse system of linear equations is also influenced by discretisation parameters (the mesh spacing). Within the thesis we refer to discretisation parameters as: the level of grid refinement (problem size) and type of grid refinement (uniform, stretched and adaptively refined grids). With such a wealth of parameters, the multigrid preconditioned Krylov solver can give poor performance (e.g converge, if at all, in a large iteration count, high storage requirements and/or large execution time).

Throughout the thesis an iterative solver is referred to as "robust" if its performance is consistent (based on the numbers of iterations), irrespective of the problem

size, the structure of convection field and the relative strength of convection. As all these conditions are difficult to achieve simultaneously in practice we will define robustness with respect to individual parameters; we consider h-robustness as consistent performance with respect to the problem size (h), Pe-robustness as consistent performance with respect to the relative strength of convection and $\vec{w}$-robustness as consistent performance with respect to the convection fields. Also, if the storage requirements and execution time of an iterative solver scale linearly ($O(n)$) with respect to the problem size (n), the iterative solver is referred to as optimal. A "black-box" preconditioner, within the context of the project, is a method for accelerating the convergence of Krylov subspace iterative methods, that either has no tuning parameter or uses a set of default parameters that make the solver robust over a wide range of problems and discretisations.

Simple point iterative methods are not robust in general as they fail to reduce consistently the entire Fourier spectrum (frequencies) of the solution error. Multigrid (MG) is a method that is made up of a nested sequence of progressively finer grids (MG hierarchy). One important component of MG is a method of approximating the discrete problem at each level. To keep computational costs low simple point iterative methods (smoothers) are commonly used for that purpose. A smoother is referred to as robust if its use results in a robust solution strategy.

"A good convergence with a small Krylov subspace means, in practice, that the multigrid preconditioner must be as 'good as possible'. It is, for example, not possible to get a convergence acceleration from a small subspace if a smoother is used that does not reduce a large number of frequencies well" [70].

To achieve a more efficient MG preconditioner we develop a new, robust and computationally effective smoother (with good parallel scalability) suitable for convection-dominated problems. A robust optimal solver strategy for the discrete diffusion problem is MG preconditioned Conjugate Gradient Krylov method. Due to the symmetric positive definite (SPD) properties of the linear system, simple (point) smoothers such as damped Jacobi or Gauss-Seidel are robust. However for the discrete convection-diffusion problems, especially for large convection strength and/or complex convection fields, standard MG methods are not robust. Research into the development of robust smoothing techniques for MG preconditioning of the convection-diffusion equation attracts considerable interest. However, the main issues with robust smoothers for convection-dominated problems currently available in the literature is that they usually require explicit knowledge of geometric information, are frequently difficult to implement than standard point smoothers and are more expensive in terms of computational time and storage costs. This adversely affects the performance of execution (poor caching, poor parallel scalability).

The project will focus on the ILU(0) smoother and attempt to reduce its computational cost (which is considerable, as ILU is regarded as one of the most expensive smoothers) while at the same time trying to retain its robustness. In this context a modification of the standard ILU(0) smoother is developed, based on static truncation of the non-zero entries in the coefficient matrices within the MG hierarchy. In the convection-diffusion case the truncation criterion can also pick local properties of the convection field which may be missed by the standard point smoothers. Also, as the coefficients of the matrix are h-dependent (see Section 2.4) the new method results in a variable smoother, where typically more entries are truncated at the finer grids. This makes the smoother computationally effective, in some cases even reducing to a damped Jacobi smoother. The new truncated ILU(0) smoother is referred to as tILU.

In the thesis we perform Fourier analysis of the new smoothing methodology for a special case (uni-directional wind). Also, we examine the solver strategy's efficiency and parallel performance for both geometric (GMG) and algebraic (AMG) multigrid on a range of physically important flow problems in both two and three spacial dimensions.

The numerical results show that tILU is a superior smoother with regards to storage cost, compared to ILU(0), as well as being competitive with respect to the execution time to Gauss-Seidel smoother. Parallel scaling of the AMG method with tILU smoother is comparable to that of using a damped Jacobi smoother (which is inherently parallel) and significantly better than using Gauss-Seidel as a smoother. In summary the numerical results obtained during the investigation confirm that tILU is a scalable, robust and an effective smoother. Furthermore, the results shows that unlike Gauss-Seidel, effectiveness of the tILU smoother is not significantly affected by ordering of the unknowns. Thus, it can also be regarded as a "black-box" method.

A comprehensive set of numerical results showing the robustness of tILU, can be found in [74].

Chapter 2

The Convection-Diffusion Problem

The physical process of diffusion is the model of a slow isotropic change of a certain physical quantity. Convection describes the physical process of directional change governed by existence of the convection field. The combination of these two processes are modelled mathematically by a convection-diffusion equation. The flow of a viscous fluid under an external force, describes the overall process of a convection-diffusion problem. An external force may be regarded as mixing or thermal effects that bring about convection [16] [75, p.1.4].

A convection-diffusion equation is an instance of a partial differential equation (PDE). These equations can be solved analytically, in some cases, or numerically. Numerical modelling involves the development of algorithms and techniques, to compute an approximate solution of continuous problems. This is achieved by introducing some form of problem discretisation. As a consequence such discrete models can be simulated efficiently on modern computers.

This chapter gives an introduction into the convection-diffusion equation and describes the Galerkin approximation method for its discretisation. The discrete problem will be represented as a system of linear equations. Towards the end of the chapter we examine the properties of the discrete convection-diffusion operator.

The convection-diffusion equation appears either as a scalar problem or as a component part of a more complex multiphysics problems. Examples of multiphysics problems that include convection-diffusion as a component are; Navier-Stokes problems [31, p.313], Boussinesq problems [22, p.95, p.96] [93, p.235].

2.1 The continuous problem

The diffusion process can be very slow depending on the density of the material. The slow rate (time scale) of diffusion is relevant when it occurs concurrently with other phenomena, in our case the convection. When diffusion is the slowest process in the

overall combination, it limits the overall rate of the process. In practice this can result in limits to the efficiency of commercial distillation, the speed with which an acid and a base will react and the rate of corrosion in steel [16, Chapter 1]. If a numerical simulation of a composite process which includes diffusion is considered, then the slow characteristic of the diffusion process may limit the size of the time step produced by the adaptive time-stepping algorithm, for example [40], thus increasing the overall computational cost. The convection process represents the bulk directional flow of a physical quantity.

Definition 2.1.1 (Continuous function [79])**.** A real function f is mapping which assigns a unique real number to every point in a domain, $f : \Omega \to \mathbb{R}$

- $\mathcal{C}^0(\Omega)$ is the set of all continuous functions defined on Ω.
- $\mathcal{C}^1(\Omega)$ is the set of all continuous functions whose $1st$ derivatives are also continuous over Ω.
- $\mathcal{C}^2(\Omega)$ is the set of all continuous functions whose $1st$ and $2nd$ derivatives are also continuous over Ω.

The classical formulation of a convection-diffusion equation is:
Find $u \in \mathcal{C}^2(\Omega)$ such that

$$-\epsilon\nabla^2 u + \vec{w} \cdot \nabla u = f \qquad \text{in } \Omega \subset \mathbb{R}^d. \tag{2.1}$$

In (2.1) the variable ($d = 2\,(3)$) represents the spacial dimension, $\nabla^2 \equiv \nabla \cdot (\nabla) = \frac{\partial^2}{\partial x^2} + \frac{\partial^2}{\partial y^2} + (\frac{\partial^2}{\partial z^2})$ the Laplace operator, and $\nabla \equiv (\frac{\partial}{\partial x}, \frac{\partial}{\partial y}, (\frac{\partial}{\partial z}))^T$ the gradient operator. A vector-value function $\vec{w} : \mathbb{R}^d \mapsto \mathbb{R}^d$ with $\vec{w} = (w_x, w_y, (w_z))$ is referred to as the convection field, or the "wind", and defines the direction of the convection. The right hand side term f is the external forcing term. The parameter variable $\epsilon > 0$, represents the viscosity or diffusivity.

If (2.1) is posed with no boundary conditions, it has infinitely many solutions. For a problem, such as (2.1) these solutions represent a family of functions that differ by an additive constant. A particular solution from that family is selected by fixing its values at the boundary $\partial\Omega$ of Ω [31, p.10]. The most general form of boundary conditions (BCs) suitable for (2.1) is of the form

$$\beta_1 u + \beta_2 \frac{\partial u}{\partial \widehat{n}} = g \quad \text{on} \quad \partial\Omega, \tag{2.2}$$

where $\frac{\partial u}{\partial \widehat{n}}$ denotes the normal derivative (the flux) of u, i.e the derivative with respect to the outward pointing normal on $\partial\Omega$.

A boundary value problem (BVP) consists of two equations (2.1) and (2.2). The solution u that satisfies the BVP (2.1)–(2.2) is referred to as the classical solution [31, p.14], given the domain boundary is a sufficiently smooth function. The general type BC (2.2) allows a number of different combinations of BCs to be introduced into the model. Dirichlet boundary conditions are introduced when $\beta_2 = 0$. On the other hand, when $\beta_1 = 0$ the boundary conditions are referred to as pure Neumann. In cases when $\beta_1 = 0$ the problem does not have a unique solution, (it is unique up to an additive constant) [31, p.10]. A more complex BC (2.2) is when the boundary is divided into two disjoint parts. One part Neumann, denoted by $\partial\Omega_N$, and the remaining boundary, Dirichlet, denoted by $\partial\Omega_D$. The whole boundary is now represented by $\partial\Omega_D \bigcup \partial\Omega_N = \partial\Omega$ with $\partial\Omega_D \bigcap \partial\Omega_N = \emptyset$. Finally the Dirichlet part of the boundary must be of finite length $\int_{\partial\Omega_D} ds \neq 0$, to ensure a unique solution. Explicit BCs for this case are

$$u = g_D \quad \text{on} \quad \partial\Omega_D\,, \quad \frac{\partial u}{\partial \widehat{n}} = g_N \quad \text{on} \quad \partial\Omega_N. \tag{2.3}$$

There are two possible cases of the computational domain: open domain and closed domain. In the former case (open domain) only part of a larger domain is modelled, concentrating on some interesting features of the solution. The characteristics of this case is that not all of the domain boundaries are rigid boundaries. An example of such a domain can be encountered in flow-through problems, where one part of the boundary represents an inflow and (to satisfy the incompressibility condition) another part of the boundary represents the outflow boundary [31, p.117]. In the latter case we have an enclosed domain. An example of such a domain can be encountered in enclosed flow problems, such as flow inside a cavity [31, p.119].

In the case of a convection-diffusion problem we can distinguish three different types of boundary, depending on the relative positions of the vectors $\vec{w}$ and $\widehat{n}$ (see Figure 2.1) [31, p.113][32, p.455]. In particular

$$\begin{aligned} &Outflow\,boundary: \ \partial\Omega_+ = \{x \in \partial\Omega \,|\, \vec{w}\cdot\widehat{n} > 0\}, \\ &Characteristic\,boundary: \ \partial\Omega_0 = \{x \in \partial\Omega \,|\, \vec{w}\cdot\widehat{n} = 0\}, \\ &Inflow\,boundary: \ \partial\Omega_- = \{x \in \partial\Omega \,|\, \vec{w}\cdot\widehat{n} < 0\}. \end{aligned} \tag{2.4}$$

To obtain a non-dimensional quantity that gives a relative measure of the convection and the diffusion in the problem, the convection-diffusion equation (2.1) is normalised. By normalising the convection-diffusion equation with respect to the domain of characteristic size L, the diffusivity parameter ϵ and the magnitude of the wind W ($\vec{w} = W\vec{w}_*$, given $W \geq 0$ and a unit wind vector $||\vec{w}_*||_1 = 1$), gives [31,

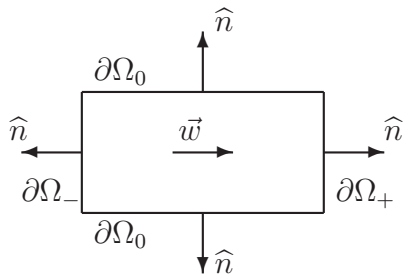


Figure 2.1: Flow through a domain showing three different types of boundaries, relative to the wind $\vec{w}$, corresponding to (2.4).

p.115]

$$-\nabla^2 u(\frac{x}{L},\frac{y}{L}) + (\frac{WL}{\epsilon})\vec{w_*}\cdot\nabla u(\frac{x}{L},\frac{y}{L}) = \frac{L^2}{\epsilon}f. \tag{2.5}$$

In equation (2.5) there is only one positive non-dimensional quantity,

$$Pe = \frac{WL}{\epsilon} > 0, \tag{2.6}$$

referred to as the Peclet number.

We now restate the dimensional version of the BVP problem (2.1)–(2.2) with $\beta_2 = 0$: Find $u \in \mathcal{C}^2(\Omega)$ such that

$$-\epsilon\nabla^2 u + \vec{w}\cdot\nabla u = f \qquad \text{in } \Omega \subset \mathbb{R}^d, \qquad u = g \quad \text{on} \quad \partial\Omega. \tag{2.7}$$

A scalar convection-diffusion equation represents a singular perturbation of the second-order scalar elliptic problem (the Poisson problem). Singular perturbation of the second-order derivative is when there are lower order terms and a small parameter that at its limit changes the characteristics of the problem. In the case of the convection-diffusion equation, an elliptic problem changes into a hyperbolic one as $\epsilon \to 0$ ($Pe \to \infty$) [76, pp.xiii–xvi].

The convection component of the problem is represented by the term $(\vec{w}\cdot\nabla u)$ and the diffusion component by $(\nabla^2 u)$. In this work we restrict ourselves to modelling the convection-diffusion problems of cases that consist of slow moving fluids, i.e fluids that have a velocity much smaller that the speed of sound ($Mach\,number \ll 1$) [42]. This implies the incompressibility of the convection field, which can be expressed as $\nabla\cdot\vec{w} = 0$ (for the case of Stokes and Navier-Stokes problems this is the mathematical expression of the mass conservation law)[42] [31, p.214]. For a scalar convection-diffusion problem the convection flow $\vec{w}$ is predefined (but assumed incompressible).

The unknown function u can represent the temperature of a fluid moving along a boundary such as a heated wall, or the transportation of a pollutant concentration moving down a river with velocity $\vec{w}$ subject to diffusion [66]. Also, a well known

convection-diffusion model is the movement of a concentration of smoke in the surrounding area coming out of a chimney subject to the external wind governed by $\vec{w}$ [73].

2.2 The Galerkin approximation method

The Galerkin method is an example of a general approximation procedure for the solution of PDEs. To use this method a continuous problem (a PDE) must be replaced by its integral counterpart (referred to as the *weak formulation*) obtained by introducing a space of test functions.

The approximate solution of the weak problem (the weak solution) is sought in a finite-dimensional space, with the solution accuracy correlated to the dimension of this space. Depending on the choice of the solution (trial) space and the space of test functions, there are different instances of the Galerkin method. For example the spectral element method [58], the finite element method (FEM) or the finite volume method (FVM) [77, Section 2.5]. The difference between these methods is based on the support for the basis set.

To derive a weak formulation of the BVP (2.7), we need to define the space of test functions and the trial space. A Sobolev space $H^1(\Omega)$ is defined as [31, p.16]

$$H^1(\Omega) = \{u \,|\, u, \nabla u \in L_2(\Omega)\}\,,$$

where $L_2(\Omega)$ is the space of square integrable functions u:

$$L_2(\Omega) = \left\{u \,|\, \int_\Omega u^2 \, d\Omega < \infty\right\}.$$

Thus, the Sobolev space $H^1(\Omega)$ is the space of functions defined on Ω which are, together with their first partial derivatives, square integrable. Test and trial spaces in the variational form of (2.7) will be suitable subspaces of the space $H^1(\Omega)$. The test space $H_0^1(\Omega)$ is a subspace of $H^1(\Omega)$, whose elements satisfy the homogeneous Dirichlet BCs on $\partial\Omega$:

$$H_0^1(\Omega) = \left\{v \,|\, v \in H^1(\Omega)\ ,\ v = 0 \text{ on } \partial\Omega\right\}.$$

The trial space $H_D^1(\Omega)$ is also a subspace of $H^1(\Omega)$, whose elements satisfy the same Dirichlet BCs as in the classical solution (2.7):

$$H_D^1(\Omega) = \left\{u \,|\, u \in H^1(\Omega)\ ,\ u = g \text{ on } \partial\Omega\right\}.$$

With the test and trial space selected, the first step in the Galerkin approximation method is to derive a weak formulation of (2.7). This is achieved by multiplying (2.7) by the test function v, from the test space $H_0^1(\Omega)$ and integrating over the domain Ω, i.e:

Find $u \in H_D^1(\Omega)$, such that

$$-\epsilon \int_\Omega \nabla^2 u v \, d\Omega + \int_\Omega (\vec{w} \cdot \nabla u) v \, d\Omega = \int_\Omega f v \, d\Omega, \qquad \forall v \in H_0^1(\Omega). \tag{2.8}$$

The condition $u \in H_D^1(\Omega)$ only guarantees that the integral in (2.8) are well-defined for functions and their first derivative in $L_2(\Omega)$. This is not fulfilled for the first integral on the left-hand side of (2.8). To get around this problem, Green's theorem [7, p.31] is applied to reduce the order of derivatives u, from a second order to a first order, at a consequence of increasing the smoothness requirement for v. Thus, we obtain

$$\begin{aligned} \epsilon \int_\Omega \nabla u \cdot \nabla v \, d\Omega - \epsilon \int_{\partial\Omega} \nabla u \cdot \widehat{n} \, v \, ds + \int_\Omega (\vec{w} \cdot \nabla u) v \, d\Omega \\ = \int_\Omega f v \, d\Omega, \qquad \forall v \in H_0^1(\Omega), \end{aligned} \tag{2.9}$$

where the second integral in (2.9) is a line integral along the boundary $\partial\Omega$.

The benefit of moving from a continuous form to a weak form is that, the weak form has less stringent requirements for the continuity of the solution than the classical problem (2.7) as, $\mathcal{C}^2(\Omega) \subset H^1(\Omega)$. If a classical solution $u \in \mathcal{C}^2(\Omega)$ to the problem (2.7) exists, then it is also a weak solution of (2.9). Conversely this may not always be the case [31, p.15], as the weak solution may not be smooth enough to be the classical solution. As the continuity requirements for the data in (2.9) are much less stringent than in (2.7) (by means $\mathcal{C}^2(\Omega) \subset H^1(\Omega)$), the weak form can allow treatment of certain physically relevant cases (for example [31, Example 1.1.4]) which do not have a classical solution [31, p.14].

By substituting ($\nabla u \cdot \widehat{n} = \frac{\partial u}{\partial \widehat{n}}$) into (2.2) and noting that $v \in H_0^1(\Omega)$ we obtain

$$\int_{\partial\Omega} \nabla u \cdot \widehat{n} v \, ds = \int_{\partial\Omega} \frac{\partial u}{\partial \widehat{n}} v \, ds = 0.$$

Rearranging (2.9): Find $u \in H_D^1(\Omega)$, such that

$$\epsilon \int_\Omega \nabla u \cdot \nabla v \, d\Omega + \int_\Omega (\vec{w} \cdot \nabla u) \; v \, d\Omega = \int_\Omega f v \, d\Omega, \qquad \forall v \in H_0^1(\Omega). \tag{2.10}$$

The Sobolev space $H^1(\Omega)$ is infinitely-dimensional, which mean that the weak problem (2.10) is continuous. A finite-dimensional representation of the problem

(2.10) can be achieved by taking a subspace $S^h(\Omega)$ of the Sobolev space $H^1(\Omega)$. That is the trial space $S_D^h \subset H_D^1(\Omega)$ and the test space $S_0^h \subset H_0^1(\Omega)$ can be characterised by their basis sets:

$$S_D^h = \text{span}\{\varphi_j\}_{j=1}^{N_I+N_D}, \quad S_0^h = \text{span}\{\varphi_i\}_{i=1}^{N_I}. \tag{2.11}$$

In (2.11) N_I is the number of basis functions in the interior of the domain, and N_D is the number of basis functions necessary to characterise Dirichlet boundary data.

By enforcing (2.10) over the finite-dimensional test and trial spaces, we obtain the discrete weak formulation of the convection-diffusion problem:

Find $u_h \in S_D^h \subset H_D^1$ satisfying

$$\begin{aligned} \epsilon \int_\Omega \nabla u_h \cdot \nabla v_h \, d\Omega + \int_\Omega (\vec{w} \cdot \nabla u_h) \, v_h \, d\Omega \\ = \int_\Omega f v_h \, d\Omega, \qquad \forall v_h \in S_0^h \subset H_0^1(\Omega). \end{aligned} \tag{2.12}$$

By substituting each test function $v_h = \{\varphi_i\}_{i=1}^{N_I}$ into (2.12) we obtain a system of N_I equations

$$\epsilon \int_\Omega \nabla u_h \cdot \nabla \varphi_i \, d\Omega + \int_\Omega (\vec{w} \cdot \nabla u_h) \, \varphi_i \, d\Omega = \int_\Omega f \varphi_i \, d\Omega, \qquad i = 1, ..., N_I. \tag{2.13}$$

From $u_h \in S_D^h$ it follows that u_h can be uniquely expressed as a linear combination of the basis set $u_h = \sum_{j=1}^{N_I+N_D} x_j \varphi_j$ (where $x_j \in \mathbb{R}$, $j = 1, ..., N_I$ are the unknown coefficients, and $j = N_I + 1, ..., N_I + N_D$ are the known coefficients that interpolate the Dirichlet boundary data, that determine Galerkin's approximation). Substituting this expression into (2.13), we obtain

$$\begin{aligned} \sum_{j=1}^{N_I+N_D} x_j \left[\epsilon \int_\Omega \nabla \varphi_j \cdot \nabla \varphi_i \, d\Omega + \int_\Omega (\vec{w} \cdot \nabla \varphi_j) \, \varphi_i \, d\Omega \right] \\ = \int_\Omega f \varphi_i \, d\Omega, \qquad i = 1, ..., N_I. \end{aligned} \tag{2.14}$$

By rearranging the terms in (2.14), so the boundary conditions are all on the right hand side

$$\vartheta_i = \sum_{j=N_I+1}^{N_I+N_D} (g)_j \left[\epsilon \int_\Omega \nabla \varphi_j \cdot \nabla \varphi_i \, d\Omega + \int_\Omega (\vec{w} \cdot \nabla \varphi_j) \, \varphi_i \, d\Omega \right], \tag{2.15}$$

we obtain

$$\boxed{\begin{aligned}\sum_{j=1}^{N_I} x_j \left[\epsilon \int_\Omega \nabla\varphi_j \cdot \nabla\varphi_i \, d\Omega + \int_\Omega (\vec{w} \cdot \nabla\varphi_j)\ \varphi_i \, d\Omega\right] \\ = \int_\Omega f\varphi_i \, d\Omega - \ \vartheta_i, \qquad i = 1, ..., N_I.\end{aligned}} \tag{2.16}$$

The equation (2.16) represents a system of N_I linear equations for the unknown coefficients $x_j\,(j = 1, ..., N_I)$. It can be rewritten in a matrix notation as

$$A\bar{x} = \bar{b}, \qquad \bar{b}, \bar{x} \in \mathbb{R}^{N_I},\ A \in \mathbb{R}^{N_I \times N_I}, \tag{2.17}$$

where A is a coefficient matrix that has two physical components $A = (\epsilon \cdot D + C)$: the discrete diffusion operator $D \in \mathbb{R}^{N_I \times N_I}$, and the discrete convection operator $C \in \mathbb{R}^{N_I \times N_I}$.

- The properties of a diffusion matrix. If the test and trial spaces have the same basis, the diffusion matrix D is symmetric and positive definite [31, p.18], and has the elements

$$D = [d_{ij}] = \int_\Omega \nabla\varphi_j \cdot \nabla\varphi_i \, d\Omega \qquad i, j = 1, ..., N_I. \tag{2.18}$$

- The properties of the convection matrix. If the test and trial spaces have the same basis (when $\nabla \cdot \vec{w} = 0$), the convection matrix C is skew-symmetric [31, p.120], and has the elements

$$C = [c_{ij}] = \int_\Omega (\vec{w} \cdot \nabla\varphi_j)\ \varphi_i \, d\Omega \qquad i, j = 1, ..., N_I. \tag{2.19}$$

By definition skew-symmetry means that the transpose of the matrix is equal to the negative of the initial matrix, for example

$$c_{ij} = \begin{cases} -c_{ji} & \text{if } i \neq j \\ 0 & \text{if } i = j. \end{cases}$$

The components of the right-hand side vector $\overline{b} \in \mathbb{R}^{N_I}$ are given by

$$b_i \ = \int_\Omega f\varphi_i \, dx - \vartheta_i \qquad i = 1, ..., N_I. \tag{2.20}$$

Finally, $\bar{x} \in \mathbb{R}^{N_I}$ is a vector of unknown coefficients that define the Galerkin solution.

2.2.1 The finite element method

The FEM is an instance of the Galerkin method where for the basis sets φ_i we adopt piecewise polynomials associated with the subdivision of Ω into disjoint subdomains Ω_i called the finite elements (FE). There are a number of different ways to perform the subdivision into FEs, that is, there is a trade-off between numerical accuracy of the approximation and computational efficiency.

We restrict ourselves to the case of quadrilateral elements and bilinear (Q_1) basis set. Each element will have four local basis functions associated to its four nodes (that is each basis function is equal to one, at one node, and zero at the remaining nodes).

To compute each element contribution to the global Galerkin matrix, a mapping is formed from the reference element $\square_* = [-1, 1]^2$ to the given element $\square_h$ shown in Figure 2.2 [31, pp.22–24]. Each point $(x, y) \in \square_h$ is mapped onto a point (ξ, η) by

$$x(\xi, \eta) = \sum_{i=1}^{4} x_i \chi_i(\xi, \eta), \tag{2.21}$$

$$y(\xi, \eta) = \sum_{i=1}^{4} y_i \chi_i(\xi, \eta), \tag{2.22}$$

where

$$\begin{aligned}
\chi_1(\xi, \eta) &= (\xi - 1)(\eta - 1)/4, \\
\chi_2(\xi, \eta) &= -(\xi + 1)(\eta - 1)/4, \\
\chi_3(\xi, \eta) &= (\xi + 1)(\eta + 1)/4, \\
\chi_4(\xi, \eta) &= -(\xi - 1)(\eta + 1)/4,
\end{aligned}$$

are the Q1 basis functions for the reference element $\square_*$ [31, p.31]. The contributions to the elemental diffusion and convection matrices for (2.18)–(2.19) are calculated:

$$d_{ij}^h = \sum_{t=1}^{m} \sum_{l=1}^{m} \kappa_{tl} |J_h(\xi_t, \eta_l)| \left\{ \frac{\partial \varphi_{i,\square_*}}{\partial x} \frac{\partial \varphi_{j,\square_*}}{\partial x} + \frac{\partial \varphi_{i,\square_*}}{\partial y} \frac{\partial \varphi_{j,\square_*}}{\partial y} \right\} |_{(\xi_t, \eta_l)} \qquad i, j = 1, ..., 4\,,$$

$$c_{ij}^h = \sum_{t=1}^{m} \sum_{l=1}^{m} \kappa_{tl} |J_h(\xi_t, \eta_l)| \left\{ \vec{w} (\frac{\partial \varphi_{j,\square_*}}{\partial x} + \frac{\partial \varphi_{j,\square_*}}{\partial y}) \varphi_{i,\square_*} \right\} |_{(\xi_t, \eta_l)} \qquad i, j = 1, ..., 4\,,$$

where κ is the tensor product of the weights associated with the classical one-dimensional quadrature rule [31, p.32] and J_h is the Jacobian matrix representing the mapping from $\square_*$ to $\square_h$ [31, p.31]:

$$J_h = \frac{\partial (x, y)}{\partial (\xi, \eta)} = \begin{bmatrix} \sum_{i=1}^{4} x_i \frac{\partial \chi_i}{\partial \xi} & \sum_{i=1}^{4} y_i \frac{\partial \chi_i}{\partial \xi} \\ \sum_{i=1}^{4} x_i \frac{\partial \chi_i}{\partial \eta} & \sum_{i=1}^{4} y_i \frac{\partial \chi_i}{\partial \eta} \end{bmatrix}.$$

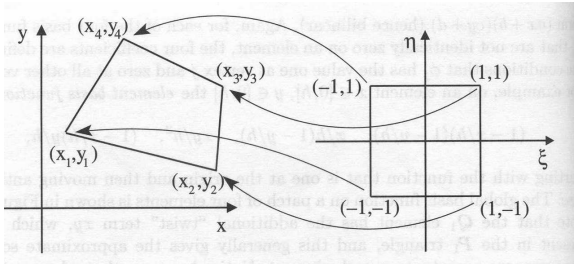


Figure 2.2: Q1 mapping from the reference element to a arbitrary element [31, p.24].

The accuracy of the Galerkin method for a convection-diffusion problem (2.7) is bounded by [31, p.135]:

$$||\nabla(u - u_h)|| \leq Ch_{max}||D^{\mathbf{2}}u||, \tag{2.23}$$

where C is a constant proportional to the mesh Peclet number [31, p.135], h_{max} is the largest element length, and $||D^{\mathbf{2}}u|| = (\int_\Omega \left((\frac{\partial^2 u}{\partial x^2})^2 + (\frac{\partial^2 u}{\partial x \partial y})^2 + (\frac{\partial^2 u}{\partial y^2})^2\right))^{\frac{1}{2}}$ measures the $\mathcal{H}^2$ regularity [31, Definition 1.9] of the solution u_h. For a Poisson problem the accuracy of the Galerkin method is bounded by (2.23) when C is a constant only [31, p.45]. If the convection-diffusion and diffusion equation is $\mathcal{H}^2$ regular then (2.23) suggests that $u_h \to u$ as $h_{max} \to 0$. The upper bound for the accuracy of the Galerkin method (2.23) is considered an optimal estimate when the mesh Peclet number ≤ 1 and there are no layers [31, p.136]. For elements that lie within the solution layers the term, $||D^{\mathbf{2}}u||$ rapidly increases as $\epsilon \to 0$ [31, p.136]. Furthermore as the constant C in (2.23) is proportional to the mesh Peclet number, we would find that as the problem becomes more convection-dominated the upper bound of the solution error may become prohibitively large.

If standard Galerkin FEM is used to discretise a convection-dominated convection-diffusion equation (2.7), layers can occur due to a discrepancy between the boundary conditions that are suitable for second order problem and the solution u that satisfies the first order problem. Layers can manifest themselves when there is a difference between Dirichlet inflow and outflow boundary conditions, or in cases when the Dirichlet inflow and outflow boundary conditions are the same but are different from those at the characteristic boundaries [30]. On the other hand the discretisation by Galerkin FEM for a diffusion-dominated problem is relatively trouble free. Therefore oscillations occur due to a rapid rate of change in the flow direction, which are too large to be dealt with by the current mesh size. Using Galerkin FEM to approximate a solution to a convection-diffusion problem where steep layers occur can cause difficulties if the mesh is not refined sufficiently in parts of the domain where the layers occur.

The difficulties manifest themselves as non-physical oscillations or "wiggles" in the solution u over the whole domain.

The local element Peclet number is defined as $(Pe)_h = \frac{h_k||\vec{w}_k||}{2\epsilon}$, where h_k is the size of element k in the direction of the wind, and $||\vec{w}_k||$ is the norm of the wind at the centre of the element [31, p.132]. Gresho et al. [42, p.127] characterizes the boundary layer thickness (asymptotic width of the layer) as $\frac{\epsilon}{||\vec{w}_k||} < \frac{h_k}{2}$. This, in turn, relates to $(Pe)_h > 1$. To obtain an accurate Galerkin FE approximation, one needs to create a mesh with a characteristic mesh width that is smaller than the layer width. Thus, if the element Peclet number satisfies $(Pe)_h > 1$ oscillations will occur, if $(Pe)_h \gg 1$ oscillations will become more widespread [42, p.127, p.230].

Example 2.2.1. A convection-diffusion equation (2.7) with vertical wind $\vec{w} = (0, 1)$ on a square domain $\Omega = [-1, 1]^2$, with the following boundary conditions [31, p.116]:

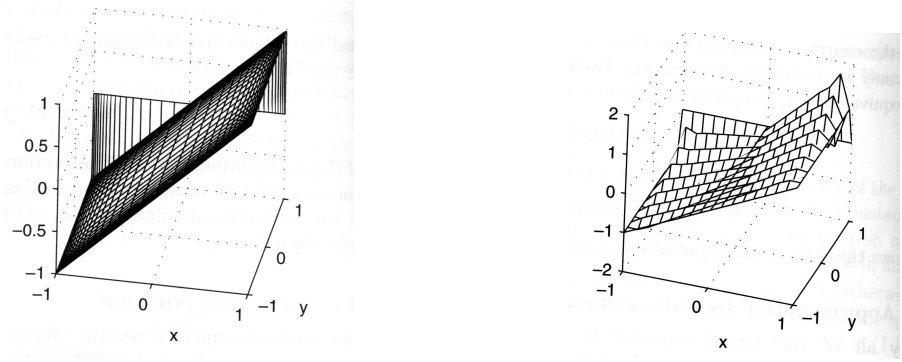
$$\begin{aligned} u(x,-1) &= x; & u(-1,y) &= -1; \\ u(x,1) &= 0; & u(1,y) &= 1. \end{aligned}$$

▷

A very accurate solution to Example 2.2.1 is shown in Figure 2.3(a) [31, p.116]. Near the outflow $y = 1$ there is an exponential boundary layer where the solution is attempting to satisfy the boundary condition. In Figure 2.3(b) the Galerkin FE solution for Example 2.2.1, obtained on a coarse uniform grid is presented. By comparing Figure 2.3(a) and Figure 2.3(b) it is clear that there is a global discrepancy in Figure 2.3(b). This global discrepancy originates from the steep gradient area near $y = 1$. The reason for this is down to the mesh being too coarse around an area that requires an exponentially fine mesh. A boundary layer near $y = 1$ then acts as a catalyst which spreads oscillations throughout the whole domain [31, p.124].

There are a number of techniques to resolve the problem of layers. The least creative is to uniformly refine the mesh over the entire domain until $(Pe)_h < 1$ in each element. In the case of three-dimensional domains, this approach can become prohibitively expensive very quickly. Moreover, as the Pe number increases i.e. the convection term becomes more dominant, and the equation moves closer to a hyperbolic equation, increasingly more levels of refinement are needed to obtain an accurate solution. However Gresho et al. [42, p.132] believe that uniform refinement should be used for convection-dominated flows if there are no obstructions, with hard Dirichlet BC, put in front of the flow.

A more effective alternative in dealing with layers is to refine the mesh only in local areas of the domain where layers occur. One method to achieve this is through incorporating a mesh grading system, through the use of a posteriori error



(a) (b)

Figure 2.3: FE solution obtained from a convection-diffusion equation with vertical wind $\vec{w} = (0, 1)$ and BC's $u(x, -1) = x$, $u(x, 1) = 0$, $u(-1, y) \approx -1$, $u(1, y) \approx 1$. (a) stretched grid around the boundary area $y = 1, -1$ [31, p.116], (b) coarse grid [31, p.124].

analysis [31, Section 3.4.2]. This is an example of adaptive mesh refinement. The disadvantage of this method is that the problem must be solved at each refinement to allow error analysis. An additional difficulty with adaptive grid refinement is that the resulting linear systems can be very ill-conditioned and the coefficient matrices are poorly scaled. The advantage of using this approach is that the same level of global accuracy is possible to achieve with fewer degrees of freedom than in the case of uniform refinement. This can lead to a smaller computational time. Gresho et al. [42, p.133] see the use of targeted mesh refinement as a tool for convection-dominated flows where obstructions, with hard BC's, are put in front of the flow that can lead to layers in the solution error.

Given (2.23), the Galerkin approximation in the case of convection-dominated problems will give a poor approximation [31, p.135], if the grids are not sufficiently refined. Another alternative to solving the problem of layers is to use an arbitrary grid size and stabilise the solution by using a modification of the Galerkin approximation, referred to as the streamline upwind Petrov-Galerkin approximation method (SUPG) [31, pp.125–130] [30].

2.3 The Petrov-Galerkin (SUPG) approximation

The concept of SUPG is to add artificial diffusion, in the direction of the convection, to the original problem. This addition of diffusion will "smooth" any sudden changes that are caused by convection. This modification is incorporated by modifying the test space, and is called a Petrov-Galerkin method. While the Galerkin method has the same choice of test and trial space, in the Petrov-Galerkin method these choices

are independent of each other.

The test space for v that will be used consists of streamline upwind functions:

$$\overline{H}_0^1(\Omega) = \left\{ v + \delta\vec{w}\cdot\nabla v \,|\, v \in H_0^1(\Omega) \right\},$$

where $\delta > 0$ is regarded as a constant stabilisation parameter. When $\delta = 0$, SUPG is reduced to the Galerkin method. The weak formulation of the problem (2.7) is to find $u \in H_D^1(\Omega)$ such that

$$\begin{aligned} -\epsilon \int_\Omega \nabla^2 u(v + \delta\vec{w}\cdot\nabla v)\, d\Omega + \int_\Omega (\vec{w}\cdot\nabla u)(v + \delta\vec{w}\cdot\nabla v)\, d\Omega \\ = \int_\Omega f(v + \delta\vec{w}\cdot\nabla v)\, d\Omega, \qquad \forall v \in H_0^1(\Omega). \end{aligned} \tag{2.24}$$

Applying Green's theorem to the diffusion term in (2.24) and splitting the integrals of the sums into separate parts, we obtain:

Find $u_h \in S_D^h \subset H_0^1(\Omega)$ such that

$$\begin{aligned} \epsilon \int_\Omega \nabla u_h \cdot \nabla v_h\, d\Omega + \int_\Omega (\vec{w}\cdot\nabla u_h) v_h\, d\Omega + \delta \int_\Omega (\vec{w}\cdot\nabla u_h)(\vec{w}\cdot\nabla v_h)\, d\Omega \\ - \delta\cdot\epsilon \int_\Omega \nabla^2 u_h (\vec{w}\cdot\nabla v_h)\, d\Omega \\ = \int_\Omega f(v_h + \delta\vec{w}\cdot\nabla v_h)\, d\Omega, \qquad \forall v_h \in S_0^h \subset H_0^1(\Omega). \end{aligned} \tag{2.25}$$

The term $\delta\cdot\epsilon\int_\Omega \nabla^2 u_h(\vec{w}\cdot\nabla v_h)\, d\Omega$ is non-conformal in $H^1(\Omega)$, as it contains a second order derivative. The issue here is that $u_h \in S_D^h \subset H_D^1(\Omega)$ does not guarantee the integral is finite. However,

$$\delta \int_\Omega \nabla^2 u_h(\vec{w}\cdot\nabla v_h)\, d\Omega = \sum_k \delta_k \int_{\Omega_k} \nabla^2 u_h(\vec{w}\cdot\nabla v_h)\, d\Omega, \tag{2.26}$$

that is, the integration can be performed elementwise k (where δ_k is defined in (2.32)). This implies that in cases when φ_j consists of either linear or bilinear functions, each of the elemental integrals will be zero, and this term can be omitted from (2.25). However, if a higher order FE approximation is used, this integral has to be taken into account and computed by (2.26)[31, p.131].

With linear/bilinear approximation of the solution (2.25) simplifies to

$$\begin{aligned} \sum_{j=1}^{N_I+N_D} x_j \Big[\epsilon \int_\Omega \nabla\varphi_j\cdot\nabla\varphi_i\, d\Omega + \int_\Omega (\vec{w}\cdot\nabla\varphi_j)\,\varphi_i\, d\Omega \\ + \delta \int_\Omega (\vec{w}\cdot\nabla\varphi_j)(\vec{w}\cdot\nabla\varphi_i)\, d\Omega \Big] \\ = \int_\Omega f\varphi_i\, d\Omega + \int_\Omega f(\delta\vec{w}\cdot\nabla\varphi_i)\, d\Omega, \qquad i = 1, ..., N_I. \end{aligned} \tag{2.27}$$

Rearranging (2.27) to move the Dirichlet BC terms to the right hand side gives

$$\varpi_i = \sum_{j=N_I+1}^{N_I+N_D} (g)_j \left[\epsilon \int_\Omega \nabla\varphi_j \cdot \nabla\varphi_i \, d\Omega + \int_\Omega (\vec{w} \cdot \nabla\varphi_j) \; \varphi_i \, d\Omega \right. \\ \left. + \delta \int_\Omega (\vec{w} \cdot \nabla\varphi_j)(\vec{w} \cdot \nabla\varphi_i) \, d\Omega \right],$$

$$\boxed{\begin{aligned} &\sum_{j=1}^{N_I} x_j \;\; \left[\epsilon \int_\Omega \nabla\varphi_j \cdot \nabla\varphi_i \, d\Omega + \int_\Omega (\vec{w} \cdot \nabla\varphi_j) \; \varphi_i \, d\Omega \right. \\ &\qquad \left. + \delta \int_\Omega (\vec{w} \cdot \nabla\varphi_j)(\vec{w} \cdot \nabla\varphi_i) \, d\Omega \right] \\ &= \int_\Omega f \; \varphi_i \, d\Omega + \int_\Omega f(\delta\vec{w} \cdot \nabla\varphi_i) \, d\Omega - \; \varpi_i, \qquad i = 1, ..., N_I. \end{aligned}} \tag{2.28}$$

As in the Galerkin case, (2.28) represents a system of linear equations of the size N_I for the unknowns $\{x_i\}_{i=1}^{N_I}$. The coefficient matrix of the linear system now has an additional component (compared with (2.16)), which is called the streamline diffusion (stabilisation) matrix S:

$$(\epsilon \cdot \; D + C + S)\bar{x} = \bar{b}. \tag{2.29}$$

The diffusion matrix D and the convection matrix C are represented by (2.18) and (2.19) respectively.

- The properties of the streamline diffusion matrix S are that it is symmetric and positive definite.

$$S = [s_{ij}] = \delta \int_\Omega (\vec{w} \cdot \nabla\varphi_j)(\vec{w} \cdot \nabla\varphi_i) \, d\Omega \qquad i, j = 1, ..., N_I. \tag{2.30}$$

  From (2.30) it follows that the additional term in the SUPG method represents the diffusion operator in the direction of the convection, that is, along the streamlines of the flow $\vec{w}$.

  However, if δ is defined locally the streamline diffusion matrix S is only symmetric and positive semi-definite, as some rows correspond to nodes associated with element patches where no stabilisation is needed (this is represented by a zero row) [31, p.152]:

$$S = [s_{ij}] = \sum_k \delta_k \int_{\Omega_k} (\vec{w} \cdot \nabla\varphi_j)(\vec{w} \cdot \nabla\varphi_i) \, d\Omega \qquad i, j = 1, ..., N_I. \tag{2.31}$$

The stabilisation parameter δ from (2.30), controls the amount of artificial diffusion being added. It is calculated locally, at the element level. In this way we can account for different strengths and directions of convection in different parts of the domain Ω,

and add just the "correct" amount of stabilisation for this element. The stabilisation parameter δ at an element (local) level is δ_k, and is given by

$$\delta_k = \begin{cases} \frac{h_k}{2||\vec{w_k}||}(1 - \frac{1}{(Pe)_h}) & \text{if } (Pe)_h > 1, \\ 0 & \text{if } (Pe)_h \leq 1. \end{cases} \tag{2.32}$$

It can be concluded that an element stabilisation is added if $h_k > 2\,\epsilon||\vec{w_k}||$. This strategy is optimal for quadrilateral elements, as it is a generalisation of the one-dimensional case [31, p.132].

If Example 2.2.1 is discretised by a uniform tensor product grid of size h, the stabilisation parameter δ_k can be further simplified to resemble,

$$\delta_k = \frac{h}{2}(1 - \frac{1}{(Pe)_h}) = \frac{1}{2}(h - 2\,\epsilon). \tag{2.33}$$

This means that if the mesh size h in a particular part of the domain is greater than $2\,\epsilon$, then stabilisation is needed on that element. Gresho [41] discusses eliminating oscillations "wiggles" through using mesh refinement. If a mesh size in part of the domain where a boundary layer exists is at most half of the layer's size, i.e. if there are at least two grid points within the width of the solution layer, the oscillations in Galerkin FEM solution will not exist. As the asymptotic width of the solution boundary layer in this case is $\sim \epsilon$, it follows that the formula (2.33) and Gresho's method are complementary: in the former case if $h > 2\,\epsilon$ we need to apply the stabilisation, while in the latter case we need to refine the grid locally and sufficiently many times until the condition $h < 2\,\epsilon$ is met.

When $(Pe)_h \leq 1$ no stabilisation is needed. If this condition is satisfied for every element in the mesh, then the SUPG method reduces to the standard Galerkin method, $\delta = 0$ in (2.30). That is, $S \equiv 0$. Therefore, (2.32) is the optimal choice (2.23) for $(Pe)_h \leq 1$. On the other hand as $(Pe)_h \to \infty$ the stabilisation parameter $\delta_k \to \frac{h_k}{2||\vec{w_k}||}$. This is considered optimal value for the convection limit [32, p.463] [31, p.133].

In Figure 2.4 we see that SUPG with an optimal choice of stabilisation parameter gives a satisfactory solution (blue). One the other hand, the Galerkin solution (green) shows large oscillations throughout. If a non-optimal choice of the stabilisation parameter is used, this can lead to either an oscillatory behaviour of the solution (gray) or the over smoothing of the solution (red). Galerkin approximation in comparison to SUPG for a convection-diffusion problem with $(Pe)_h \gg 1$ and coarse mesh discretisation is known to be inaccurate [31, p.197] [42, p.201]. This can be seen numerically in [31, pp.139–141].

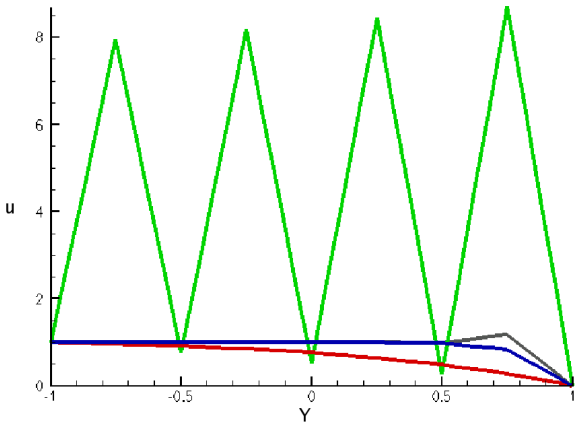


Figure 2.4: Convection-diffusion equation with a domain $\Omega = [-1, 1]^2$ where $y = -1$ is the only heated wall, and a wind $\vec{w} = (0, 1)$. One-dimensional slice at $x = 0$ of the solution to the convection-diffusion problem with $N_I = 49$ and $Pe = 1000$. Problem discretisation is performed with Q_1 Galerkin FEM (green) and Q_1 SUPG FEM with different values of the stabilisation parameter: $\delta_k = \delta_{opt} * 0.5$ (grey), $\delta_k = \delta_{opt}$ (blue), $\delta_k = \delta_{opt} * 5$ (red), where δ_{opt} is given by (2.32).

The disadvantage with the SUPG method is that the upwind stabilisation suggests to the user that the solution is accurate. While the computed solution may be accurate globally, the main issue is the local accuracy of the obtained FE approximation. By adding too much diffusion the boundary layers become wider, in comparison to an exact solution (the SUPG method over smoothes the solution) [31, p.161]. However the advantage of SUPG is that it does reduce the damage that is caused by boundary layer oscillations, making the solution more accurate in comparison to using Galerkin approximation on an inappropriately coarse mesh [31, p.134].

Stabilisation is only relevant when the mesh is insufficiently refined. By using Fourier analysis, on a stencil level, it can be shown that for $(Pe)_h > 1$ oscillations can occur in the solution when using Galerkin approximation [31, p.158]. Furthermore, by using SUPG there is a reduced risk of oscillations occurring [31, p.160].

The solution outside the boundary layer is qualitatively accurate, in most cases, where the layers are not always fully resolved [31, p.140]. This is shown clearly from [31, p.141, Table 3.2], where the residual error is roughly resolved by SUPG on all refinement levels. By contrast, the Galerkin approximation is slow to reduce the residual error, as the mesh becomes finer, until a point where the mesh is sufficiently refined to resolve the oscillations. A further advantage of using SUPG is that it can be used in conjunction with mesh refinement techniques, therefore SUPG can be controlled and used in circumstances when in certain parts of the domain the mesh has not yet been correctly refined. This will be an essential concept in the following

chapter when multiple levels of mesh refinement are introduced in the context of a MG [31, p.195].

2.4 Properties of the discrete operator

In this section we give details on the properties of the coefficient matrix A obtained from either Galerkin or SUPG FEM discretisation of the convection-diffusion problem (2.7) (the equations (2.16) and (2.28) respectively). The discrete convection-diffusion matrix consists of the diffusion matrix D (scaled by ϵ), the convection matrix C, and, in the case of the SUPG method, the streamline diffusion matrix S (2.29). We examine the matrix properties of A by considering the matrix stencil of each of the components. Matrix stencils depict matrix coefficients in a particular row of the global matrix as a function of the connectivity in the underlying FE mesh. Matrix stencils are simple in the case of uniform, tensor product grids, where there exists a fixed connectivity pattern between the nodes in the grid.

The stencils of the matrices D, C and S, connectivity pattern in the case of a 2D tensor product grid and bilinear approximation, can be represented as [31, p.154]:

$$D \sim \tfrac{1}{3}\times \begin{array}{ccc} & -1 & \\ -1 & 8 & -1 \\ & -1 & \end{array}, \qquad C \sim \tfrac{h}{12}\times \begin{array}{ccc} & 4w_y & \\ -w_x+w_y & 0 & w_x+w_y \\ -(w_x+w_y) & -4w_y & w_x-w_y \end{array},$$

$$S \sim \delta\times \begin{array}{ccc} \frac{1}{3}w_x^2-\frac{2}{3}w_y^2 & & -\frac{1}{6}(w_x^2+w_y^2)-\frac{1}{2}(w_xw_y) \\ -\frac{2}{3}w_x^2+\frac{1}{3}w_y^2 & \frac{4}{3}(w_x^2+w_y^2) & -\frac{2}{3}w_x^2+\frac{1}{3}w_y^2 \\ -\frac{1}{6}(w_x^2+w_y^2)+\frac{1}{2}(w_xw_y) & & \frac{1}{3}w_x^2-\frac{2}{3}w_y^2 \end{array},$$

where $\vec{w} = (w_x, w_y)$ is the wind function at a particular (stencil) point.

These stencils can be greatly simplified, depending on the wind function. For the simple case introduced by Example 2.2.1 with vertical wind $\vec{w} = (0, 1)$, the stencils of the component matrices become:

$$D \sim \tfrac{1}{3}\times \begin{array}{ccc} & -1 & \\ -1 & & -1 \\ -1 & 8 & -1 \\ -1 & & -1 \\ & -1 & \end{array}, \quad C \sim \tfrac{h}{12}\times \begin{array}{ccc} & 4 & \\ 1 & & 1 \\ 0 & 0 & 0 \\ -1 & & -1 \\ & -4 & \end{array},$$

$$S \sim \tfrac{1}{2}(h-2\epsilon)\times \begin{array}{ccc} & -\frac{2}{3} & \\ -\frac{1}{6} & & -\frac{1}{6} \\ \frac{1}{3} & \frac{4}{3} & \frac{1}{3} \\ -\frac{1}{6} & & -\frac{1}{6} \\ & -\frac{2}{3} & \end{array}.$$

From the simple Example 2.2.1 we see that the convection matrix stencil is a function of the discretisation parameter h, while the diffusion matrix has a constant stencil (in 2D). This means that diagonal dominance, and therefore the effectiveness of the solvers, is dependent on h. Furthermore, it is easy to understand that for a fixed h, the increase in the value Pe (i.e $\epsilon \to 0$) will lead to the situation where the matrix A loses its diagonal dominance. This has implications for the choice of iterative method suitable for the systems (2.16) or (2.28). The example also demonstrates the stabilising effect that the matrix S has in such circumstances, whereby it has increased the symmetric part of A (Definition 2.4.7).

Diagonal dominance of the coefficient matrix is crucial for the convergence of simple-point iterations which are, in turn, a key ingredient of MG methods (see Chapter 3).

Definition 2.4.1 ([77])**.** Matrix A is strictly diagonal dominant when

$$|a_{ii}| > \sum_{\substack{j=1 \\ i\neq j}}^{N_I} |a_{ij}| \qquad i = 1, ..., N_I\,.$$

Definition 2.4.2 ([77])**.** Matrix A is weakly diagonal dominant when

$$|a_{ii}| \geq \sum_{\substack{j=1 \\ i\neq j}}^{N_I} |a_{ij}| \qquad i = 1, ..., N_I\,.$$

Definition 2.4.3 ([37])**.** Condition number of matrix A:

$$\kappa(A) = ||A|| \cdot ||A^{-1}||,$$

where the matrix norm $||\cdot||$ is consistent with any vector norm.

Example 2.4.1. We demonstrate the diagonal dominance of the coefficient matrices for different values of h and Pe for Example 2.2.1 discretised on a sequence of uniform tensor product grids, with each finer level obtained by a uniform refinement of the previous coarse grid. Discretisation of the convection-diffusion problem (2.7) is performed by SUPG FEM with Q1 elements. From Table 2.1 it can be seen that as the Peclet number increases and/or the mesh becomes more coarse (h increases) the diagonal dominance (Definition 2.4.2) of the coefficient matrix is lost. This is an important feature in a multilevel context, as on coarse grids, iterative methods may fail to converge. ▷

Table 2.1: Diagonal dominance of the coefficient matrix A from (2.29) as a function of the grid size h and the Peclet number Pe in the case of a constant wind from Example 2.4.1. Here '1' denotes the cases when the coefficient matrix is weakly diagonally dominant (Definition 2.4.2) and '0' denotes the cases when this property is lost. The shaded areas in the table correspond to the cases when $h > 2\,\epsilon$.

N	9	49	225	961	3,969	16,129	65,025
Pe=40	0	0	0	1	1	1	1
Pe=100	0	0	0	0	1	1	1
Pe=200	0	0	0	0	0	1	1
Pe=500	0	0	0	0	0	0	0
Pe=1000	0	0	0	0	0	0	0

The main property of the convection-diffusion coefficient matrix, obtained from FE discretisation, is its sparsity. Sparsity is broadly defined as "a matrix is sparse if there is an advantage in exploiting its zeros" [23, p.1]. In addition, coefficient matrices arising in FE discretisations are known to be ill-conditioned. The extent of this ill-conditioning is dependent on the type of problem, the type of discretisation, the problem parameters used and the discretisation parameter h [32, p.112].

The main difficulty with the discrete convection-diffusion problem is that the matrix is non-normal [84, Chapter 12]. This property has a profound effect on the choice of iterative method used for the solution of the system and its accuracy. To give some insight into this issue, we need to introduce some definitions.

Definition 2.4.4 ([77])**.** Matrix $A \in \mathbb{R}^{N_I \times N_I}$ is an M-matrix if

1. A is nonsingular,

2. $a_{i,j} \leq 0$ for $i \neq j$,

3. $(A^{-1})_{i,j} \geq 0$,

4. $a_{i,i} > 0$.

However, conditions 1 and 3 from Definition 2.4.4 can be replaced by $\sigma(B) < 1$, where $\sigma(B)$ is the spectral radius of B and $B = I - D^{-1}A$ [77, p.28].

Definition 2.4.5 ([77])**.** Matrix A is Hermitian if

$$A^H = A,$$

or $A^T = A$ if A is a real matrix.

Definition 2.4.6 ([77])**.** Matrix A is normal if it satisfies the relation

$$A^H A = A A^H.$$

The matrix is referred to as non-normal if a matrix of order N_I does not have N_I eigenvectors (complete set) or the eigenvectors do not form an orthogonal set. By contrast, the diffusion matrix D (which is a symmetric positive definite matrix) has a complete orthogonal set of eigenvectors. Non-normality of a convection-diffusion matrix increases with Pe [42, p.212]. This has an effect on the accuracy of the iterative solutions and the choice of the stopping criterion for a iterative solver.

Definition 2.4.7. The symmetric part of matrix A:

$$H = \frac{A + A^T}{2}.$$

Definition 2.4.8. The skew-symmetric part of matrix A:

$$F = \frac{A - A^T}{2}.$$

2.5 The Navier-Stokes problem

The steady state Navier-Stokes equations in two spacial dimensions is defined as:

Find $(\vec{u}, p) \in ([\mathcal{C}^2(\Omega)]^2 \times \mathcal{C}^1(\Omega))$ such that

$$\left.\begin{array}{ll} -\epsilon\nabla^2\vec{u} + \vec{u}\cdot\nabla\vec{u} + \nabla p & = \vec{f} \\ \nabla\cdot\vec{u} & = 0 \end{array}\right\} \text{in } \Omega. \tag{2.34}$$

where $\vec{u} = [u_x; u_y]$ is the velocity, p represents the pressure and $\epsilon > 0$ is the molecular viscosity which is inversely proportional to the Reynolds number (Re) in a non-dimensional form of the equations [31, p.314]. The first equation in (2.34) is referred to as the momentum equation (Newton's second law). That is, an external force $\vec{f}$ (such as gravity [31, p.3]) is balanced by the diffusive force $-\epsilon\nabla^2\vec{u}$, pressure gradient ∇p and convection $\vec{u}\cdot\nabla\vec{u}$. The second equation in (2.34) is referred to as the continuity equation and is an expression for the conservation on mass (or incompressibility constraint). This can be seen if the divergence theorem [31, p.1] is applied to the continuity equation. That is, the total flux of the fluid through the boundaries of any enclosed part of the domain Ω is equal to zero, which implies that the volume of fluid entering the domain is equal to the volume of fluid leaving the domain [31, p.215].

The Navier-Stokes equations (2.34) contains the convection-diffusion equation (2.7) $\nabla^2\vec{u} + \vec{u}\cdot\nabla\vec{u}$, as part of the momentum equation, with velocity playing the role of the convective field. The convection term $\vec{u}\cdot\nabla\vec{u}$ is nonlinear [31, p.313], this makes the entire Navier-Stokes problem nonlinear. Therefore, the discretisation results in a system of nonlinear equations, which needs to be solved iteratively (by, for example, Picard's [31, p.326] or Newton's method [31, p.325]). At each iteration, a linearised version of the problem (in the case of Picard's method [31, p.326], Oseen's problem) must be solved.

As was the case with convection-diffusion for Pe, when $Re \leq 1$ we have a diffusion-dominated problem and $Re \gg 1$ we have a convection-dominated problem [31, p.314]. As $Re \to \infty$ following the same pattern as a convection-diffusion problem the Navier-Stokes equations tends to a hyperbolic problem (the incompressible Euler equation [31, p.314]).

There are two main groups of fluid flow problems with respect to the boundary conditions: enclosed flow problems implied by imposing Dirichlet boundary conditions for both components of the velocity on the boundary $\partial\Omega$, and flow-through problems. In the latter case we can distinguish three parts on the boundary: the inflow, the characteristic, and the outflow boundary (similar to (2.4) in the convection-diffusion case). Using the notation $\partial\Omega_D$ for the Dirichlet part of the boundary and $\partial\Omega_N$ for the Neuman part, the typical BCs considered for the Navier-Stokes problem (2.34), on a two-dimensional domain, are of the form

$$\vec{u} = \vec{w} \text{ on } \partial\Omega_D, \qquad \epsilon\frac{\partial\vec{u}}{\partial\widehat{n}} - \vec{n}p = \vec{0} \quad \text{on} \quad \partial\Omega_N, \tag{2.35}$$

where $\frac{\partial\vec{u}}{\partial\widehat{n}}$ denotes the normal derivative of $\vec{u}$, i.e the derivative with respect to the normal direction on $\partial\Omega$ ($\vec{n}$ is the unit normal vector in an outward direction to the boundary) [31, p.214, p.313]. As there are no BCs associated with the pressure, the

pressure solution in the case of enclosed flow problems is unique up to a constant [31, p.313, p.314].

To derive a weak formulation of the BVP (2.34)–(2.35), we need to define the space of test functions and the trial space. The test space $\mathbf{H}^1_0(\Omega)$ is a subspace of $\mathbf{H}^1(\Omega) = [H^1(\Omega)]^2$, with elements that satisfy the homogeneous Dirichlet boundary conditions on $\partial\Omega_D$:

$$\mathbf{H}^1_0(\Omega) = \left\{ \vec{v} \,|\, \vec{v} \in \mathbf{H}^1(\Omega) \ , \ \vec{v} = \vec{0} \text{ on } \partial\Omega_D \right\}.$$

The trial (solution) space $\mathbf{H}^1_D(\Omega)$ is also a subspace of $\mathbf{H}^1(\Omega)$, with elements that satisfy the same Dirichlet BCs as (2.35):

$$\mathbf{H}^1_D(\Omega) = \left\{ \vec{u} \,|\, \vec{u} \in \mathbf{H}^1(\Omega) \ , \ \vec{u} = \vec{w} \text{ on } \partial\Omega_D \right\}.$$

The weak formulation (in residual form) is derived as follows [31, p.319]:

Find $\vec{u} \in \mathbf{H}^1_D(\Omega)$ and $p \in L_2(\Omega)$, such that

$$\boxed{\begin{aligned} R^U &= -\epsilon \int_\Omega \nabla\vec{u} : \nabla\vec{v}\, d\Omega + \int_\Omega (\vec{u}\cdot\nabla\vec{u})\cdot\vec{v}\, d\Omega + \int_\Omega p(\nabla\cdot\vec{v})\, d\Omega \\ &\qquad - \int_\Omega \vec{f}\cdot\vec{v}\, d\Omega = \vec{0} \qquad \forall \vec{v} \in \mathbf{H}^1_0(\Omega), \\ R^P &= \int_\Omega q(\nabla\cdot\vec{u})\, d\Omega = 0 \qquad \forall q \in L_2(\Omega), \end{aligned}} \tag{2.36}$$

where $\nabla\vec{u} : \nabla\vec{v} = \nabla u_x \cdot \nabla v_x + \nabla u_y \cdot \nabla v_y$.

The FE discretisation of the problem (2.36) is achieved using a mixed method. A mixed method means that the approximation spaces of the velocity and pressure components of the solution can be selected to be two independent function spaces. However, not all combinations of the approximation spaces lead to a stable approximation. The condition that needs to be fulfilled for a stable approximation is referred to as the inf-sup (LBB) condition (see [31, p.224, p.228]). In the case of enclosed flow, satisfying the LBB condition is sufficient for the pressure to be unique up to a constant. The mixed FE approximation spaces used within the project for the Navier-Stokes problem (Section 4.6) are isoparametric LBB stable $Q_2 - Q_1$, also known as Taylor-Hood elements. That is, bi-quadratic Q_2 finite elements for the velocity $\vec{u}$ and bi-linear Q_1 finite elements for pressure p. The combination of these rectangular elements are known to be stable [31, Chapter 5.3.1]. An alternative stable mixed finite element approximation available in the OOMPHLIB library is Crouzeix-Raviart [45].

If we denote by φ_j the global bi-quadratic basis function associated with the node

j and $\vec{\varphi}_j$ represents either $[0\,;\,\varphi_j]$ or $[\varphi_j\,;\,0]$ [31, p.226]. If we denote $\vec{u}_h$ the FE approximation of the velocity, the approximation of the velocity can be defined as a linear combination of the vector basis functions and associated coefficient velocity components $\overline{U}^x \in \mathbb{R}^{N_u}$ and $\overline{U}^y \in \mathbb{R}^{N_u}$ (where $\overline{U} = [\overline{U}^x; \overline{U}^y] \in \mathbb{R}^{2N_u}$) by[1],

$$\vec{u}_h = \sum_{j=1}^{2N_u} \overline{U}_j \vec{\varphi}_j, \tag{2.37}$$

and the approximation of the pressure field p_h represented as a linear combination, using a bi-linear basis function ψ_j and $P_j \in \mathbb{R}^{N_p}$, $j = 1, \ldots, N_p$,

$$p_h = \sum_{j=1}^{N_p} P_j \psi_j. \tag{2.38}$$

In (2.37) and (2.38) N_u denotes the number of unknowns representing the velocity and N_p denotes the number of unknowns representing the pressure field.

The error of the $Q_2 - Q_1$ FE approximation of the Navier-Stokes problem is bounded by [31, p.332]:

$$||\nabla(\vec{u} - \vec{u}_h)|| + ||p - p_h|| \leq C h_{max}^{\mathbf{2}}(||D^{\mathbf{3}}\vec{u}|| + ||D^{\mathbf{2}}p||), \tag{2.39}$$

where C is a constant.

Stable FE discretisation of the Navier-Stokes problem must have a velocity space "richer" (of higher dimensions) than the pressure approximation space [31, p.229]. If the approximation spaces do not satisfy the discrete inf-sup stability condition [31, p.224, p.228], some form of stabilisation must be added (such as to relax the incompressibility constraint) [31, Chapter 5.3.2].

Finite element discretisation of the Navier-Stokes problem results in a system of nonlinear equations, which is solved in OOMPHLIB iteratively using Newton's method. Denote the nonlinear residual of the Navier-Stokes system as $R = [R^{\overline{U}}; R^{\overline{P}}]$ which corresponds to the current approximation to the solution $X = [\overline{U}; \overline{P}]$. The procedure is summarised in Algorithm 2.1.

When Newton's method is applied to a linear system, Algorithm 2.1 will terminate after one Newton iteration. For the nonlinear system of equations Newton's method will converge quadratically, given that the initial approximation is sufficiently close to the fixed-point [31, p.326].

By applying Newton's method to the Navier-Stokes system, at each iteration the

[1]The boundary conditions are not included for simplicity.

Algorithm 2.1 Newton method [54] [45]

Given an initial approximation for the unknowns X^0; $k = 0$
Compute the nonlinear residuals $R(X^0)$
do while $(||R(X^k)|| \geq e_N)$
  Assemble the Jacobian matrix $J(X^k) = \frac{\partial R}{\partial X^k}$
  Solve the linear system $J(X^k)\ \delta X^k = R(X^k)$
  Correct the current approximation $X^{k+1} = X^k - \delta X^k$
  Compute the nonlinear residual $R(X^{k+1})$; $k = k + 1$
end while

following block linear system is obtained [31, pp.328–330]:

$$\overbrace{\begin{bmatrix} F+W & B^T \\ B & 0 \end{bmatrix}}^{J(X)} \overbrace{\begin{bmatrix} \delta\overline{U} \\ \delta\overline{P} \end{bmatrix}}^{\delta X} = \overbrace{\begin{bmatrix} R^{\overline{U}} \\ R^{\overline{P}} \end{bmatrix}}^{R(X)}, \tag{2.40}$$

where F is the momentum block, which represents a block-diagonal matrix with scalar convection-diffusion operators, and $B \in \mathbb{R}^{N_p \times 2N_u}$ is the discrete divergence operator. The scalar convection-diffusion operators in the momentum block F are comprised of the discrete diffusion operator $D \in \mathbb{R}^{2N_u \times 2N_u}$ and the discrete convection operator $C \in \mathbb{R}^{2N_u \times 2N_u}$. The block contributions to the coefficient matrix in (2.40) are defined as follows:

- The Laplacian matrix has the elements [31, p.225]:

$$D = [d_{ij}] = \int_\Omega \nabla\vec{\varphi}_j : \nabla\vec{\varphi}_i \, d\Omega \qquad i,j = 1, ..., N_u. \tag{2.41}$$

- The convection matrix has the elements [31, p.328]:

$$C = [c_{ij}] = Re \int_\Omega (\vec{u_h} \cdot \nabla\vec{\varphi}_j)\ \vec{\varphi}_i \, d\Omega \qquad i,j = 1, ..., N_u. \tag{2.42}$$

- The divergence matrix has the elements [31, p.225]:

$$B = [b_{ij}] = -\int_\Omega \psi_i \nabla \cdot \vec{\varphi}_j \, d\Omega \qquad i = 1, \ldots, N_p, \quad j = 1, ..., N_u. \tag{2.43}$$

- The term W is a contribution that arise from the application of Newton's method called the Newton derivative matrix, and has the elements [31, p.328]:

$$W = [w_{ij}] = Re \int_\Omega (\vec{\varphi}_j \cdot \nabla\vec{u}_h)\vec{\varphi}_i \, d\Omega \qquad i,j = 1, \ldots, N_u. \tag{2.44}$$

Chapter 3

Methods for Solving Linear Algebraic Systems

The discretisation of the convection-diffusion equation, by either Galerkin or the SUPG approximation method, leads to a non-symmetric sparse system of linear equations with a non-normal coefficient matrix. The solution of this system represents computationally the most demanding, and algorithmically the most challenging part, of the overall simulation process.

Two strategies are often considered when solving a linear system. Gauss elimination is a direct method that works equally well for any type of system. However, making it work efficiently for sparse systems is an additional challenge. For systems with dense coefficient matrices the asymptotic complexity of Gauss elimination is known in advance (the computational cost of the algorithm is $O(n^{\mathbf{3}})$ and memory cost $O(n^{\mathbf{2}})$). This is not the case for systems with sparse matrices, where the algorithmic complexity depends on the sparsity pattern, the number of non-zero entries and other matrix properties such as diagonal dominance and positive definiteness [64, Chapter 3]. A direct method is designed to produce an accurate solution up to computational round-off error (provided the coefficient matrix is not ill-conditioned).

To alleviate this problem, reordering techniques and graph reduction techniques are used to exploit sparsity patterns to lessen the potential increase in storage and computational costs [20, Chapter 6]. However the amount of "fill-in" (newly generated non-zero entries that do not exist in the coefficient matrix) grows rapidly with the number of degrees of freedom and spacial dimension of the continuous problem. The sparsity pattern also becomes more complex, even in cases of regular tensor product grids. When solving large systems that arise in the discretisation of 3D problems, direct solvers are faced with a considerable challenge [77, p.xvii]. This is the reason for looking at alternatives, namely iterative methods. Iterative methods offer the prospect of optimal scaling. This optimality is achieved through performing

only sparse matrix-vector products and a few vector updates. The storage cost of iterative methods typically includes the sparse coefficient matrix and a small number of auxiliary vectors.

Saad and van der Vorst [78] present the historical development of iterative solvers in the last 100 years. The survey covers the general progression of iterative solvers, starting with simple-point iterations and covering Krylov methods and MG. In [78] the authors relate rapid development of iterative solvers to the rise in applications, in particular the FE discretisation of PDEs and the development of computer systems.

In this chapter we give an overview of several different types of iterative methods for the solution of sparse linear systems. In particular, we cover details of simple iterations (referred to as simple-point iterations or smoothers), Krylov subspace solvers and MG. We then discuss the concept of preconditioning as a means of improving the speed of convergence of Krylov iterative methods and the use of MG in this context. The main aim of this project is the development of efficient smoothing techniques for multigrid preconditioning of the discrete convection-diffusion problem. In this context, we cover different smoothing techniques, starting from simple point smoothers with the default ordering of nodes, followed by different strategies for the lexicographical and downwind nodal ordering. We also cover more complex smoothing strategies, in particular, the incomplete factorisation smoothers, that forms the basis for a novel smoothing technique introduced in this thesis, which is based on incomplete factorisation of truncated matrices.

3.1 Basic iterative methods

We want to solve a system of linear equations

$$A\bar{x} = \bar{b}, \tag{3.1}$$

where $A \in \mathbb{R}^{n\times n}$ is a large, sparse, in general, non-symmetric matrix; $\bar{b} \in \mathbb{R}^n$ is the right-hand vector, and $\bar{x} \in \mathbb{R}^n$ is a vector of unknowns. Although the presentation in this section is general, we can think of (3.1) as a system that arises from the discretisation of PDEs, such as (2.28).

An iterative method for solving linear systems starts with a suitable guess for the solution $\overline{x}^0$ (where in the absence of a better initial guess a zero vector is used), which is then improved at each iteration, until the desired solution $\overline{x}^k$ is "sufficiently close" to the true solution $\overline{x}$ [64, p.1].

The criteria for a good iterative method are [80]:

- The computational cost of moving from $\overline{x}^k$ to $\overline{x}^{k+1}$ should be "cheap" with

respect to the cost of solving the linear system.

- The rate of convergence to a given tolerance should be "fast".

It is useful to be able to monitor the progress a solver makes towards the solution at each iteration. Two quantities that describe the quality of the approximate solution $\overline{x}^k$ are the *solution error* and the *residual*. The solution error at the k^{th} iteration is defined as

$$\overline{e}^k = \overline{x} - \overline{x}^k, \text{ or } \quad \overline{e}^k = A^{-1}\overline{b} - \overline{x}^k. \tag{3.2}$$

As the true solution $\overline{x}$ is not known in advance, the solution error cannot be computed directly and hence is of little practical use. On the other hand, if one computes the solution using a direct method one can use this as a reference solution $\overline{x}$, to assess the accuracy of an iterative method on small problem sets. Calculating the forward error of $||\overline{x}^k||$, $\frac{||\overline{x}-\overline{x}^k||}{||\overline{x}||}$ [52, p.12], is a method to measure or estimate how accurate $\overline{x}^k$ is with respect to $\overline{x}$.

The residual $\overline{r}^k$ at the k^{th} iteration is defined as

$$\overline{r}^k = \overline{b} - A\overline{x}^k, \tag{3.3}$$

where $\overline{x}^k$ is the k^{th} approximation to the solution.

The residual is a measure of how close $A\overline{x}^k$ is to $\overline{b}$. However, the accuracy of the residual (3.3) is scalar-dependent. That is, if A and $\overline{b}$ are multiplied by β, $\overline{r}$ is multiplied by β. An alternative, to make the residual scalar-independent, is to use the relative residual

$$\frac{||\overline{b} - A\overline{x}^k||}{||\overline{b}||}, \qquad \overline{x}^0 = 0. \tag{3.4}$$

Convergence of an iterative method is defined by $||\overline{e}^k|| \to 0$ as $k \to \infty$. As the residual is the only information that is known at each iteration, the following linear relationship between the solution error and the residual, referred to as the residual equation [10, p.8], is necessary:

$$A\overline{e}^k = \overline{r}^k. \tag{3.5}$$

From (3.5) it follows that the error can be reduced by selecting a matrix M, referred to as a preconditioning matrix or a preconditioner, such that

$$\overline{e}^k = M^{-1}\overline{r}^k, \text{ or } \quad \overline{e}^k = M^{-1}(\overline{b} - A\overline{x}^k). \tag{3.6}$$

Equation (3.6) implies that a good reduction in the solution error requires that the matrix M be "close" in some sense to the matrix A. In an ideal case $M = A$, $\overline{e} = 0$, but then the inverse M^{-1} is computationally as expensive to compute as finding the

A^{-1}. In practice (3.6) is solved approximately, with the choice of M as a trade-off between numerical and computational efficiency.

A systematic approach of using (3.6) to improve the accuracy of the current iteration $\overline{x}^k$ is to perform a splitting of matrix A as $A = M + N$, and construct the iteration

$$\overline{x}^{k+1} = \overline{x}^k + M^{-1}\overline{r}^k, \quad k = 0, 1, \ldots . \tag{3.7}$$

Equation (3.7) can also be written as

$$M\overline{x}^{k+1} = -N\overline{x}^k + \overline{b}, \quad k = 0, 1, ... \tag{3.8}$$

where M is a non-singular matrix that is easier to invert than A, and N is the remainder matrix. In addition, when M^{-1} and $-N$ are non-negative (a non-negative matrix is a matrix whose entries are non-negative [77, p.25]) this is known as a regular splitting iteration [77, p.115].

By representing the matrix splitting as $A = D + L + U$ where D is the diagonal, L strictly lower and U is the strictly upper parts of A, well known instances of the splitting iteration method can be introduced [77]:

$$\begin{array}{ll} M = D & \text{(Jacobi)} \\ M = D + L & \text{(Gauss-Seidel)} \\ M = \widetilde{L}\widetilde{U} & \text{(ILU(0))} \\ M = I & \text{(Richardson),} \end{array} \tag{3.9}$$

where $\widetilde{L}$ and $\widetilde{U}$ are lower and upper incomplete factors of A.

3.1.1 The Jacobi method

The Jacobi method is the simplest and the computationally cheapest simple-point iterative method [77, Chapter 4] [64, Chapter 5], where M is represented by the diagonal entries of A. The storage cost of implementing the Jacobi method is $O(n)$, and includes the storage for the vector $\overline{x}^k \in \mathbb{R}^n$, the storage for the updated vector $\overline{x}^{k+1} \in \mathbb{R}^n$ and the storage for $M = D \in \mathbb{R}^n$.

The component-wise formula for Jacobi updates is

$$x_i^{k+1} = \frac{1}{a_{ii}}(b_i - \sum_{\substack{j=1 \\ j \neq i}}^{n} a_{ij}x_j^k) \qquad i = 1, \ldots, n. \tag{3.10}$$

The nature of the Jacobi method, is that each update of the vector component in x_i^{k+1} relies only on the same previous component in x_i^k. This allows for simplicity during

parallel computation of the solution components (the method is highly parallel) [86, p.214].

The simplicity of the choice $M = D$ in many cases leads to a rather poor approximation of the matrix A. This results in slow convergence. The introduction of a damping constant allows some flexibility into the Jacobi method. If the Jacobi method converges then the damped Jacobi method will also converge [98, p.107]. However the opposite may not be true. Damping the Jacobi iteration can result in an increase in the efficiency of the method, compared to standard Jacobi. Given an appropriate damping constant $\gamma \in [0, 1]$, the damped Jacobi method is

$$\overline{x}^{k+1} = \overline{x}^k + \gamma D^{-1}\overline{r}^k, \qquad k = 0, 1, \ldots . \tag{3.11}$$

However the optimal value of γ for a particular problem is not known in advance. It is possible to tune the damped Jacobi method for certain problems and purposes. Also, it is possible to obtain an optimal value of γ based on Fourier analysis [93, Chapter 7, p.98] to make damped Jacobi effective in eliminating certain error components (see Section 3.3). A final point, is that the Jacobi method should be used on isotropic meshes, as a large aspect ratio can have a detrimental effect on convergence as the mesh width h decreases [86, p.215].

3.1.2 The Gauss-Seidel method

In the Gauss-Seidel method [77, Chapter 4][64, Chapter 5] we choose $M = D + L$ for (3.7), thus obtaining the following iteration

$$\overline{x}^{k+1} = \overline{x}^k + (D + L)^{-1}\overline{r}^k, \qquad k = 0, 1, \ldots . \tag{3.12}$$

The computational cost of a Gauss-Seidel iteration is that of a forward substitution. The storage requirements of the Gauss-Seidel method is $O(n)$, as only the lower triangular parts of A plus two additional vectors need to be stored.

The component-wise formula for Gauss-Seidel updates is

$$x_i^{k+1} = \frac{1}{a_{ii}}(b_i - \sum_{j=1}^{i-1} a_{ij}x_j^{k+1} - \sum_{j=i+1}^{n} a_{ij}x_j^k) \qquad i = 1, \ldots, n. \tag{3.13}$$

Gauss-Seidel benefits from an improved convergence, because for each iteration component the entry x_i^{k+1} is updated with the previously updated entries x_j^{k+1} for $i > j$ (3.13). This updating procedure is very useful for reducing the storage requirements, by overwriting the old values by the new ones.

The improved convergence characteristics of Gauss-Seidel compared to the Jacobi

method are attributed to the improved preconditioner M ($M_J = D$, $M_{GS} = D + L$). The "closer" matrix M becomes to A, consequently the "smaller" N becomes, the faster the method will converge [98, p.77]. However, calculating the M^{-1} is computationally more expensive for (3.12) than (3.11). A further attribute to Gauss-Seidel is that the order with which the updates is performed will also have an important consequence to the convergence [10, p.11]. In the Jacobi method the order of performing the updates is irrelevant.

The main drawback of the Gauss-Seidel method (3.12) is that it is inherently sequential [64, p.221], in the sense that a new iteration cannot start on some components of the solution before the current iteration is completely finished. Efficient parallelisation of this method involves subdivision of the unknowns into disjoint subsets, for example a red-black split of the unknowns into x_{2i+1} and x_{2i} (odd and even) components. The Gauss-Seidel method then applies concurrently to disjoint sets [64, p.222]. This decoupling will neglect all the non-zero entries in L that connect the unknowns from different sets. This will lead to a deterioration of convergence properties. As a final point the Gauss-Seidel method will perform better and to a larger aspect ratio than the Jacobi method, however the convergence rate will begin to deteriorate as the mesh width h decreases [86, p.215].

Line Gauss-Seidel method

For convection-dominated problems line Gauss-Seidel is an efficient solver, where the unknowns on the same grid line are relaxed simultaneously. However, the efficiency of the method as a solver, is dependent on the ordering of the nodes in the linear system [31, p.179, p.180]. Ordering in a downwind direction will improve the convergence rate [31, p.181].

For complex convection flow, such as recirculating wind or a wind that changes both its magnitude and direction throughout the domain, it is difficult to order the unknowns in the direction of the wind. To tackle this problem, multiple sweeps of line Gauss-Seidel can be performed along the lexicographical ordering of the nodes, namely bottom to top, left to right, top to bottom and right to left [31, p.191]. The disadvantage with line ordering is that it can be as expensive as using the ILU method. If line ordering was performed in all four directions the computational and storage cost grows by a factor of 4. However for the case studied in [31, p.192], there is a significant reduction in the iteration count for multiple-directional line Gauss-Seidel when compared to line Gauss-Seidel in one Cartesian direction.

3.1.3 Convergence of splitting iterations

By substituting (3.2) into (3.8) we get the relationship between the errors at two successive iterations $\overline{e}^k = (E_{amp})\,\overline{e}^{k-1}$, where the error amplification matrix E_{amp} is defined as

$$E_{amp} = M^{-1}(-N) = I - M^{-1}A. \tag{3.14}$$

The equation (3.14) indicates that a splitting iteration method will converge depending on the norm of E_{amp}, called the contraction number [93, p.38]. In (3.15) this error is bounded by powers ($\mathbf{k}$) of E_{amp} [84, p.231].

$$||\overline{e}^k|| = ||(E_{amp})^{\mathbf{k}}\overline{e}^0|| \leq ||(E_{amp})^{\mathbf{k}}||\,||\overline{e}^0||. \tag{3.15}$$

The condition $\lim_{k\to\infty} ||(E_{amp})^{\mathbf{k}}|| = 0$ will be satisfied if and only if $\sigma(E_{amp}) < 1$ [10, p.17], where $\sigma(E_{amp})$ is the modulus of the largest eigenvalue of E_{amp} (i.e. spectral radius). In the context of iterative methods, the spectral radius here is also known as the asymptotic convergence factor, and $-\log_{10}(\sigma(E_{amp}))$ is referred to as the asymptotic convergence rate [10, p.17] [93, p.41]. This is summarised in the following theorem.

Theorem 3.1.1 ([64]). *A stationary iterative method is convergent, if and only if the spectral radius $\sigma(E_{amp}) < 1$.*

Theorem 3.1.2 ([80]). *As $\sigma(\cdot) \leq ||\cdot||$ for every matrix norm $||\cdot||$, a sufficient criteria for convergence of an iterative method is $||E_{amp}|| < 1$ for any matrix norm $||\cdot||$.*

The speed of convergence, of splitting iterations, is directly related to size of $\sigma(E_{amp})$, where fast convergence is obtained when $\sigma(E_{amp})$ is close to zero [80]. There is also a direct relationship between diagonal dominance and convergence of the splitting iteration [73].

Theorem 3.1.3 ([64]). *If matrix A is strictly diagonal dominant (Definition 2.4.1) then (3.8) will converge for Jacobi and Gauss-Seidel.*

Furthermore, if A is irreducible[1] and is weakly diagonally dominant (Definition 2.4.2), except for a single row where strict inequality is satisfied, then the Jacobi and Gauss-Seidel methods converge [77, p.118] [98, p.108]. For non-normal matrices it is sometimes the case that $\sigma(E_{amp}) < 1 < ||E_{amp}||$ [38, p.28]. The practicality of this result is that the spectral radius does not determine the initial convergence rate, but it will determine the asymptotic convergence rate.

In the case when (Theorem 3.1.1) is true for the Jacobi algorithm, we can say that if A is ($a_{ii} > 0$ and $a_{ij} \leq 0$, $\forall\ i \neq j$) then A must be an M-matrix [64, p.212].

[1]Matrix A is irreducible if and only if the graph $G(A)$ is connected [77, p.26, p.89] [43, p.146].

Theorem 3.1.4 ([93]). *If A is an M-matrix (Definition 2.4.4) and M and N are formed through regular splitting, then (3.8) will always converge.*

To study the efficiency of Gauss-Seidel point iterative solvers, ordering of the unknowns is analysed in [24] [25] for the one-dimensional convection-diffusion equation. The resulting coefficient matrix is tridiagonal where A is banded by $tri[-b\ a\ -c]$ (with different values of coefficients). In the case when ordering of the unknowns is against the direction of the wind there is a lower bound of $||E^k_{amp}||_1 \geq (1-c^2)^{k-1}(1-c^{n-(k-1)})$, $k < n$. As $c \to 0$ (i.e. the problem is convection-dominated), $||E^k_{amp}||_1 \approx 1$. This indicates that the Gauss-Seidel method in this case will have a latency of $n-1$ before convergence. When ordering of the unknowns is along the flow of the wind there is an upper bound of $||E^k_{amp}||_1 \leq (1-b^n)(1-b^{n-1})^{k-1}$, favouring no latency in convergence. This latency is caused by non-normality in the matrix [84, pp.232–241].

3.1.4 Incomplete LU (ILU) factorisation

The Gauss elimination process is subdivided into two main stages: transforming the original system to an equivalent linear system, that has the same solution, but with a coefficient matrix which has an upper triangular form. The second stage is finding the solution by applying backward substitution. An LU factorisation is a numerically equivalent procedure to Gauss elimination. The idea is to decompose the original matrix into a product of a lower triangular matrix and upper triangular matrix. The upper triangular matrix in this decomposition is the same as an upper triangular matrix obtained by Gauss elimination, while the lower triangular matrix with a unit diagonal contains the elimination coefficients obtained during the elimination process [59, p.70, p.71] [80]. In the case of LU factorisation the solution of the system is obtained by the forward substitution followed by the backward substitution. However the computational cost of LU factorisation is twice as large as that of Gauss elimination (but both methods have the same asymptotic cost).

The LU factorisation of sparse matrices can lead to a considerable increase in the density of the computed matrix factors compared to the sparsity of the original matrix i.e. $\frac{nnz(L)+nnz(U)}{nnz(A)} \gg 1$, where $nnz(\cdot)$ is the number of non-zeros. This "fill-in" factor is a function of the problem size, ordering of the unknowns and density of the original matrix. As we move from 2D to 3D problems the density of the coefficient matrix A will increase, increasing the amount of "fill-in". This is the main hinderance that prevents direct methods from achieving optimal scaling when applied to sparse linear systems obtained from discretisation of PDEs.

Incomplete LU factorisation (ILU(p)) is derived from LU factorisation, where the

value of p controls the level of "fill-in" generated during factorisation. ILU factorisation is an approximate method for the solution of linear systems, where the preconditioning matrix M is taken to be the product of the incomplete factors $M = \widetilde{L}\widetilde{U}$. There is a trade-off between numerical efficiency and computational complexity in the algorithm.

In Algorithm 3.1 the entries of original matrix A are represented by $a_{i,j}$ initially. Upon completion of Algorithm 3.1, $a_{i,j}(i \leq j)$ will hold the elements of the factor $\widetilde{U}$ and $a_{i,j}(i > j)$ the elements of the factor $\widetilde{L}$, where the unit diagonal of $\widetilde{L}$ is implicitly assumed. The set P represents the zero entry pattern of the matrix A, which is used

Algorithm 3.1 Algorithm ILU [77, p.289]

```
Set a_{i,j} = 0, for each (i,j) ∈ P
for k = 1 : n − 1 do
  for i = k + 1 : n do
    if (i,k) ∉ P then
      a_{i,k} = a_{i,k}/a_{k,k}
      for j = k + 1 : n do
        if (i,j) ∉ P then
          a_{i,j} = a_{i,j} − a_{i,k} * a_{k,j}
        end if
      end for
    end if
  end for
end for
```

to control "fill-in". The only restriction is that the diagonal entries of A must be always present, that is: $P \subset \{(i,j)|i \neq j; 1 \leq i,j \leq n\}$ [77, p.289]. In the cases where the zero entries of A all belong to P we would achieve ILU(0). This is the simplest case where no "fill-in" is allowed. Standard LU factorisation would contain no zero-patten, therefore the conditional statements P in Algorithm 3.1 would be omitted.

The value of p, in a general algorithm $ILU(p)$, determines the rank of an element in the elimination process, from which "fill-in" is allowed. When $p = 0$, no "fill-in" is allowed, therefore the factors preserve the sparsity pattern of the original coefficient matrix. If $p = 1$ only the "fill-in" created by the original elements in the coefficient matrix is allowed, not the "fill-in" created by the already created "fill-in". For $p = 2$, the "fill-in" is allowed only if created by the original elements and the "first generation fill-in". Example 3.1.1 illustrates this point.

Example 3.1.1. A simple example of "fill-in" is taken from [64, Chapter 3.2], where

matrix A is structurally symmetric, and $\triangle$ represents the non-zero entries.

$$\begin{pmatrix} \triangle & \triangle & 0 & \triangle & 0 \\ \triangle & \triangle & \triangle & 0 & 0 \\ 0 & \triangle & \triangle & 0 & \triangle \\ \triangle & 0 & 0 & \triangle & 0 \\ 0 & 0 & \triangle & 0 & \triangle \end{pmatrix}.$$

The "fill-in" is defined by the symbol $\star$.

"first generation" "second generation" "third generation"

$$\begin{pmatrix} \triangle & \triangle & 0 & \triangle & 0 \\ 0 & \triangle & \triangle & \star & 0 \\ 0 & \triangle & \triangle & 0 & \triangle \\ 0 & \star & 0 & \triangle & 0 \\ 0 & 0 & \triangle & 0 & \triangle \end{pmatrix}, \quad \begin{pmatrix} \triangle & \triangle & 0 & \triangle & 0 \\ 0 & \triangle & \triangle & \star & 0 \\ 0 & 0 & \triangle & \star & \triangle \\ 0 & 0 & \star & \triangle & 0 \\ 0 & 0 & \triangle & 0 & \triangle \end{pmatrix}, \quad \begin{pmatrix} \triangle & \triangle & 0 & \triangle & 0 \\ 0 & \triangle & \triangle & \star & 0 \\ 0 & 0 & \triangle & \star & \triangle \\ 0 & 0 & 0 & \triangle & \star \\ 0 & 0 & 0 & \star & \triangle \end{pmatrix}.$$

The "fill-in" element at coordinate $(4, 3)$ in the second generation matrix is generated using the "fill-in" element at coordinate $(4, 2)$ produced during the elimination of the element $\triangle$ at coordinate $(2, 1)$. This would only be allowed if $p \geq 1$.

In the final generation an LU factorisation is produced:

$$L = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 \\ \triangle & 1 & 0 & 0 & 0 \\ 0 & \triangle & 1 & 0 & 0 \\ \triangle & \star & \star & 1 & 0 \\ 0 & 0 & \triangle & \star & 1 \end{pmatrix}, \qquad U = \begin{pmatrix} \triangle & \triangle & 0 & \triangle & 0 \\ 0 & \triangle & \triangle & \star & 0 \\ 0 & 0 & \triangle & \star & \triangle \\ 0 & 0 & 0 & \triangle & \star \\ 0 & 0 & 0 & 0 & \triangle \end{pmatrix}.$$

▷

By increasing p, the accuracy of ILU(p) preconditioner will increase. However, the increase in p will increase the computational complexity of factorising and applying the preconditioner, as the number of non-zero entries in the factors increases. For $p \to N$, ILU(p) becomes the standard LU factorisation. However it is difficult to know in advance how much "fill-in" $p > 0$ will produce. For each particular problem there exists the value of p that would give a method with the shortest execution time. However, an additional requirement of tagging the new "fill-in" elements to know which "generation" they belong to is needed. This can be alleviated if "fill-in" is based on the entries being larger than a certain threshold (ILUT) [12] [77, p.307].

Although, if $p > 0$, we need to worry about the storage for the additional "fill-in", an amount we do not known in advance. This makes memory management difficult. If the coefficient matrix is stored in sparse compressed row/column format, newly introduced elements would involve an intricate operation to preserve ordering of the elements. The use of ILU(0) does not involve any of these difficulties and we can expect the best execution speed and caching, due to a relatively small number of non-zero elements.

The level of "fill-in" in ILU(p) factorisation is dependent on the amount of non-zero entries in A as well as the non-zero patten (which is connected with the node numbering in a finite element mesh) [64, p.133]. In particular, the situation is expected to get worse (more "fill-in" generated) when discretisation is done by non-structured or adaptively refined grids and no reordering algorithm (such as minimum degree ordering) is applied. ILU(0) factorisation coincides with LU factorisation only in the case of banded matrices with no zero elements within the band (such as a tridiagonal matrix) or matrices resembling the right hand side in Example 3.1.2. The narrow banded matrices arises from a FEM discretisation of a 1D problem, but not in higher spacial dimensions.

Example 3.1.2. The two extreme cases of "fill-in" are;

$$\begin{pmatrix} \triangle & \triangle & \triangle & \triangle & \triangle \\ \triangle & \triangle & 0 & 0 & 0 \\ \triangle & 0 & \triangle & 0 & 0 \\ \triangle & 0 & 0 & \triangle & 0 \\ \triangle & 0 & 0 & 0 & \triangle \end{pmatrix} \qquad \begin{pmatrix} \triangle & 0 & 0 & 0 & \triangle \\ 0 & \triangle & 0 & 0 & \triangle \\ 0 & 0 & \triangle & 0 & \triangle \\ 0 & 0 & 0 & \triangle & \triangle \\ \triangle & \triangle & \triangle & \triangle & \triangle \end{pmatrix}$$

all zero entries are "filled in", no zero entries are "filled in".

▷

General methods (such as Cuthill McKee [77, p.82] and minimum degree ordering [17, Section 7.1]) are developed for sparse direct solvers for reducing the amount of "fill-in" by reordering (taking a permutation of) the matrix rows and/or columns. The Cuthill McKee algorithm is very similar to performing a breadth first search [77, p.81] where the adjacent vertices are traversed from lowest to highest degree. This produces a narrow banded matrix that reduces "fill-in". The minimum degree ordering algorithm reduces "fill-in" by ordering the pivots based on the node with the smallest degree, attempting to achieve the structure of the right hand side matrix in Example 3.1.2 prior to factorisation as best it can. Tarjan's algorithm [82] uses a depth first search to find strongly connected components using a stack and linking to potentially closed paths (the edges that connect the nodes at the top of the stack to

those further lower down the stack) [23, Section 6.8]. A strong connected component has no other nodes in its matrix row that do not belong in its closed path [23, p.114].

Stability is an important concept during LU factorisation as it is important for $\widetilde{L}\widetilde{U}$ to represent A as accurately as possible. Partial pivoting is a procedure whereby at each step, rows yet to be eliminated are interchanged to bring the maximum absolute value from the current column entries onto the diagonal [80]. For $\widetilde{L}\widetilde{U}$, it is possible to use partial pivoting as a technique to retain numerical stability (see Example 3.1.3).

Example 3.1.3. Matrix A demonstrates a loss of stability during LU factorisation [52, p.15]:

$$\begin{array}{ccc} A & \widetilde{L} & \widetilde{U} \end{array}$$

$$\begin{pmatrix} \pi & -1 \\ 1 & 1 \end{pmatrix} = \begin{pmatrix} 1 & 0 \\ \pi^{-1} & 1 \end{pmatrix} \begin{pmatrix} \pi & -1 \\ 0 & 1+\pi^{-1} \approx \pi^{-1} \end{pmatrix}.$$

If π is a small number then

$$A - \widetilde{L}\widetilde{U} = \begin{pmatrix} 0 & 0 \\ 0 & 1 \end{pmatrix}.$$

▷

Clearly the factorisation for Example 3.1.3 is unstable, even though the condition number of A is small $\kappa_\infty(A) = \frac{4}{1+\pi}$, unless partial pivoting is used[1]. As with Gauss elimination, partial pivoting provides numerical stability of the factorisation procedure. Chow et al. [13] has also suggested that column partial pivoting improves the numerical stability of ILU. However, partial pivoting will not be considered in the context of the ILU(0) algorithm as the sparsity pattern of the incomplete factorisation is fixed in advance. If A is an M-matrix (Definition 2.4.4) then the matrix obtained after applying the "first generation" step (see Example 3.1.1) of Gauss-elimination is also an M-matrix [77, p.288].

Theorem 3.1.5 ([52])**.** *If A is diagonally dominant by row (Definition 2.4.2), then Gauss-elimination without pivoting is stable.*

If the preconditioning matrix M is taken to be an ILU factorisation of $M = \widetilde{L}\widetilde{U}$, then we can write

$$A = \widetilde{L}\widetilde{U} + R, \tag{3.16}$$

[1]If $\kappa_\infty(A) = ||A||_\infty \cdot ||A^{-1}||_\infty \gg 1$ then A is ill-conditioned.

where $N = R$ is the residual matrix. The norm of the residual matrix is usually considered a good indicator to how close the incomplete factorisation is to the LU factorisation of the matrix A.

Theorem 3.1.6 ([62]). *If matrix A is an M-matrix and P in Algorithm 3.1 is static then unique factors $\widetilde{L}$ and $\widetilde{U}$ exist. Moreover (3.16) is a regular splitting.*

Theorem 3.1.7 ([62]). *If A is an M-matrix, then the construction of an ILU decomposition is at least as stable as the construction of a complete decomposition $A=LU$ without any pivoting.*

A generalisation of Theorem 3.1.7 by Manteuffel [60] later extended this property to H-matrices[2]. On the other hand, if the diagonal entries are not positive in matrix A, the construction of $\widetilde{L}$ and $\widetilde{U}$ may fail [63]. In cases where the diagonal entries are close to zero it is clear from Algorithm (3.1) that the factorisation process may break down, due to stability problems. Meijerink et al. [63] give a list of method modifications in order to improve stability. That is, if a diagonal entry is less than a given tolerance, during the construction of $\widetilde{L}\widetilde{U}$ (where $\widetilde{U} = \widetilde{L}^T$), then setting some of the previously constructed off-diagonal entries in the column of $\widetilde{U}$ to zero will recover stability. Alternatively, one may choose to increase the overall diagonal entries by adding εI to matrix A, where ε is chosen to be large enough. This is called a Manteuffel shift. The final modification, which is preferred by Meijerink et al. is to skip a (Gaussian elimination) column update if as a consequence the magnitude of the diagonal will be made smaller; this leads to a incomplete decomposition. The version of ILU(0) that is implemented in OOMPHLIB does not have a strategy for dealing with zero diagonal pivots.

It seems a strange concept to throw away, what may be useful information, in order to protect the sparsity pattern of a matrix. A recurring alternative to ILU(0) is modified ILU (MILU). This method makes one distinct change to the traditional ILU method shown in Algorithm 3.1, where the diagonal entries now compensate for the loss during zero "fill-in". That is, instead of ignoring the disallowed "fill-in" in Algorithm 3.1, the entries are added to the diagonal of the same row in matrix $M = LU$. This modification is shown in Algorithm 3.2. This method guarantees that the row sum of A is equal to the row sum of LU [77, p.305, p.306]. A generalisation of this methodology is to use a parameter $\omega \in [0, 1]$ that allows a degree of both standard ILU ($\omega = 0$) and MILU ($\omega = 1$) methods, called (RILU(ω)) [11] [95].

ILU(0) can be used as a preconditioner for a splitting iteration (3.7) by taking

[2] $(A = a_{i,j})$ is a H-matrix if $(B = b_{i,j})$ is an M-matrix where $b_{i,j} = -|a_{i,j}|$, $b_{i,i} = a_{i,i}$.

Algorithm 3.2 Algorithm MILU [20, p.206] [12]

```
⟨ insert ⟩
Set a_{i,j} = 0, for each (i, j) ∈ P
for k = 1 : n − 1 do
  for i = k + 1 : n do
    if (i, k) ∉ P then
      a_{i,k} = a_{i,k}/a_{k,k}
      for j = k + 1 : n do
        if (i, j) ∉ P then
          a_{i,j} = a_{i,j} − a_{i,k} ∗ a_{k,j}
        else
          a_{i,i} = a_{i,i} − a_{i,k} ∗ a_{k,j}
        end if
      end for
    end if
  end for
end for
```

$M = \widetilde{L}\widetilde{U}$ and $N = R$ [77, p.288, p.292] [93, p.47] with

$$\overline{x}^{k+1} = \overline{x}^k + \overline{y}^k, \qquad k = 0, 1, \dots, \tag{3.17}$$

where $\overline{y}^k$ is calculated using forward and backward substitution:

$$\begin{aligned} \widetilde{L}\overline{z}^k &= \overline{r}^k \qquad \text{(forward substitution)} \\ \widetilde{U}\overline{y}^k &= \overline{z}^k \qquad \text{(backward substitution).} \end{aligned}$$

Furthermore, the ILU(0) method can be implemented with a damping parameter to improve efficiency, as was done with the Jacobi method (3.11),

$$\overline{x}^{k+1} = \overline{x}^k + \gamma(\widetilde{L}\widetilde{U})^{-1}\overline{r}^k \qquad \gamma \in [0, 1]. \tag{3.18}$$

For the matrices that arise in FEM discretisation of PDEs, ILU(0) factorisation is an optimal algorithm, both in terms of memory requirement and algorithmic cost. This is a consequence of the fact that FE matrices always have $O(n)$ non-zero entries due to a constant number of non-zeros per row. However poor vectorisation properties of the algorithm and poor caching, due to the use of sparse data structures, and the sequential nature of the algorithm make its efficient implementation tricky in a parallel framework. Also the incomplete factorisation represents a considerable computational overhead compared to the back/forward substitution phase (despite the fact that in absolute terms both these phases have the same asymptotic computational cost). If we perform an ILU factorisation repeatedly as part of a complex

computation this can increase computational cost significantly.

ILU(0) is present as a basic routine in many software libraries: see for example `MI11` in HSL [1], `ParCSR Euclid` in Hypre [2], `IFPACK` and `AstecOO`Trilinos [3].

Parallel ILU

Parallelisation of a sequential algorithm usually needs to take care of two additional overheads: load balancing and communication between processors. Successful parallel algorithms assume approximately equal share of the computational load on each processor (without processors waiting for synchronisation), while parallel communication should be to a kept minimal. A further consideration is that efficient parallelisation of many algorithms require a substantial change to the original sequential algorithm.

Block Jacobi is a method for partitioning a matrix into block diagonal submatrices, where there is a one-to-one correspondence between submatrices and processors. Each processor/submatrix then applies a sequential iterative method to the linear system [77, p.378, p.379]. Hysom and Pothen [53] describe a parallel ILU (PILU) preconditioner that uses graph partitioning and an internal ordering strategy that minimises the dependence of the factorisation process on other subdomains. Each subdomain is free to choose a variant of ILU, however a level of "fill-in" is necessary with ILU(p) being the preferred ILU method in this paper. Hysom and Pothen show that in PILU(p) the way the matrix is subdivided does not affect the convergence properties of the algorithm. Furthermore, the paper compares PILU(p) and block Jacobi ILU(p) preconditioners. Their results show that the block Jacobi ILU(p) method converged in a slightly larger number of iterations, when the number of nodes per subdamain was larger than the number of subdomains, however the block Jacobi ILU(p) method always has a smaller ratio between the number of non-zeros in the preconditioner and the original matrix.

3.2 Krylov methods

The advantage of using simple-point iterations to solve sparse linear systems is that they have a small cost per iteration. However, these solvers require a large number of iterations to reach a given tolerance. Simple-point iterations, belong to a class of one-dimensional projection ($\overline{x}^{k+1} = \overline{x}^k + \overline{r}^k$, where $\overline{x} \in \text{span}\{\overline{r}\}$)[77, Sections 1.12, 5.1.1]. The different types of simple-point iterations (3.9) seen previously can be defined by introducing a preconditioner matrix M, which gives $\overline{x}^{k+1} = \overline{x}^k + M^{-1}\overline{r}^k$, where $\overline{x} \in \text{span}\{M^{-1}\overline{r}\}$ [38, p.2]. Krylov methods are iterative methods based on extracting the approximate solution of a linear system from the search subspace, referred to as the Krylov subspace [77, Chapter 6].

The residual reduction at every Richardson iteration (3.9) applied to the solution of linear systems (3.1), where $\overline{x}^0 = \overline{0}$, is

$$\overline{r}^{k+1} = (I - A)\overline{r}^k, \quad k = 0, 1, \ldots . \tag{3.19}$$

The Richardson iteration can be expressed in terms of residual vectors [87, p.22]:

$$\begin{aligned} \overline{x}^{k+1} &= \sum_{j=0}^{k} \overline{r}^j, \\ \overline{x}^{k+1} &= \sum_{j=0}^{k} (I - A)^{\mathbf{j}} \overline{r}^0, \\ \overline{x}^{k+1} &= P^k(A)\overline{r}^0, \end{aligned} \tag{3.20}$$

where P^k is a matrix polynomial of degree k with a value of 1 at the origin [38, p.4]. Therefore $\overline{x}^{k+1}$ belongs to the space

$$\overline{x}^{k+1} \in \quad \text{span}\{\overline{r}^0, A\overline{r}^0, \ldots, A^{\mathbf{k}}\overline{r}^0\} = \mathcal{K}^{k+1}(A, \overline{r}^0). \tag{3.21}$$

The space $\mathcal{K}^{k+1}(A, \overline{r}^0)$ is regarded as a $(k+1)$-dimensional Krylov subspace, generated by increasing powers of A applied to $\overline{r}^0$ [87, p.24, p.25]. The size of the Krylov subspace depends on the number of iterations. Moreover, each Krylov subspace method has the same search space (Krylov subspace) and thus forms the same polynomial approximation to the matrix inverse. The approximation solution $\overline{x}^{k+1}$ can be defined as

$$\overline{x}^{k+1} \in \overline{x}^0 + \mathcal{K}^{k+1}(A, \overline{r}^0) \quad \text{given} \quad \overline{r}^{k+1} \perp \mathcal{L}^{k+1}.$$

That is, the approximation to the solution $\overline{x}^{k+1}$ can be extracted from the Krylov subspace by imposing $(k+1)$ independent constraints. These are usually imposed by requesting that the current residual $\overline{r}^{k+1}$ is orthogonal to $\mathcal{L}^{k+1}$, the space spanned by the constraints. In comparison to the simple-point iteration methods, which has a constraint space limited to only one vector, the dimension of a Krylov subspace increases by one at each iteration. This fact will imply better convergence of Krylov methods, compared with simple-point iterations.

A general basis set of Krylov subspace $\mathcal{K}^{k+1}$ consists of non-orthogonal vectors. However, it is more desirable to have orthogonal basis sets as the Krylov algorithm exhibits better numerical stability and is less susceptible to round off errors. Thus, the main operation in each Krylov algorithm is to orthogonalise the Krylov basis set. At each iteration a new Krylov vector is orthogonalised with respect to the previous orthogonal basis. In the case of symmetric matrices this is done by the

Lancoz algorithm [77, p.187]. The advantage to this method is that orthogonalisation is needed only with respect to the previous two vectors. This implies optimal computational cost of this operation. For a non-symmetric matrix, the orthogonalisation needs to be performed with respect to all previous vectors. This is done by Arnoldi's algorithm [77, p.154] (which represents a modified Gram-Schmidt method [77, p.10]). This operation is non-optimal in terms of computational cost, and may represent a considerable computational overhead, if the iteration count is large (see Section 3.2.1).

It is possible to rearrange non-symmetric matrices to be amenable to the application of a Krylov solver aimed at symmetric systems. However the computational cost associated with this matrix transformation is not justified by the reduction in the solution time [88].

Krylov methods can be classified into four groups, with respect to the choice of subspace constraints ([87, p.25, p.26] [20, p.158]):

- Ritz-Galerkin: These are methods where at each iteration $\overline{r}^k \perp \mathcal{K}^k(A, \overline{r}^0)$ [87, Chapter 4.1]. A representative of this class is the Conjugate Gradient (CG) method [77, p.190].

- Minimum norm residual: Methods that belong to this category have the property, that $||r^k||_2$ is minimised over $\mathcal{K}^k(A, \overline{r}^0)$ [87, Chapter 4.2]. An example of a Krylov method that belongs to this class is GMRES [77, p.165].

- Petrov-Galerkin: Methods from this class produce at each iteration an approximation with residuals that are orthogonal to some space other than $\mathcal{K}^k(A, \overline{r}^0)$ [87, Chapter 4.3]. An example of a Krylov method that belongs to this class is Bi-CG [77, p.223].

- Minimum norm error: These are methods that at each iteration minimise the norm of the solution error $||\overline{e}^k|| = ||\overline{x}^k - \overline{x}||_2$ over the subspace $A^T\mathcal{K}^k(A^T, \overline{r}^0)$ [87, Chapter 4.4]. A representative of this class, for the symmetric case, is SYMMLQ [20, p.161].

Hybrids of these methods also exist, such as BiCGStab [87, p.26].

Krylov methods perform, in general, with greater consistency (with respect to the iteration count) than splitting iterations when applied to ill-conditioned systems obtained from the discretisation of PDEs. In exact arithmetic, the upper bound to the number of iterations for a Krylov method is equal to the size of the linear system. Therefore, the upper bound for the computational complexity of a Krylov method is $O(n^2)$. This is unacceptably high for practical applications. Some Krylov methods have an optimal computational cost per iteration, making them potential candidates

to be an optimal solver. To achieve this, a Krylov method needs to converge to a prescribed tolerance in a small number of iterations and (ideally) be robust.

3.2.1 The GMRES method

In this section we introduce the Krylov solver (GMRES method) that will be used in all our numerical tests.

GMRES is a Krylov method defined by the constraint space $\mathcal{L}^k = A\mathcal{K}^k$ and the standard Krylov subspace $\mathcal{K} = \mathcal{K}^k$, where $\mathcal{K}^k$ is the k^{th} Krylov subspace (3.21). For GMRES the Krylov space is constructed to be orthonormal through using the Arnoldi algorithm [38, p.38]. This process can be expressed in matrix form as

$$AV^k = V^{k+1}H^k, \tag{3.22}$$

where $H^k \in \mathbb{R}^{k+1\times k}$ is the Hessenberg matrix. A Hessenberg matrix is an upper bi-triangular ($h_{ij} = 0$ for any $i > j + 1$) matrix [77, p.155]. This matrix consists of the main storage overhead in the Arnoldi algorithm compared to the symmetric Krylov method based on the Lanczos algorithm where H is symmetric tridiagonal [77, p.186].

An approximate solution obtained by GMRES, $\overline{x}^k \in \overline{x}^0 + \mathcal{K}^k$, can be presented as an initial guess $\overline{x}^0$ and a vector formed from a linear combination of basis vectors $\mathcal{K}^k$. This can be expressed as

$$\overline{x}^k = \overline{x}^0 + V^k\overline{y}^k, \tag{3.23}$$

where $V^k \in \mathbb{R}^{n\times k}$ stores as its columns the orthogonal basis vectors of the k^{th} Krylov subspace and $\overline{y}^k \in \mathbb{R}^k$. The aim of the GMRES method is to compute at each iteration the approximate solution $\overline{x}^k$, which, for all vectors in the Krylov subspace $\mathcal{K}^k(A, \overline{r}^0)$ minimises the Euclidian norm of the residual $||\overline{r}^k||_2$. In order to approximate $\min\limits_{\overline{x}^k\in\mathcal{K}^k} ||\overline{r}^k||_2$, we solve approximately the least square problem for the vector $\overline{y}$ by minimising the functional $J(\overline{y}) = ||\beta\overline{e}^1 - H^k\overline{y}||_2$, [77, p.165]:

$$\begin{aligned}\min ||\overline{r}^k||_2 &= \min_x ||\overline{b} - A\overline{x}^k||_2 \\ &= \min_y ||\overline{b} - A(\overline{x}^0 + V^k\overline{y})||_2 \\ &= \min_y ||\overline{r}^0 - A(V^k\overline{y})||_2.\end{aligned}$$

Using (3.22), we can write

$$\begin{aligned}\min ||\overline{r}^k||_2 &= \min_y ||\overline{r}^0 - V^{k+1}H^k\overline{y}||_2 \\ &= \min_y ||V^{k+1}(\beta\overline{e}^1 - H^k\overline{y})||_2,\end{aligned}$$

where $\beta = ||\overline{r}^0||_2$ and $\overline{e}^1 \in \mathbb{R}^{k+1}$ is the first column vector of an identity matrix. Furthermore, the column vectors of V^{k+1} are orthogonal in $\mathcal{K}^{k+1}(A, \overline{r}^0)$; therefore

$$\begin{aligned}\overline{y}^k &= \min_y ||\beta\overline{e}^1 - H^k\overline{y}||_2 \\ J(\overline{y}) &= ||\beta\overline{e}^1 - H^k\overline{y}||_2,\end{aligned} \tag{3.24}$$

from the algorithm [77, p.165][38, p.39]. The process for minimising $||\beta\overline{e}^1 - H^k\overline{y}||_2$ is achieved by solving the least squares problem. The initial structure of matrix H^k is upper bi-triangular. After performing a Givens rotation [38, p.40] [77, p.167] matrix H^k becomes upper triangular. Then, performing a backsolve ($H^k\overline{y}^k = \beta\overline{e}^1$) will find the minimum (3.24).

To summarise, Algorithm 3.3 shows at each GMRES iteration the following tasks that need to be performed.

Algorithm 3.3 The GMRES algorithm [38, p.41]

- Apply the Anorldi algorithm.
  - Add the new column to the Hessenberg matrix H^k.
  - Orthogonalise the columns of H^k using (3.22).
  - Perform Givens rotations. This is a replacement of the Gram-Schmidt algorithm.
  - Normalise V^k if necessary.
- Perform a backsolve on H^k (to find the minimum of (3.24)).
- Correct the solution using (3.23).

The speed of convergence of GMRES is determined by the eigenvalue distribution of the coefficient matrix if A is close to Hermitian [39]. If the spectrum of eigenvalues is tightly clustered away from the origin, the convergence should be rapid. If the symmetric part (Definition 2.4.7) of the coefficient matrix $A = \epsilon D + C + S$ is positive definite ($H = \epsilon D + S$ in the convection-diffusion case) then the residuals generated by the GMRES algorithm satisfy [31, p.171]:

$$\frac{||\overline{r}^k||}{||\overline{r}^0||} \leq \left(1 - \frac{\lambda_{min}(H)^{\mathbf{2}}}{\lambda_{min}(H)\lambda_{max}(H) + \sigma(F)^{\mathbf{2}}}\right)^{\frac{\mathbf{k}}{\mathbf{2}}}, \tag{3.25}$$

where $F = C$ is the skew-symmetric part of A (Definition 2.4.8), $\lambda_{min}(H)$ is the smallest eigenvalue of matrix H, $\lambda_{max}(H)$ is the largest eigenvalue of matrix H and $\sigma(F)$ is the spectral radius of matrix F [31, p.171]. The general behaviour of GMRES is difficult to predict using eigenvalues alone [38, p.55] [39]. For non-normal matrices a tight clustering of eigenvalues around the value 1 may not be as beneficial as for normal matrices (i.e. may not lead to rapid convergence).

Table 3.1 presents the absolute value of the extreme eigenvalues λ_{min} and λ_{max} and the iteration count of the GMRES method applied to the solution of the discrete convection-diffusion operator, obtained by Q_1 SUPG FEM discretisation of the double-glazing [31, p.119] problem on a uniform quadrilateral grid, with increasing problem size and increasing Peclet number. The results show that as Pe increases the difference between the max and min eigenvalues grows rapidly. This growth of the condition number coincides with the increase in the iteration count.

Table 3.1: The absolute value of the eigenvalue limits for the discrete convection-diffusion operator obtained by SUPG, FEM discretisation of problem [31, p.119]. Also the number of GMRES iterations needed to reduce the initial residual by six orders of magnitude.

N	9			225			961		
Pe	Iter	$\lvert\lambda_{min}\rvert$	$\lvert\lambda_{max}\rvert$	Iter	$\lvert\lambda_{min}\rvert$	$\lvert\lambda_{max}\rvert$	Iter	$\lvert\lambda_{min}\rvert$	$\lvert\lambda_{max}\rvert$
80	9	1.1679	15.128	78	0.0765	8.1207	153	0.0192	4.8167
2,000	9	2.8322	349.23	184	0.0806	198.96	200+	0.0194	118.23
4,000	9	3.1181	697.27	197	0.0848	397.76	200+	0.0195	236.37
20,000	9	4.3001	3481.5	200+	0.1169	1988.1	200+	0.0206	1181.5
40,000	9	5.6882	6961.9	200+	0.1547	3976.1	200+	0.0219	2362.8

The convergence behaviour of GMRES is not sensitive to the nodal ordering in the case of convection flows. The use of SUPG discretisation to stabilise the solution leads to better conditioning of the resulting coefficient matrix. As a result stabilisation has a correlation to a good approximation, when using GMRES as a solver [35]; this is also the case for three-dimensional convection-diffusion problems [71].

The disadvantage of GMRES is that with each iteration the computational cost and memory storage increases, due to the need to store the Krylov vectors V^k and orthogonalise a new vector $A^{k-1}\bar{r}^0$ with respect to all previous Krylov vectors using the Arnoldi algorithm. One method to reduce this overhead is to restart GMRES after a number of iterations. This gives rise to the GMRES(m) method. The crucial question is when to restart. When comparing the computational cost of GMRES(m) (which takes a single matrix-vector multiplication per iteration) against BiCGStab or non-symmetric CG methods (which take two sparse matrix-vector multiplications per iteration) it may be the case that, on average, the cost of a single GMRES iteration is lower [34]. However the disadvantage to GMRES(m) is that convergence is no longer

guaranteed [78]. An alternative which is currently popular is to reduce the number of iterations needed for convergence through the introduction of a preconditioner.

3.2.2 Preconditioned GMRES method

The idea behind introducing a preconditioner is to transform a linear system (3.1) into an equivalent system (with the same solution) but with a coefficient matrix that has more favourable spectral properties, which should lead to a more rapid convergence of the Krylov iterative method [78] [34]. Systems that are easy to solve by the GMRES method are close to a normal matrix and have a spectrum that is tightly clustered, with the eigenvalues contained in an enclosed part of the complex plane with the spectral bounds away from the origin and independent of the discretisation and problem parameters [31, p.80, p.177] [38, p.120]. This implies that a Krylov method, applied to a transformed system, should converge to a given tolerance within a small and (almost) constant number of iterations. This, in turn, may result in an optimal solver for a particular class of problems. The practical aspects in achieving an optimal solver requires that the computational overheads associated with the preconditioning operator, that involve the assembly of the preconditioner and computing its inverse to a vector, need to have optimal cost and reduce the number of iterations (execution time) significantly compared to a non-preconditioned Krylov method [20, Chapter 9].

A preconditioned operator can be applied to a system (3.1) in several different ways. Here we present two possible ways: right-oriented or left-oriented (where M is the preconditioner):

$$\begin{array}{c|c|l} \textit{right oriented} & \textit{left oriented} & \\ A\overline{x} = \overline{b} & A\overline{x} = \overline{b} & --\text{ the original system} \\ AM^{-1}M\overline{x} = \overline{b} & M^{-1}A\overline{x} = M^{-1}\overline{b} & --\text{ the preconditioned system} \\ AM^{-1}\overline{z} = \overline{b} & & . \end{array} \tag{3.26}$$

The eigenvalue spectrum between the left and right preconditioners are identical, but the eigenvectors for the two variants may be different [87, p.175][77, p.271]. This may potentially lead to different convergence properties when matrix M is ill-conditioned [77, p.272].

The residual equation (3.5) implies that the connection between the error and the residual depends on the conditioning of the coefficient matrix A. If A is well-conditioned, $\overline{e}$ and $\overline{r}$ are closely related, that is a small residual implies a small error. On the other hand, if the coefficient matrix A is ill-conditioned and/or non-normal,

a small residual does not necessarily imply a small error. An interesting example of this relationship is demonstrated by Briggs et al. [10, p.8]. This property must be taken into consideration as the stopping criterion of an iterative method is based on the residual being smaller than a given tolerance. Projections of an error vector $||\overline{e}^k||$ to a non-orthogonal basis may produce small error components in the direction of the eigenvectors, even when the error vector norm is large. This is relevant in the case of the convection-diffusion problem, and will have an impact on the stopping criterion for Krylov solvers. However if $\overline{r} = 0$ then $\overline{e} = 0$, provided A is non-singular. The stopping criterion within GMRES is typically based upon the relative residual (3.4) or the absolute residual $||\overline{r}_k||$. Like (3.3) the absolute residual is scalar dependent.

In the case of left preconditioning the quantity that is computed is the preconditioned residual $M^{-1}\overline{r}$, therefore we do not have information about the residual itself (unless an additional matrix-vector multiplication with M is performed, which is not always possible in the case when MG is used as a preconditioner). If the preconditioned residual is used for the stopping criterion of the Krylov solver, the level of accuracy can potentially be compromised due to ill-conditioning of the preconditioner [77, p.271]. A further property of left-preconditioning is that the matrix $M^{-1}A$ is similar to a symmetric matrix if M is symmetric positive definite (see [20, p.196] for further details).

Right-preconditioning may also not result in a symmetric matrix if matrix A and M are symmetric. A right-preconditioner only affects the left-hand side of the equation. A stopping criterion based upon $||\overline{z} - \overline{z}_k||$ may have the effect of being considerably smaller than the norm of (3.2) which is equal to $||M^{-1}(\overline{z} - \overline{z}_k)||$ [20, p.196]. In cases when matrix M is ill-conditioned one may find a substantial difference between the two methods of preconditioning [77, p.271] .

In a preconditioned GMRES method, at each iteration a solution to the linear system $M\overline{z} = \overline{r}$ is performed. One possible indicator for the speed of convergence of the preconditioned algorithm is the condition number of the preconditioned matrix, $\kappa(M^{-1}A)$, see Definition 2.4.3. However, the estimates of the convergence rate based on the condition number alone are usually pessimistic. In [12] Chan et al. present that for a second-order elliptic problem, the use of ILU(0) preconditioning for the Krylov solver does not lead to asymptotic improvement in the condition number, i.e. $\kappa(M^{-1}A) \sim \kappa(A) = O(h^{-\mathbf{2}})$. Contrary to first impressions this is not the whole story. Practically, the eigenvalues distribution plays a key role in the iteration count of Krylov methods, see Vorst et al. [89] and Meijerink et al. [63] for the case of using incomplete Cholesky as a preconditioner. Better convergence of an ILU-preconditioned Krylov method is attributed to better clustering of the eigenvalues of $M^{-1}A$ than that of the matrix A [31, p.177]. For the case when two different

preconditioners lead asymptotically to the same condition number for $M^{-1}A$ the preconditioner that has better clustering of the eigenvalues would generally produce a lower iteration count.

To improve the asymptotic convergence rate: $\kappa(M^{-1}A) = O(h^{-1})$, for second order elliptic problems the following insert:

$$\left[\begin{array}{l} \textbf{for } k = 1 : n \textbf{ do} \\ \quad a_{k,k} = a_{k,k} + ch^2 \\ \textbf{end for}, \end{array}\right.$$

can be included in Algorithm 3.2 (where h is the mesh size and c is a parameter constant) [11] [20, p.206].

The Fourier analysis of RILU(ω), on a periodic second order elliptic problem defined on a unit square domain, was performed by Chan [11] and Donato et al. [19], where it was demonstrated that the condition number $\kappa(M^{-1}A)$ decreases as ω changes from 0 to 1. However, upon reaching $\omega = 1$ (the MILU case) there was a large increase in the condition number. Furthermore, Chan et al. [12] summerises that MILU is dependent on the matrix ordering, its existence may not be guaranteed for M-matrices and MILU has a poorer eigenvalue distribution than ILU for elliptic problems.

Table 3.2 presents the absolute value of the extreme eigenvalues λ_{min} and λ_{max} for the discrete convection-diffusion operator, obtained by Q_1 SUPG FEM discretisation of the double-glazing problem [31, p.119] on a uniform quadrilateral grid, when preconditioned with several general purpose preconditioners (Jacobi, Gauss-Seidel and ILU(0)) for different values of the problem size N and Peclet number Pe. The results from Table 3.2 indicate that ILU(0) is a better preconditioner than Gauss-Seidel or Jacobi. We also note that the quality of all three preconditioners deteriorate as N and Pe increase. Although for a fixed N it appears that $\kappa(M^{-1}A)$ behaves with $O(Pe)$ for all three preconditioners. Detailed Fourier analysis of the spectral properties of these preconditioners (motivated by [93, Chapter 7]) are deferred until the next chapter.

The introduction of a point iterative method as a preconditioner can help reduce the number of iterations of a Krylov solver, needed to reach a given tolerance, in comparison to a non-preconditioned version. In [77, p.172] Saad reports the iteration counts of solving discrete convection-diffusion problems [77, pp.95–97] using only GMRES as a solver. In [77, p.287] the same problems are solved using GMRES preconditioned by symmetric Gauss-Seidel leading to a reduction in the iteration counts by a factor of 2-3. Finally, in [77, p.294] the same problems are solved by GMRES preconditioned by ILU(0) leading to a further reduction in the iteration

Table 3.2: Spectral properties of the discrete convection-diffusion operator obtained from Q_1 SUPG FEM discretisation on a uniform tensor product grid of a unit square, for problem [31, p.119]. When preconditioned by three different general purpose preconditioners (Jacobi, Gauss-Seidel and ILU(0)) as a function of the discrete problem size N and Peclet number Pe.

N		9		225		961	
		$\lvert\lambda_{min}\rvert$	$\lvert\lambda_{max}\rvert$	$\lvert\lambda_{min}\rvert$	$\lvert\lambda_{max}\rvert$	$\lvert\lambda_{min}\rvert$	$\lvert\lambda_{max}\rvert$
Pe=80	Jac	0.1321	1.4713	0.0223	1.6861	0.0071	1.4671
	GS	0.2440	1.3974	0.0439	1.2952	0.0141	1.1962
	ILU	0.6512	1.0512	0.1781	1.1330	0.0598	1.1873
Pe=2,000	Jac	0.0223	1.5954	0.0015	2.2792	0.0007	2.4488
	GS	0.0440	1.4897	0.0031	1.5919	0.0014	1.6464
	ILU	0.1699	1.0396	0.0283	1.3674	0.0131	1.4378
Pe=4,000	Jac	0.0190	1.6418	0.0008	2.2978	0.0004	2.4898
	GS	0.0376	1.4901	0.0016	1.5994	0.0007	1.6725
	ILU	0.1477	1.0378	0.0163	1.3766	0.0079	1.5170
Pe=20,000	Jac	0.0143	1.7888	0.0002	2.3130	0.0001	2.5239
	GS	0.0283	1.4857	0.0004	1.6043	0.0002	1.6945
	ILU	0.1152	1.0265	0.0049	1.3796	0.0020	1.5552
Pe=40,000	Jac	0.0126	1.8443	0.0001	2.3149	0.0000	2.5282
	GS	0.0251	1.4836	0.0003	1.6048	0.0001	1.6974
	ILU	0.1038	1.0286	0.0033	1.3800	0.0011	1.5566

counts.

In cases of dominant convection, diagonal dominance of matrix A can be compromised. The use of ILU(0) as a preconditioner in cases where the matrix is indefinite, has large non-symmetric parts or when diagonal dominance is lost may lead to poor convergence [77, p.321].

Y. Feng et al. [34] compares three Krylov solvers preconditioned by ILU(0): BiCGStab, non-symmetric CG and GMRES(m). BiCGStab has a non-monotonic convergence characteristic, but converges in a smaller number of iteration than the other two methods. However, GMRES is the preferred method for convection-diffusion problems. That is, the overall convergence rate of GMRES may be lower than that of BiCGStab and non-symmetric CG, but it has consistently a smoother reduction in the relative residual norm. A further comparison is made between right- and left-preconditioning, with no conclusive results (there are advantages to each strategy in different cases).

Ordering the coefficient matrix before incomplete factorisation will have no beneficial effect in reducing the iteration count of the GMRES(m) algorithm, due to the lack of "fill-in" [77, p.334]. This is also the case reported in [5] for convection-diffusion problems. If some level of "fill-in" is allowed (for example, by using ILUT or ILU(1) method), the resulting preconditioner for the GMRES(m) solver performs

as Pe-robust, when reverse Cuthill McKee or Cuthill McKee reordering is used. For moderate ϵ^{-1} there seems no considerable advantage to applying Reverse Cuthill McKee or Cuthill McKee algorithms before the ILU(0) factorisation (the preconditioner obtained in each case shows similar iteration counts to that obtained with default ordering of unknowns). However, using the minimum degree ordering algorithm leads ILU(0) preconditioner closer to Pe-robust for convection-dominated problems (leading to a convergent iteration even when the ILU(0) preconditioner with other orderings fails) [5].

Van der Vorst [87, p.177] remarks that increasing the amount of "fill-in" as a means of increasing the efficiency of a preconditioner is not always advantageous, as it can sometimes lead to a spectral clustering close to the origin that deteriorates the efficiency of the solver.

An alternative to using the ILU method as a preconditioner for the discrete convection-diffusion problem is to use the line Gauss-Seidel method. In [31, p.193] the effectiveness of four-directional line Gauss-Seidel, two-directional ILU(0) and ILU(0) with default lexicographical nodal ordering as preconditioners to GMRES are compared, when applied to the double glazing problem. The results show that a multi-directional ordering results in a reduction in the number of iterations. Also, that a Krylov solver with a four-directional line Gauss-Seidel preconditioner converges in a smaller number of iterations than standard ILU(0) as a preconditioner.

Elman et al. [27] analyses the performance of line iterative methods for solving the discrete convection-diffusion equations with a variety of convective fields. The iterative methods used are ILU preconditioned GMRES and Gauss-Seidel, with an initial red-back ordering of the nodes ("natural ordering"). Using natural (lexicographical) ordering of the nodes a line reordering method for grid diagonals (also known as zebra relaxation [10, p.64]) is introduced. A further red-black ordering is then applied to the diagonal lines of the nodes ("red-black line ordering"). The results show that ILU preconditioned GMRES is close to $\vec{w}$, Pe-robust with no further benefit of introducing line red-black ordering. The results also show, that preconditioned GMRES is closer to $\vec{w}$, Pe-robust than using Gauss-Seidel as a solver.

Although using line or simple-point iterations leads to the reduction in iteration counts for certain classes of problems. In [31, p.194] it is demonstrated that for the double glazing problem a four-directional line Gauss-Seidel preconditioner for a GMRES solver leads to an increase in the iteration counts that is proportional to the problem size.

3.3 Multigrid method

All previously introduced methods belong to a class of single-grid methods. They are referred to as single-grid methods as they apply only to the coefficient matrix A obtained from the discretisation of a PDE on a given grid. The multigrid method belongs to a class of multi-level methods, which involves a hierarchical representation of the problem. The main components of a MG method are the smoother, the representation of the problem at each level and the means of communicating the information between different levels. Within MG the problem is solved iteratively by solving the residual equation (3.5) on each level using a smoother (relaxation) and using the correction to improve the current approximate solution. The methods described in Section 3.1 can be used as smoothers for MG methods.

3.3.1 Geometric multigrid method

The problem hierarchy for geometric multigrid is defined on a sequence of progressively finer grids. The continuous problem is discretised directly on each grid within the hierarchy. The idea of introducing a hierarchy of grid levels is introduced to rectify the inefficiencies that a simple-point iteration exhibits when applied to the discrete PDEs (smoothing of the error).

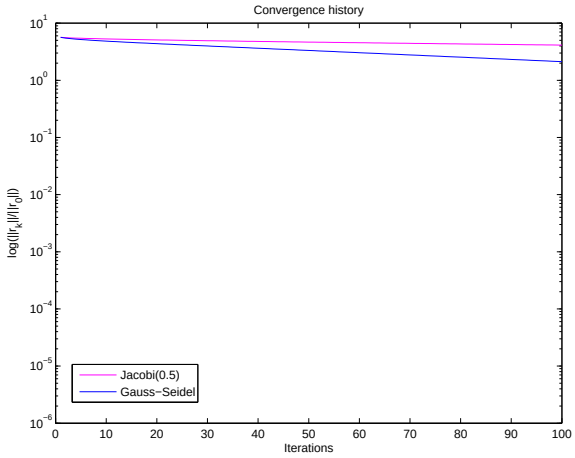
Error analysis of simple-point iterations by Fourier analysis

Consider the following one-dimensional model-problem:

$$\begin{aligned} -u'' &= f \qquad \text{in } [0,1], \\ u(0) &= u(1) = 0. \end{aligned} \tag{3.27}$$

Assume that the problem is discretised by finite elements using a uniform grid with $N+1$ elements (that is N unknowns). This results in a linear system (3.1) of size N, where the coefficient matrix A is tridiagonal.

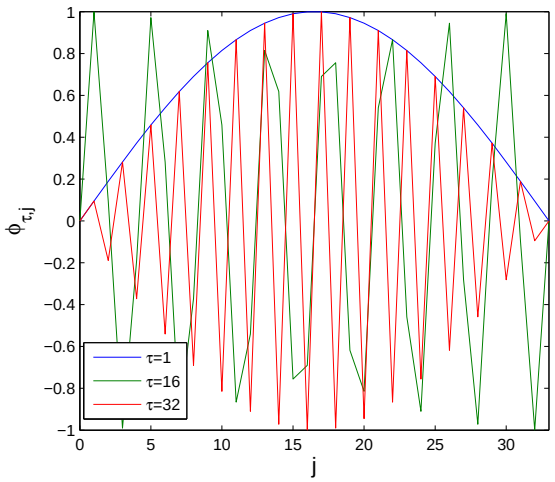
We want to solve this tridiagonal system by simple-point iteration. The convergence characteristic of a simple-point iterative method is very poor in this case. Looking at the convergence history in Fig 3.1, one can observe that after an initial reduction in the residual norm, the convergence effectively stalls and it takes a very large number of iterations until the system is solved to a suitable tolerance (note that ILU(0) has not been included in Fig 3.1, because for tridiagonal systems there is no "fill-in", therefore ILU(0) reduces to an LU factorisation). To give an insight into the problem, we are going to study the behaviour of the solution error (3.6) by decomposing it into discrete Fourier components $\overline{e}^k = \sum_{\tau=1}^{N} \xi_\tau \lambda_\tau^k(E_{amp})\overline{\phi}_\tau$ where $\xi_\tau \in \mathbb{R}$ is



(a)

Figure 3.1: Convergence history of the damped Jacobi ($\gamma = 0.5$) and Gauss-Seidel methods applied to the discrete model problem (3.27) with $f = 1$ and $N = 32$.

a control parameter for the amount of each mode in the error, λ_τ^k are the eigenvalues of E_{amp} and $\overline{\phi}_\tau$ are the discrete Fourier vectors with the components $\phi_{\tau,j} = \sin(\frac{j\tau\pi}{N+1})$ that form a basis of $\mathbb{R}^N$ [10, p.18, p.19]. For illustration of the Fourier vectors $\overline{\phi}_1$, $\overline{\phi}_{16}$ and $\overline{\phi}_{32}$ (see Figure 3.2) [10, p.13]. From Figure 3.2 it is clear that Fourier components



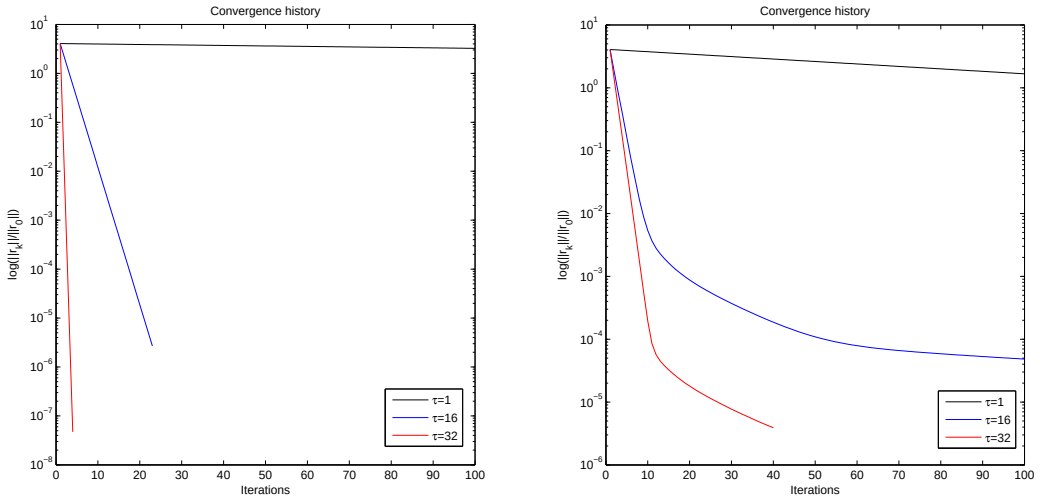
(a)

Figure 3.2: Fourier vectors $\phi_{\tau,j} = \sin(\frac{j\tau\pi}{N+1})$, $0 \leq j \leq N+1$ with wavenumbers $\tau = 1, 16, 32$ for $N = 32$.

with small wave numbers for τ correspond to slowly varying functions (smooth), while Fourier components with large τ correspond to rapidly changing (oscillatory) discrete functions. This is the reason for subdividing the Fourier spectrum into two parts:

the smooth part, that corresponds to Fourier vectors $\overline{\phi}_\tau$, $1 \leq \tau < \frac{N}{2}$ (low frequency) and the oscillatory part, that corresponds to Fourier vectors $\overline{\phi}_\tau$, $\frac{N}{2} \leq \tau \leq N$ (high frequency).

By solving a sequence of linear systems $A\overline{y}_\tau = \overline{\phi}_\tau$ [10, pp.12–25] [61, p.12], we see how the application of a simple-point iteration affects different Fourier components of the solution error. The convergence results for $N = 32$ are presented in Fig 3.3 [10, p.14], for three different values of $\tau = 1, 16$ and 32. From this we can conclude



(a) (b)

Figure 3.3: Convergence history of (a) damped Jacobi ($\gamma = 0.5$), (b) Gauss-Seidel methods applied to the reduction of different Fourier modes ($\overline{f} = \overline{\phi}_\tau$ with $\tau = 1, 16, 32$ corresponding to smooth, medium and high oscillatory modes) and N=32.

that the convergence of simple-point iterations is rapid when resolving oscillatory components of the error, but progressively slows down when the wavenumber τ is reduced and eventually stalls for smooth error components. The reason for such a behaviour lies in the fact that a simple-point iteration corrects the error by a small amount in cases when the residual is small compared to the solution error. That is, smooth error modes have small residuals and this leads to slow convergence of simple-point iterations. By contrast, high-frequency error components correspond to large residuals, which are effectively reduced by a simple-point correction [10, p.25]. This implies that simple-point iterations are effective in reducing oscillatory components of the error, i.e. they smooth the error. A simple-point iterative method cannot convert a mode with a certain wavenumber τ into a mode with a different wavenumber, but can change its amplitude [10, p.19]. Such a behaviour can be explained by the local nature of simple-point iterations (each error component at each iteration is updated

using only information about its nearest neighbour). To eliminate effectively low-frequency error components one needs to know global information about the error (this can be achieved by making distant nodes in the grid "communicate" directly).

The idea behind MG is to tune the simple-point iteration by applying an optimal damping parameter γ to maximise the reduction of high frequency errors. The low-frequency error components are reduced by solving (directly or iteratively) the residual equation (3.5) on a coarse grid. The concept behind this is that if an approximate solution of the linear system (3.1) is computed (through an iterative method) it can be improved by $\widetilde{x} = \widetilde{x} + \overline{e}$, (3.2). The solution error $\overline{e}$ can be computed by solving the residual equation (3.5). However, the problem is that solving (3.5) exactly is just as difficult as solving the original linear system (3.1). To overcome this difficulty, the residual equation is solved on a coarse grid and the correction projected back on the fine grid. There are, however, practical difficulties associated with the solution of (3.5) on a coarse grid. If the system is solved exactly (direct method), the computational cost of such a two-grid method may not be considerably smaller than the cost of solving the original problem directly. Furthermore, the asymptotic cost of such an algorithm is not optimal. An iterative solution of the coarse grid residual equation (3.5) would also suffer from a slow convergence, if simple-point iteration was used.

On a coarse grid (which in 1D has half as many Fourier vectors if uniform refinement is used) some of the smooth Fourier modes from a fine grid look sufficiently more oscillatory [10, p.32], which the application of a simple-point iteration will reduce effectively. Further reduction of smooth error modes is possible if this procedure is applied recursively. This leads to a multigrid method. In order to make the multigrid method work, we need to define a nested sequence of uniformly or adaptively refined grids $J_1 \subset J_2 \subset \ldots, \subset J_L$ where J_L is the finest and J_1 is the coarsest grid. The method initially solves $A_L\overline{x}_L = \overline{b}_L$ by a small number of simple-point iterations, giving an approximate solution $\widetilde{x}_L$, then computes the residual $\overline{r}_L = \overline{b}_L - A_L\widetilde{x}_L$. The recursive application of multigrid consists of restricting the residual to a sequence of progressively coarser grids and solving approximately (by simple-point iteration) the residual equation $A_l\overline{e}_l = \overline{r}_l$, at the grids for $l = L-1, \ldots, 2$. At the coarsest grid the residual equation $A_1\overline{e}_1 = \overline{r}_1$ is solved using a direct method. The size of the problem at level $l = 1$ is sufficiently small to make a direct solve feasible. The correction from the coarsest level needs to be projected successively through the grid hierarchy. Projection of the solution error between two consecutive levels tends to introduce new high frequency errors. For this reason a small number of simple-point iterations are applied to the residual equation, after each projection of the solution error. This procedure is referred to as a multigrid V-cycle (see Fig 3.4).

A multigrid method consists of five components:

- An initial matrix $A = A_L$ on the finest grid level.
- A smoother to relax the oscillatory components of the solution error.
- A nested hierarchy of uniformly or adaptively refined grids $J_1 \subset J_2 \subset \ldots, \subset J_L$ (J_L is the finest grid). Also the discretisation or algebraic equivalent A_l ($l = 1, ..., L$) of the continuous problem on these grids is needed.
- Intergrid transfer operators: restriction R_{l+1}^{l} and interpolation P_l^{l+1} ($l = 1, ..., L-1$) for transfer of information between successive grids.
- A solver (usually direct method) to solve the system $A_1\overline{e}_1 = \overline{r}_1$ on the coarsest grid.

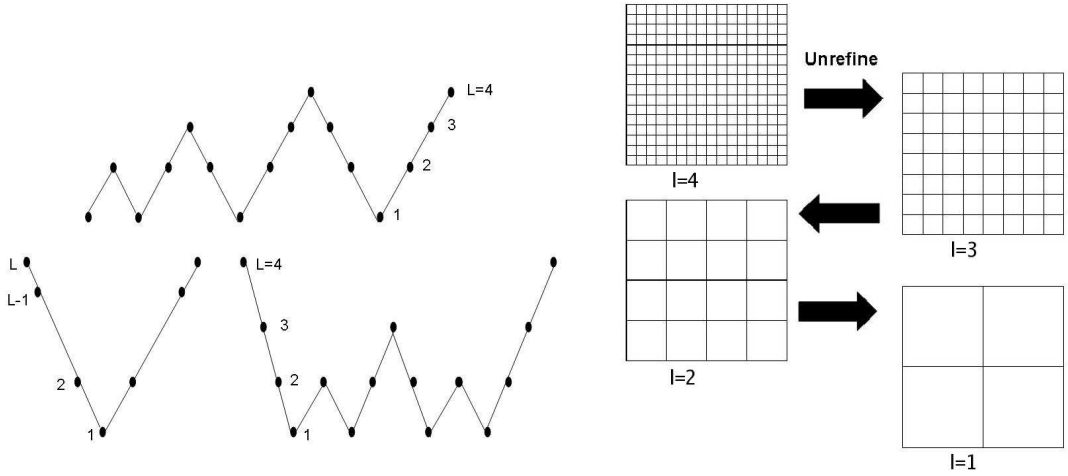
Let $S_l^{\nu}, (l = 2, \ldots, L)$ be the smoother operator where ν represents the number of iterations at a grid level l; $P_l^{l+1}(l = 1, \ldots, L-1)$ the interpolation operator (such that $P_l^{l+1} \in \mathbb{R}^{n_{l+1} \times n_l}$) that projects the correction to the solution $\overline{e}_l$ to the level $l+1$; $R_{l+1}^{l}(l = 1, \ldots, L-1)$ the restriction operator that transfers the residual $\overline{r}_{l+1}$ from a fine grid level $l+1$ to a coarse-grid level l. The MG V-cycle can be presented as:

$\widetilde{u}_L = S_L^{\nu_1}(A_L, \overline{f}_L)$	$\widetilde{u}_L = S_L^{\nu_2}(A_L, \overline{f}_L)$
$\overline{r}_L = \overline{f}_L - A_L\widetilde{u}_L$	$\widetilde{u}_L = \widetilde{u}_L + \overline{e}_L$
$\overline{r}_{L-1} = R_L^{L-1}\overline{r}_L$	$\overline{e}_L = P_{L-1}^{L}\overline{e}_{L-1}$
$\widetilde{e}_{L-1} = S_{L-1}^{\nu_1}(A_{L-1}, \overline{r}_{L-1})$	$\widetilde{e}_{L-1} = S_{L-1}^{\nu_2}(A_{L-1}, \overline{r}_{L-1})$
$\widetilde{r}_{L-1} = \overline{r}_{L-1} - A_{L-1}\widetilde{e}_{L-1}$	$\widetilde{e}_{L-1} = \widetilde{e}_{L-1} + \overline{e}_{L-1}$
$\ddots$	$\iddots$
$\widetilde{e}_2 = S_2^{\nu_1}(A_2, \overline{r}_2)$	$\widetilde{e}_2 = S_2^{\nu_2}(A_2, \overline{r}_2)$
$\widetilde{r}_2 = \overline{r}_2 - A_2\widetilde{e}_2$	$\widetilde{e}_2 = \widetilde{e}_2 + \overline{e}_2$
$\overline{r}_1 = R_2^1\widetilde{r}_2$	$\overline{e}_2 = P_1^2\overline{e}_1$
$A_1\overline{e}_1 = \overline{r}_1$	

(3.28)

Other standard MG cycles are W-cycle and Full multigrid (F-cycle) [38, p.194]. For a convection-diffusion problem, the preconditioned MG V-cycle is shown in [69] to have on occasions a smaller solve time compared to W- and F-cycles. The advantage of a simple V-cycle is that it is computationally cheap in comparison. What really matters in this case is to have an effective smoother and accurate coarse representation of the problem.

The solution of a linear system by multigrid consists of repeated application of a V-cycle described in (3.28) until the required convergence is achieved. This can be



(a) MG-cycles (b) Refinement

Figure 3.4: (a) Different types of MG cycles: F-cycle (FMG), V-cycle and W-cycle; (b) Several grid levels representing a MG hierarchy in the case of uniform grid refinement.

represented as the Richardson iteration of the form

$$\overline{x}^{k+1} = \overline{x}^{k} + MG_l(A, \overline{r}^{k}), \qquad k = 0, 1, \ldots \tag{3.29}$$

where $MG_l(A, \overline{r}^k)$ symbolises the use of a MG V-cycle, with l grid levels, applied to the approximate solution of the residual equation (3.5). A V-cycle is symbolised by $V(\nu_1, \nu_2)$, where ν_1 denotes the number of pre-smoothing iterations (the application of $S_l^{\nu_1}, l = 2, \ldots, L$) on each MG level, and ν_2 denotes the number of post-smoothing iterations (the application of $S_l^{\nu_2}, l = 2, \ldots, L$).

The computational cost of a MG solver applied to the solution of the discrete scalar second-order elliptic problem (Poisson problem), where $O(n)$ is the cost of a V-cycle, is $O(n \log(n))$ as $O(\log(n))$ iterations is needed to satisfy an error tolerance that is dependent on the number of levels [10, p.77]. In the case of the convection-diffusion problem the MG algorithm will perform robustly, provided that the relative contribution of the convection term remains small. In cases when convection is dominant, the application of MG as a solver will deteriorate. This is further emphasized by the profound impact that the structure of the convective field has to the MG convergence [70] [49] [69].

Interpolation in GMG

The prolongation (interpolation) matrix, P_l^{l+1}, is a means of communication between two successive grid levels. In standard interpolation procedure, an interpolation matrix is naturally connected with the finite basis set associated with the coarse-grid l. The entries of an interpolation matrix are calculated as the values of the coarse-grid FE basis functions evaluated at the fine grid points. Due to the locality of a FE basis the interpolation matrix is sparse.

If $\overline{u}_l \in \mathbb{R}^{n_l}$ is a discrete function defined on a grid at level l, then a discrete representation of this function at a grid level $l+1$, $\overline{u}_{l+1} \in \mathbb{R}^{n_{l+1}}$ can be obtained as $\overline{u}_{l+1} = P_l^{l+1}\overline{u}_l$, where $P_l^{l+1} \in \mathbb{R}^{n_{l+1}\times n_l}$ is an interpolation matrix with the entries

$$P_{ij} = \varphi_j^l(x_i, y_i) \qquad i = 1, \ldots, n_{l+1}\,, \quad j = 1, \ldots, n_l\,, \tag{3.30}$$

where φ_j^l is the global basis function associated with the node $j \in J_l$ evaluated at the node $(x_i, y_i) \in J_{l+1}$ [31, p.89] [10, pp.34–36]. The implementation for the interpolation matrix in OOMPHLIB is given in Appendix-B.6 (Algorithm B.1). The construction of the interpolation matrix based on FE basis set forms a partition of unity ($\sum_{j=1}^{n} \varphi_j(x_i, y_i) = 1$ for all $(x_i, y_i) \in \Omega$).

In the case of an adaptively refined grid that involves hanging nodes (depicted in red in Fig 3.5) construction of the interpolation matrix involves further recursive computation of the interpolation weights for a hanging node from the nodes which have non-zero basis functions at the position of the hanging node (see Appendix-B.6, Algorithm B.2).

The restriction matrix R_{l+1}^l may be constructed in a number of different ways. The simplest case is referred to as *injection* [10, p.35] (see Appendix-B.6, Algorithm B.3), where the nodes at a coarse grid directly take the values from the fine grid. An adaptation to this is the *sum injection* method (see Appendix-B.6, Algorithm B.4), where the nodes at a coarse grid directly take the values from the fine grid and are multiplied by the sum of the interpolation matrix column. A more sophisticated method is referred to as *full-weighting*. The *full-weighting* restriction operator is connected with the interpolation operator via [10, p.75]:

$$R_{l+1}^l = c(P_l^{l+1})^T, \tag{3.31}$$

i.e. the restriction and the prolongation matrix differ from each other by a constant c. This constant is 2^d in the case of finite difference discretisation [77, p.422] and 1 in the case of FE discretisation [65], where d is the spacial dimension. By taking the transpose of the interpolation matrix for the restriction matrix there is no additional

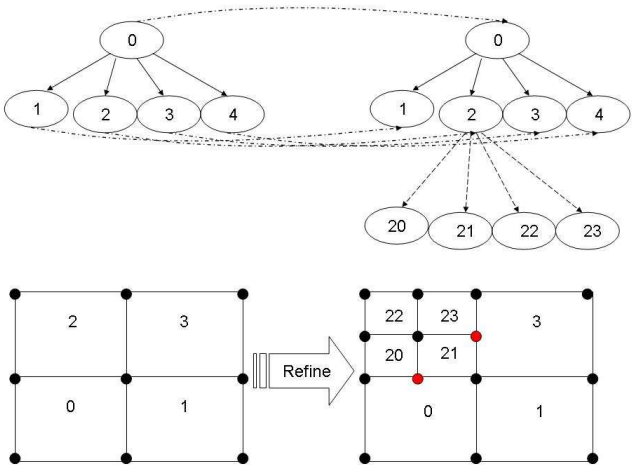


Figure 3.5: Tree data structure associated with adaptive refinement. In black are the regular nodes, and in red are the hanging nodes. The values of the solution at (red) hanging nodes are obtained as linear combinations of the neighbouring (black) master nodes.

computational and storage cost. Once the interpolation and restriction matrices are assembled, an alternative method of assembling the coarse-level matrices $A_l\,(l = 1,\ldots,L-1)$, is by Galerkin projection [77, p.423]:

$$A_l = R_{l+1}^l A_{l+1}\, P_l^{l+1}\,, \qquad l = 1,\ldots,L-1. \tag{3.32}$$

To allow interpolation to be effective it is necessary that the high frequency error modes are sufficiently smoothed, as aliasing within oscillatory error modes can lead to poor interpolation [10, p.35].

The variational properties (3.31) and (3.32) are used to assist in the analysis of the two-gird correction scheme $(MG_2(A,\overline{r}))$[10, p.75], to ensure its efficiency [93, p.90,p.91]. Here only the main result is quoted.

Theorem 3.3.1 ([95])**.** *The contraction factor of a two-grid correction scheme can be analytically bounded:*

$$\sigma(MG_2^{\nu_1}(A,\overline{r})) \leq ||MG_2^{\nu_1}(A,\overline{r})|| \leq ||A_{l+1}^{-1} - PA_l^{-1}R||\,||A_{l+1}S_{l+1}^{\nu_1}||.$$

Theorem 3.3.2 ([94])**.** *If A is symmetric, the V-cycle with ν pre and post smoothing sweeps is uniformly convergent:*

$$\sigma(MG_l^{(\nu,\nu)}(A,\overline{r})) \leq \frac{C_A}{C_A+\nu} < 1,$$

where $||A_{l+1}^{-1} - PA_l^{-1}P^T|| \leq C_A$ *(see Theorem 3.3.1).*

In Theorem 3.3.1, $MG_2^{\nu_1}$ denotes the action of a two-grid operator with ν_1 pre-smoothing iterations only, $A_{l+1}S_{l+1}^{\nu_1}$ is the smoothing property and $A_{l+1}^{-1} - PA_l^{-1}R$ is the coarse grid correction [93, p.91] [31, p.95, p.96]. From Theorem 3.3.1 it follows that the effectiveness of the two grid-correction scheme depends on the quality of a smoother (its ability to damp the high frequency error components at a fine grid level) and the ability of the coarse grid correction to remove the remaining components. The difficulty with the convection-diffusion case is to find a simple smoother (relaxation procedure) that damps effectively the high frequency error components for a wide range of parameters ($\vec{w}$, Pe and h) [81, p.30, p.31].

3.3.2 Algebraic multigrid method

The geometric multigrid method relies on the existence of an underlying grid hierarchy. There are a number of cases where the construction of such a nested grid hierarchy is difficult or even impossible (for example problems posed over complicated domains are frequently discretised by unstructured grids) [61, p.74]. Also problems with discontinuous or anisotropic coefficients can lead to a deterioration in the performance of the standard GMG method [77, p.437]. Algebraic multigrid (AMG) is an alternative to GMG which requires only the original coefficient matrix $A = A_L$ as an input. The coarse representation of that matrix, using (3.32), is created in an automatic coarsening procedure which is based on certain heuristic principles.

One of the differences between GMG and AMG is that; for GMG all coarse levels are predefined, and for AMG no geometric information is available therefore properties of the smooth error are used to construct an interpolation operator. In AMG the concept of algebraic smoothness is introduced, where an algebraically smooth error cannot be reduced significantly by the application of a smoother [77, p.438, p.439] [10, p.139] [61, p.84]. Previously stated was that smooth errors are characterised by small residuals, $A\overline{e} \simeq \overline{0}$. This can be used componentwise to define the interpolation procedure in AMG [77, p.438] [10, p.140] [61, p.85]. Given that $\overline{e}$ is an algebraically smooth error, the weighted average of its neighbouring error components e_j, can be used to approximate e_i.

$$a_{ii}e_i \approx -\sum_{i \neq j} a_{ij}e_j. \tag{3.33}$$

However (3.33) cannot be used directly to construct the interpolation operator. As the components of $\overline{e}$ change slowly (smooth) in the direction of strong connections [10, p.142] [61, p.85], it is important for interpolation to distinguish between strong and weak dependence [77, p.439]. A dependence in the ith equation is defined by:

if u_j is influential in determining u_i then point i depends on point j. For classical (Ruge-Stüben) AMG coarsening (where A is an M matrix) we define a set of points S_i that have a strong influence on i, for each equation i:

$$S_i \equiv \{i \neq j : -a_{ij} \geq \theta \max_{k \neq i}(-a_{ik})\}, \tag{3.34}$$

where θ determines the amount of influence which is acceptable, $0 < \theta \leq 1$. For 2D problems, in practice, $\theta = 0.25$ is used in accordance with [61, p.100] and for 3D problems $\theta = 0.5$ is a better choice according to [50].

By (3.34) each point i can be defined by its number of strongly connected neighbours j (i.e. the unknowns i is strongly influenced by).

The process of coarse grid selection is divided into two main stages. The first stage is to partition the set of all points $J_l \equiv F \cup C$, where C is a set of coarse grid points and F the set of fine grid points that strongly depend on it. This is achieved by finding the point with the largest number of influences (strong connections) and setting it to C. The points that depend on this point in C are set to F. All points that strongly influence the newly added points in F have their influence number (weight) incremented. This method is applied recursively until J_l is empty ($F = J_l \setminus C$). (see [15]). The second stage is to perform small adjustments to the subset F during construction of the interpolation operator in order to satisfy the following heuristics [61, pp.100–102].

Criterion 3.3.1 ([50][8]). *For each point $i \in F$, each point $j \in S_i$, j is either in C or strongly connected to at least one point in C_i ($C_i \subset C$), where $C_i = S_i \cap C$.*

Criterion 3.3.2 ([50] [8]). *The set of coarse points C should be a maximal subset of all points where no two points from C depend on the other.*

The two heuristics given by, Criteria 3.3.1 and 3.3.2, are used as a guide, with more emphasis on satisfying Criterion 3.3.1 [50].

For each point i in subset F we can divide its neighbours into three subcategories: coarse grid points with a strong influence on i (denoted C_i), coarse and fine grid points with a weak influence on i (D_i^w) and fine grid points with a strong influence on i (F_i^S). Given that the $\sum_{j \in D_i^w} a_{ij}$ is relatively small in comparison to a_{ii}, we can rewrite (3.33) as

$$(a_{ii} + \sum_{j \in D_i^w} a_{ij})e_i \approx -\sum_{j \in C_i} a_{ij}e_j - \sum_{j \in F_i^s} a_{ij}e_j, \tag{3.35}$$

where $e_j \approx \frac{\sum_{k \in C_i} a_{jk}e_k}{\sum_{k \in C_i} a_{jk}}$. As $\overline{e}$ changes slowly in the direction of strong connections, e_j is more accurate when j is strongly connected to points in C_i. [61, p.100].

Equations (3.33) and (3.34) are used to construct the interpolation operator. The interpolation operator can be defined as [10, p.143] [61, p.99]:

$$(P_l^{l+1}\, e)_i = \begin{cases} e_i & \text{if} \quad i \in C \\ \sum\limits_{j \in C_i} w_{i,j} e_j & \text{if} \quad i \in F\,, \end{cases} \tag{3.36}$$

where $w_{i,j}$ are weighting parameters. By rearranging (3.35) the weight w_{ij} can be calculated for (3.36) [10, p.144] [77, p.440, p.441]:

$$w_{ij} = -\frac{a_{ij} + \sum\limits_{m \in F_i} \left(\frac{a_{im} a_{mj}}{\sum\limits_{k \in C_i} a_{mk}} \right)}{a_{ii} + \sum\limits_{n \in D_i^w} a_{in}}.$$

However, classical AMG was developed for M-matrices. In cases when the off-diagonal entries of a_{ij} have the same sign as a_{ii} the interpolation formula becomes inaccurate this may lead to divergence [50]. The AMG algorithm BoomerAMG used in this work, developed as a part of the `hypre` package [2], addresses this problem by not allowing coefficients in a_{ij} that have the same sign as a_{ii} to contribute towards calculating a weight w_{ij} [50]. The formula (3.34) is modified so that if a_{ii} has the same sign as a_{ij}, j is not included in the set S_i.

The construction of the MG hierarchy for AMG is called the "setup phase" (see Algorithm 3.4). The above method, where coarse grid selection is based on the "strength

Algorithm 3.4 The "setup phase" for AMG [50]

Initialte $l = L$
do while(size(J_l) > 1)
  do (while J_l is not empty)
    Partition J_l into disjoint sets C^l and F^l via coarse grid selection
  end do
  Set $J_{l-1} = C^l$
  Construct the interpolation P_{l-1}^l
  Define the restriction $R_l^{l-1} = (P_{l-1}^l)^T$
  Apply Galerkin projection $A_{l-1} = R_l^{l-1} A_l P_{l-1}^l$
  Set $l = l - 1$
end while

of dependence" principle is referred to as classical AMG method. In the case of the Poisson problem discretised on a uniform tensor product grid the resulting coarsening is exactly the same as full coarsening used in GMG. In convection-dominated

problems the Ruge-Stüben method produces a variant of semi-coarsening in directions characterised by the convective field [97] [50] [10, p.151]. This contributes to the accuracy of interpolation and is the key to the success of the AMG method in convection-dominated cases.

The computational cost of the AMG method is difficult to predict in advance, as the matrices' size at coarse levels and the number of non-zero entries in the matrices are not known. However the costs can be estimated after the "setup phase" on a per problem basis [10, p.154]. There are several commonly used measures in this context.

Definition 3.3.1 ([10])**.** Grid complexity is defined as the ratio of the total size of discrete operators at all levels over the size of the original operator A_L:

$$C_G = \frac{\sum_{l=1}^{L} \text{size}(A_l)}{\text{size}(A_L)}, \tag{3.37}$$

where size(A) is the dimension of the square matrix A.

Grid complexity gives an indication of how fast the matrices are reducing in size [15]. In the case of uniform refinement with GMG full coarsening and direct discretisation of the coarse-level matrices, where at each coarser level the number of nodes is reduced in each spacial dimension by a factor of two, a grid complexity of $C_G = \frac{4}{3}$ and $\frac{8}{7}$ is expected for 2D and 3D problems respectively [10, p.154]. A large difference away from these values indicates that Criterion 3.3.2 has been violated significantly [8].

Definition 3.3.2 ([10])**.** Operator complexity is defined as the sum of the total number of non-zeros in the entire grid hierarchy over the number of non-zeros in the fine level matrix A_L:

$$C_A = \frac{\sum_{l=1}^{L} NNZ(A_l)}{NNZ(A_L)}, \tag{3.38}$$

where NNZ is the total number of non-zero entries in matrix $(\cdot)$.

The operator complexity (3.38) is an indicator of the storage costs for AMG. However, the storage for the interpolation matrices is not included in this measure. The solving phase of AMG is dominated by the application of the smoother and computation of the residual vector (essentially a matrix-vector product). These operations are directly proportional to the number of non-zero entries in matrix A_l. This implies that the operator complexity (3.38) also governs the computational cost of the solve phase. In the case of uniform refinement with GMG full coarsening and direct discretisation of the coarse-level matrices, $C_G \simeq C_A$. A feature of classical AMG

coarsening is that, for unstructured grids, the coarse grid operators tend to have more non-zero entries than the operator on the finest level (i.e. the coarse levels matrices, on average, have larger matrix stencils) [10, p.158]. This is indicated by a difference between the C_G and C_A. The problem is exaggerated further in 3D. A difference between these values also implies an increase in the density of the interpolation matrices increasing the computational cost of applying the interpolation. To reduce the value of C_A, the strength of dependence (3.34) threshold θ can be increased at a cost of reducing the effectiveness of the preconditioner [8].

Definition 3.3.3 ([8])**.** Average stencil size is defined as

$$C_S = \frac{1}{L}\sum_{l=1}^{L}\frac{NNZ(A_l)}{\text{size}(A_l)}. \tag{3.39}$$

The average stencil size C_S (3.39) is usually an optimistic estimate. This is due to the large difference in the stencil sizes between coarse levels. However, a large difference between the average stencil size C_S and the stencil size at the finest level (C_S when $l = L$) indicates that the density of the coarse level matrices has increased, in comparison to the original matrix [8].

In conclusion, AMG is found to be more expensive than GMG in terms of storage, computational cost and time [97].

We now introduce a measure for the truncation of coefficient matrices in the MG hierarchy. This measure is relevant for the newly introduce tILU smoother (see Section 3.4), where truncation is based on (3.34).

Definition 3.3.4. Truncation ratio is defined as

$$\eta = \frac{\sum_{l=1}^{L} NNZ(\widetilde{A}_l)}{\sum_{l=1}^{L} NNZ(A_l)}, \tag{3.40}$$

where $\widetilde{A}_l$ are the truncated matrices on all MG levels.

The truncation ratio is an indicator into the level reduction in the number of non-zero entries throughout the MG hierarchy. This ratio therefore is a direct indicator of the reduction in the storage requirements and the computational cost of the tILU smoother over the standard ILU smoother.

The largest advantage that AMG has over GMG is that it can be used as a "black-box" solver for a wide range of problems [97].

3.3.3 Parallel AMG

The parallelisation of FE computations is usually done using the domain decomposition method, which belong to the divide and conquer class. In this approach, the spacial domain is subdivided into overlapping or non-overlapping subdomains where all computations on each of the subdomains are associated with different processors. An equivalent of this approach at a matrix level is to partition the matrix into a number of submatrices and distribute these submatrices among parallel processors. All independent operations are then performed in parallel (with possibly some interprocessor communication), before the whole problem is reassembled [77, Chapter 14].

The "setup phase" of AMG is the most time-consuming. However, for classical AMG, the process of coarse grid selection is essentially sequential in nature. The simplest form of parallel coarse grid selection is to decompose the matrix A into a number of submatrices and apply Ruge-Stüben coarsening disregarding any connection (non-zero entries) across the interprocessor (subdomain) boundaries. Using Ruge-Stüben coarsening can create a particulary high concentration of C points close to the subdomain boundaries. As a result the Criterion 3.3.1 can occasionally be violated [50].

An alternative to local Ruge-Stüben coarsening is the Cleary-Luby-Jones-Plassman (CLJP) algorithm [14], which constructs coarsening independent of the number of subdomains, based on parallel graph-partitioning algorithms. Matrix S associated with (3.34) is now constructed of boolean variables such that $S_{ij} = 1$ if j influences i otherwise $S_{ij} = 0$. The CLJP algorithm initially defines the weight $w(i) = |(S_i^T)|+(\text{random number} \in [0,1])$, where S_i^T is the ith column of S. Let D be a set of points where $w(i) > w(k)$ for $k \in S_i \bigcap S_i^T$ (independent set of points). CLJP coarsening relies on the following two heuristics [14].

Criterion 3.3.3 ([50]). *The values of points in C are not interpolated. Neighbours that influence a point in C are less valuable as potential points in C.*

Criterion 3.3.4 ([50]). *If k and j both depend on $i \in C$, and j influences k, then j is less valuable as a potential point in C, because k can be interpolated from i.*

By complying to Criteria 3.3.3 and 3.3.4 the weights $w(j)$ that influence or depend on $i \in D$ are reduced to indicate that another influence has accounted for this point. When $w(j) < 1$, the point j is added to F. After Criteria 3.3.3 and 3.3.4 are applied to all points in D the remaining points are added to C. Each processor then performs a global communication to update w in the neighbouring processor boundary points. A new independent set D is then chosen and the process repeated until all points are in C or F.

An advantage to CLJP is that it is an entirely parallel coarsening strategy. Also, given that the same random numbers are used when selecting weights, the same coarse grids are selected regardless of the number of processors. The disadvantage with CLJP is that more coarse grid points are selected than is necessary compared to the Ruge-Stüben method, resulting in dense matrices A_l that leads to an increase in memory requirements and complexity of the solution phase [50].

A compromise between Ruge-Stüben and CLJP is called the Falgout coarsening. This is a hybrid method that applies Ruge-Stüben coarsening in the interior of each subdomain, to construct points in C that are not adjacent to the processor boundary points. The application of CLJP algorithm is then applied where the points in C are used as the first independent set of points in D, followed by recursive CLJP steps until all points are either in C or F. This process allows the CLJP algorithm to deal with the processor boundary points.

The iteration counts and solve times are useful measures of efficiency of a solver when only a single processor is used. As the number of processors increases, we expect the solve time to reduce, as the work load is shared between processors. The efficiency of a parallel solver must therefore also be based on its parallel scaling. The two common measures of parallel performance [20, p.61] are, speed up:

$$S_p = \frac{t_1}{t_p}, \tag{3.41}$$

where t_1 is the execution time of the algorithm with one processor and t_p is the execution time of the algorithm on P processors. Also, normalised speedup relative to the number of processors is referred to as parallel efficiency:

$$E_t = \frac{t_1}{P\, t_p} \qquad E_t \in [0, 1], \tag{3.42}$$

where optimal scaling, implies $E_t = 1$.

3.3.4 Literature summary of multigrid

Using ILU(0) as a solver or preconditioner for the discrete convection-diffusion problem is ineffective (see Table 3.2). However, for a wide range of problems the smoothing properties of ILU(0) are good [85, p.259]. Turek [86, p.57] also compliments the use of ILU on the convection operator stating that it leads to resilient ("robust") convergence when used as a smoother. Furthermore, for complex mesh geometries Turek [86, p.216] favours ILU as a resilient ("robust") "black-box" smoother. This makes the method a suitable alternative to a point smoother for multigrid.

Trottenberg et al. [85, pp.257–268] state that ordering of the nodes for the same

grid can have a considerable impact on the effectiveness of an ILU(0) smoother, due to different stability properties and different amounts of "fill-in" that is generated by different orderings of the same matrix.

In [94] Wittum proves the convergence of ILU(0) as a smoother for the MG V-cycle in the case of an anisotropic diffusion problem (Theorem 3.3.2). The anisotropic diffusion problem used in this paper extends the proof of convergence of the MG V-cycle to include negative eigenvalues in the iteration matrix E_{amp} that are found when an ILU smoother is used.

In [96] a truncated ILU smoother is presented, based on reducing the stencil size in the coefficient matrix by taking a particular grid block, and standardising the boundary of the block to represent the remainder of the grid domain. This results in a reduction in the storage and computational requirements of factorisation (due to repetition of the matrix entries).

Kettler and Wesseling [56] analyse the use of a block ILU smoother for standard GMG. Their analysis was performed on anisotropic diffusion and convection-diffusion problems in three dimensions. This was a continuation of the work from [55], where a block ILU smoother is used in two-dimensions. The motivation behind the paper was to develop a "black-box" MG method, in three dimensions, for cases with complex flow patterns. In the case of an anisotropic diffusion problem, when diffusion is dominant in a certain plane, using a plane Gauss-Seidel smoother aligned with the dominant plane is needed to satisfactorily smooth the error. Therefore if a dominant plane has a variable orientation, alternating plane Gauss-Seidel needs to be used at a considerable computational cost compared to block ILU method. However, Kettler et al. propose, as a substitute to alternating plane Gauss-Seidel smoothing, the use of a Krylov method to reduce the high frequency modes that block ILU has missed; as opposed to dealing with a dominant plane through semi-coarsening. It was also found that if a directional plane Gauss-Seidel smoother sweeps in the opposite direction to the flow, the method would break down.

In [47] Hemker shows by means of Fourier analysis and amount of work per iteration, that ILU(0) is a better smoother for MG than modified ILU(0) (see Algorithm 3.2) or symmetric Gauss-Seidel method for convection-diffusion problems. In [48] Hemker performs efficiency and Fourier analysis of a two-grid solver with either ILU(0) or line Gauss-Seidel smoother. The analysis demonstrate superiority of ILU(0) smoother for the convection-diffusion problem.

Wesseling [92] compares a MG method with an ILU smoother against different types of MG cycles. Wesseling reports that ILU is a "better" smoother than Gauss-Seidel, for the diffusion problems considered. In [91] a "black-box" MG solver with ILU(0) smoother (MGD1) is proposed. In this paper the use of ILU(0) smoother is

preferred to other Gauss-Seidel type smoothers.

The limitations to using Fourier analysis on MG is that it is restricted to a two-grid cycle. This may not reflect accurately the performance of a MG V-cycle, due to the introduction of a sequence of coarser grids which have not been considered in the analysis [49]. In [49] it has been demonstrated that effectiveness of ILU smoother deteriorates when diagonal dominance of the coefficient matrices is lost. This can be the case for the convection-diffusion problem. However, Kettler et al. still regards ILU as a suitable smoother for convection-diffusion and proposed modifying the ILU algorithm to block ILU.

For convection-dominated problems ordering the unknowns in the direction of the convection flow is a common approach to improving the convergence rate of Gauss-Seidel type smoothers for discrete convection-diffusion problems [24] [25]. Bey and Wittum [6] further developed this concept and produced a downwind numbering strategy for Gauss-Seidel as a smoother to MG for adaptively refined grids. Wang and Xu [90] use a block Gauss-Seidel method where the blocks are ordered in a crosswind direction, by using Tarjan's algorithm [82]. However their block method performs poorly as a smoother to MG.

Wu et al. [97] analyzed both GMG and AMG, using line and ordered Gauss-Seidel as smoothers, on two different convection-diffusion problems (equivalent to model problems Example 3.1.3 and 3.1.4 from [31, p.118, p.119]). The first problem is solved using both MG methods where AMG is theoretically and numerically shown to converge faster than GMG under certain conditions. MG however, as a solver, for the second problem is not sufficient.

For anisotropic diffusion problems the convergence characteristic of MG as a solver does not depend on h, but will depend on the anisotropy parameter ϵ [38, p.196]. This is also the case for convection-diffusion problems when $h \ll \epsilon$, however as the problem becomes more convection-dominated GMG is found to lose its h-independence [97]. To deal with this problem appropriate prolongation, restriction and smoothers can be used [18]. However, problem specific solvers may not always be the most appropriate solution, as the properties of the problem may not be known in advance. Greenbaum [38, p.197] therefore suggests that multigrid be treated as a preconditioner to Krylov methods. By using MG as a preconditioner to a Krylov solver we are essentially accelerating the convergence of MG. Also, we obtain a more robust solver than when using MG as a stand alone solver.

3.3.5 Multigrid preconditioning of Krylov solvers

"The reliability of iterative techniques, when dealing with various applications, depends more on the quality of the preconditioner than on the particular Krylov subspace accelerators used" [77, p.261]. This is further justified in papers by Faber, Manteuffel and Voevodin where they found that an algorithm will have a "short solution" only if the matrix is Hermitian or skew Hermitian [78]. In the previous section point iterative methods were introduced as preconditioners. If quality of the preconditioner is key, in order to design an optimal solver for a given problem, we need to consider more sophisticated preconditioning techniques than simple-point iterations.

In [97] MG is used as a preconditioner to GMRES. The overall performance of AMG, on uniform and adaptive mesh refinement, is robust and for both problems found to converge faster than GMG. However, GMG is shown to be a near $Pe, \vec{w}$-robust preconditioner to GMRES on uniform and adaptive mesh refinement.

In order to make MG work efficiently as a preconditioner for discrete convection-dominated problems, some adjustments in the coarsening strategy and/or smoothing operator needs to be made [70].

Goldstein [36] proposed to map the original equation to one that will result in a symmetric positive definite stiffness matrix [77, p.245, p.252], solve this problem using MG preconditioned conjugate gradient, and map the solution back to the original problem. Goldstein also considered using Petrov-Galerkin (SUPG) approximation. Standard MG V-cycle is used with a combination of symmetric red-black Gauss-Seidel smoothers. The main disadvantage with mapping the problem is that it required a parameter which is dependent on Pe and h.

Standard Galerkin approximation of the convection-diffusion problem leads to oscillatory solutions on coarse grids. This effect reduces the effectiveness of MG as a preconditioner. To solve this problem, in [72] it is proposed that the SUPG method is applied with an appropriate level of stabilisation to the discretisation of the convection-diffusion problem at all levels of the MG hierarchy. As we are interested in the solution of the problem at the finest grid, the stabilisation on coarse grids will not have any adverse effects on its accuracy. Moreover, the stabilisation on coarse grids makes MG a more effective preconditioner. In [72] the effect of a stabilised MG preconditioner for GMRES is studied on increasingly stretched grids. The testing performed by Ramage is based on a single MG V-cycle as a preconditioner to GMRES. The smoother used is line Gauss-Seidel with one horizontal and one vertical sweep per MG level for pre- and post-relaxation step. Ramage also studied the effect of using different interpolation procedures, a bilinear interpolation and a De Zeeuw interpolation [18]. The results suggest that bilinear interpolation is sufficient to achieve adequate convergence, provided stabilisation and (computationally

expensive) alternating line smoothers are used.

Oosterlee and Washio [70] use a Krylov solver preconditioned with MG for solving the discrete convection-diffusion equation. A further improvement to the Krylov solver is achieved by introducing Krylov acceleration within the MG cycle. The best results in [70] are observed when a Krylov accelerator is applied at level $L-1$ after a post-smoothing step. The application of multiple Krylov acceleration steps at multiple MG levels does not lead to a substantial improvement in convergence.

Syamsudhuha [81, pp.55–60] compares two different smoothing techniques (ILU(0) and point Gauss-Seidel) for MG preconditioned GMRES solvers applied to the convection-diffusion problem (with uniform wind and recirculating wind) discretised on a uniformly refined grids. Although ILU(0) smoothing leads to a reduction in the iteration counts, the computational cost of performing ILU(0) factorisation at all MG levels and applying the smoother is expensive. The convergence tests are performed both with lexicographical and downwind ordering of the nodes (see [81, p.46]). The downwind ordering led to a substantial reduction in the iteration counts in comparison to lexicographical ordering, for both smoothers (with an exception in the case of ILU(0) smoothing for the double glazing problem, where the solver diverged). Also, Syamsudhuha shows that GMRES preconditioned by MG, gives more favourable results than using MG as a solver.

3.4 A new tILU smoother

In [9] GMRES preconditioned by classical AMG with a range of smoothers is applied to a convection-diffusion problem, discretised using SUPG. The MG preconditioner that proved to be the most robust, in this context for the cases of strong variable vertical wind [31, p.117] and recirculating wind [31, p.119], involves alternating line Gauss-Seidel method on the finest level and damped ($\gamma = 0.5$) Jacobi smoother on the coarse levels. This means that a computationally expensive smoother is applied at the finest level and an inexpensive smoother to all coarse levels. The idea behind tILU is exactly the opposite to this hierarchical smoothing strategy.

For a singular perturbation of a second-order elliptic problem in [95] it was shown that ILU(0) is an exact solver in the limiting case of pure convection. Also it was determined that the convergence rate of MG W-cycle with ILU(0) smoother deteriorates slightly when the grid is refined, but tends asymptotically to a constant for each fixed value of ϵ.

From the matrix stencils introduced in Section 2.4 it can be concluded that, due to the h-dependence of the convection matrix, its relative contribution to the overall discrete convection-diffusion matrix increases when h is increased (i.e. the

contribution is larger at coarse grids). In a MG setting this means that the convection part becomes a more prominent factor in the overall coefficient matrix at coarser levels. Assuming the incompressibility of the convective field (i.e. skew-symmetry of the convection matrix), the diagonal dominance of the discrete convection-diffusion operators can be completely lost (see Table 2.1). This would make point smoothers progressively more ineffective on coarse MG levels. This intuitive finding suggests that a computationally cheap smoother can be sufficient at finer levels and a robust (and computationally more expensive) smoother is needed only at a few coarsest levels.

In practice, it would be difficult to determine for each particular case a cutoff MG level from which an application of a more complex smoother is necessary. Instead, we propose a type of variable smoother which adapts itself to the strength and characteristics of the convective field, represented locally by matrix stencils at each grid node, and becoming progressively more complex towards coarser MG levels (where the cost of its assembly and application is reasonable). The variable effect is achieved by static analysis of the matrix entries before the application of the smoother. This analysis will produce information of which off-diagonal entries are "relevant" in each matrix row, thus capturing the local information about the convective field that may be missed by the standard point smoothers. The criterion for determining which off-diagonal entries are considered to be "relevant" (significant in magnitude) is the same criterion that quantifies the strength of dependence principle in AMG (see (3.34)). In a truncated matrix, for each row we keep the diagonal entry and all off-diagonal entries that are "relevant", i.e. those with the absolute value that is above a given threshold determined by (3.43):

$$|a_{ij}| > \alpha \max_k |a_{ik}|, \qquad \alpha \in [0,1]. \tag{3.43}$$

The truncation level is controlled by the parameter α, this is the only parameter in the method. The smoother is then defined by applying the incomplete LU factorisation with no "fill-in" to the truncated matrices at all MG levels. We denote the resulting method as $tILU_0(\gamma, \alpha)$, where γ is the damping parameter. The new smoothing technique reduces, in two extreme cases, to the standard ILU(0) method for $\alpha = 0$ (no truncation), and for $\alpha = 1$, to the Jacobi method (where all off-diagonal entries are neglected).

The proposed method is hence a trade-off between a computationally cheap but potentially numerically ineffective Jacobi method, and a robust but computationally expensive ILU(0) method. Due to the h-dependence of the convection matrix, the magnitude of the off-diagonal entries will become larger at coarse levels. The new

smoother adapts automatically to this situation by truncating the majority of the off-diagonal entries at a few of the finest levels, thus resembling the Jacobi method, whilst keeping progressively more off-diagonal entries at the coarse levels, thus resembling the ILU(0) method (where the application to relatively small coarse matrices is not sufficiently expensive). The amount of off-diagonal entries kept at a particular MG level will depend on the discrete problem size at that level and on the strength and the structure of the convective field, where it is anticipated that more off-diagonal entries will be kept in strongly convective cases with complex winds (see Tables 4.9 and 5.5).

The new smoothing method introduces a single parameter α that controls the level of truncation at all MG levels. Although this fact may have a decremental effect with regard to the effort of building a robust and effective "black-box" MG preconditioner for a wide range of convection-dominated problems, we found by extensive testing and Fourier analysis that for a wide range of test problems, a fixed parameter value of $\alpha = 0.25$ for GMG and $\alpha = 0.5$ for AMG give the best results in term of the execution time, for problems in two and three spacial dimensions. These values are standardly used in the context of AMG.

If no partial pivoting is allowed in ILU, diagonal dominance of the underlying coefficient matrix is essential to ensure numerical stability of the algorithm (Theorem 3.1.5). As the Pe increases diagonal dominance is preserved for longer in the truncated coefficient matrix than in the original coefficient matrix. This implies, by Theorem 3.1.5, that ILU factorisation of the truncated coefficient matrix is more stable than the original coefficient matrix. The reason for this is that in cases when one of the off-diagonal entries in a row is larger than the diagonal entry, diagonal dominance is surely lost. However if the off-diagonal entries are all smaller than the diagonal entry then it is important to look into the sum of the off-diagonal entries. By truncating all the small entries there is a smaller chance of losing diagonal dominance. Therefore the truncation procedure, though it reduces the accuracy of the smoother, does increase the stability of factorisation by retaining the property of diagonal dominance for larger values of Pe.

In the following chapters we test the MG preconditioner with the new tILU_0 smoother on a range of different problems, both in 2D and 3D, with progressively more complex structures of the convection field (which make them progressively harder to solve by a MG preconditioned iterative solver). We start our numerical tests by determining experimentally the optimal value of the truncation parameter α when using the tILU_0 smoother for GMG. The preconditioned iterative solver that showed the shortest execution times and performed the most consistently on a range of problems and discretisations is $\alpha = 0.25$, as a result we adapt this to be the default value in

all our experiments with GMG. A continuation of the experiments is then performed using an AMG preconditioner with the tILU_0 smoother. The optimal value of the truncation parameter α when using the tILU_0 smoother is $\alpha = 0.5$. Using a publicly available parallel AMG solver we also tested the scalability of the new preconditioner. All these tests on tILU_0 are compared to the cases when MG is used with standard point smoothers (Jacobi and Gauss-Seidel) or ILU(0) smoother.

The results in the sequel are presented in terms of iteration counts of the preconditioned GMRES method, as well as the execution times. The execution time recorded in the thesis is separated into the assembly part of the preconditioner (which includes MG coarsening, assembly, nodal ordering in some cases and in the case of ILU(0) and tILU_0 methods the factorisation of the smoothing operators) and the total execution time, which also includes the Krylov solve time.

The preconditioning methodology is also tested on problems described by stretched or adaptively refined grids. These types of grids are frequently used to obtain accurate and computationally effective discretisations of convection-diffusion problems. The natural ordering of the nodes, used by default in the OOMPHLIB package, are devised from tree structured ordering. That is, ordering is based on mesh refinement. This type of ordering results in a more scattered non-zero pattern than that found in directional ordering, where the nodal structure results in a banded matrix pattern. We test the use of directional nodal ordering, as it is well known that smoothers may perform better given an appropriate nodal ordering in convection-diffusion cases. In LU factorisation the "fill-in" ratio is dependent on the non-zero structure of the coefficient matrix. We also investigate the effect of reordering the coefficient matrix, using Tarjan's "black-box" algorithm, so that potential "fill-in" during ILU(0) factorisation is limited.

For completeness, the proposed MG convection-diffusion preconditioner in the context of preconditioning the Navier-Stokes problem (see [31, Chapter 8]) is used. We use the MG preconditioner with the tILU_0 smoother in a "black-box" fashion, as a building block for the LSC Navier-Stokes preconditioner (see Section 4.6).

Chapter 4

Two-Dimensional Case Studies

This chapter will present the results of a comprehensive numerical evaluation of the novel smoothing technique for multigrid preconditioning performed on a number of discrete convection-diffusion problems in two spacial dimensions. We start with the simple diffusion problem, with the aim of obtaining benchmark convergence results which will serve as a basis for comparison. The following sections will introduce four test problems for the convection-diffusion equation, with increasing complexity of the convective field ranging from the constant uni-directional wind to cases with single and multiple recirculations.

We test the Krylov solver using both geometric and algebraic multigrid as a preconditioner. In the case of algebraic multigrid preconditioner we also evaluate parallel performance of the iterative solver. Our primary objective is to evaluate the effectiveness and the performance of the multigrid preconditioner with the novel tILU_0 smoother. The project will assume that the MG interpolation procedure is fixed (constructed using the finite element basis set or by a heuristical formula in the case of algebraic multigrid (AMG) [10, p.145, p.146] [50]). Therefore, the efficiency of the MG scheme remains solely dependent on the choice of the relaxation procedure. To put this evaluation into perspective, we compare the results with that obtained using the limiting cases of tILU_0 smoother (point Jacobi and ILU(0)) as well as the standard point symmetric Gauss-Seidel smoother. To simplify the notation ILU(0) will from now on be referred to as ILU_0. Also, the notation $Pe_* = \frac{1}{\epsilon}$ is used to represent the inverse diffusion parameter in the remaining contents of the thesis. For the case of geometric multigrid we give a small sample of results that test different strategies for ordering the unknowns.

The previous chapter introduced iterative methods for solving a sparse system of linear equations. In this chapter these linear equations are constructed via the SUPG FE discretisation of the convection-diffusion equation and standard Galerkin FE discretisation of the Navier-Stokes equations. The solving strategy that will

be used within this chapter is: GMRES as a Krylov solver, right-preconditioned by multigrid, using the relative residual (3.4) being less than 10^{-6} as the stopping criterion. This solution strategy is implemented as part of the OOMPHLIB software package [45]. For algebraic multigrid we use publicly available code BoomerAMG [50], which is part of the Hypre library [2] [33].

The computations are performed on "Horace", one of the parallel computers at The University of Manchester. "Horace" is a Bull Itanium 2 system with 192 processor cores. This is divided into 24 computational nodes each with 4 dual core Intel Itanium 2 Montecito 1.6GHz. Thus, each node has 8 cores, with 16GB RAM. The peak computational performance of each core is 6.4 GFlop/s. The interconnections between the computational nodes is a single rail Quadrics QsNetII [68].

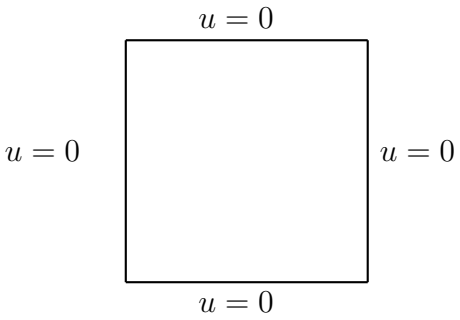
4.1 The diffusion problem

The diffusion equation, although mathematically simple, represents a model for a large number of physical processes and phenomena such as gravitation, electromagnetism and elasticity [31, p.10]. The diffusion problem represents a part of the convection-diffusion problem, and is obtained as a limiting case when $Pe = 0$. The Poisson equation for Case study 4.1 is defined as:

Find $u \in \mathcal{C}^2(\Omega)$ such that

$$-\nabla^2 u = f \text{ in } \Omega \subset \mathbb{R}^2, \tag{4.1}$$

on a square domain $\Omega = [-1, 1]^2$, subject to homogeneous Dirichlet BCs, $u = 0$ on $\partial\Omega$. This problem can be regarded as a model of temperature distribution in a homogeneous plate where boundaries are kept at a zero temperature and which is subject to a heat source ($f = 1$) [31, p.11].



The solution to this problem can be seen in Figure 4.1. From the figure we observe that the solution is smooth, without any sudden jumps (solution layers).

The problem is discretised using a uniform quadrilateral grid with bilinear (Q_1)

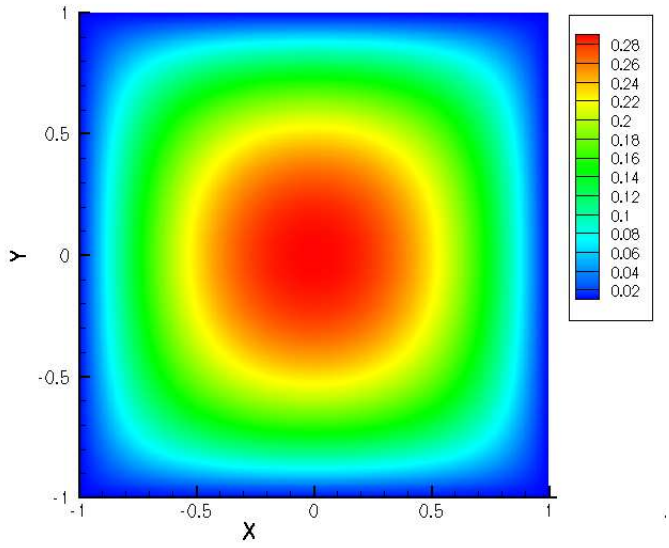


Figure 4.1: A contour plot of the solution u of the diffusion problem (Case study 4.1).

basis set. We apply Solver Strategy 4.1 to linear systems obtained from the discretisation of Case study 4.1.

Solver Strategy 4.1

Krylov Method: GMRES (tolerance 10^{-6})
Preconditioner: GMG/AMG
  Cycle: $V(2,2)$
Smoothers:
  $Gauss - Seidel$
  $ILU_0(0.5)$
  $tILU_0(0.5, \alpha)$
  $Jacobi(0.5)$

The coefficient matrix A, obtained from the discretisation of the Poisson problem by Galerkin FEM is symmetric positive definite. Thus, the resulting system can be solved by the (preconditioned) conjugate gradient method. However the symmetric positive definite properties of the discrete Poisson problem are lost when convection is introduced. To give a more direct comparison to the efficiency of solving convection-diffusion systems, we therefore solve the Poisson problem using GMRES. For each case we record the number of GMRES iterations, the preconditioner setup time and the total execution time. The preconditioner setup time includes the time needed to generate coarse grid operators, assemble the interpolation matrices and setup the smoothers (which may include nodal ordering) at all MG levels. In the case of incomplete factorisation smoothers this also includes the time for performing the factorisation as well. The total execution time consists of the preconditioner setup

time and the time needed to solve the discrete problem with GMRES.

In Table 4.1 we summarise the convergence characteristics of the right-preconditioned GMRES solver applied to the discrete Poisson problem. We examine both GMG preconditioner in Table 4.1(a) and AMG preconditioner in Table 4.1(b). For both

Table 4.1: The iteration counts and the (setup,total) execution time (in seconds) required for the convergence of the GMRES solver right-preconditioned by multigrid with several smoothers, when applied to the solution of the discrete Poisson problem (Case study 4.1) obtained from $Q1$ SUPG FEM on uniform grids. (a) GMG preconditioner; (b) AMG preconditioner; (c) parallel AMG preconditioner applied to the largest discrete problem ($N = 1,046,529$).

(a) GMG

Smoother \ N	3969	16125	65025	261121	1046529
$ILU_0(0.5)$	5(0.68,0.81)	5(3.04,3.69)	5(13.0,16.3)	5(53.4,68.0)	5(222.5,285.7)
$tILU_0(0.5, 0.25)$	5(0.68,0.74)	6(3.04,3.35)	6(13.0,14.6)	6(53.5,61.0)	6(223.7,259.3)
GaussSeidel	4(0.68,0.72)	4(3.06,3.26)	4(13.0,14.2)	4(53.4,59.0)	4(220.1,244.5)

(b) AMG

Smoother \ N	3969	16125	65025	261121	1046529
$ILU_0(0.5)$	8(0.05,0.06)	8(0.20,0.48)	8(0.97,5.13)	8(3.85,19.9)	8(15.1,79.6)
$tILU_0(0.5, 0.25)$	9(0.04,0.05)	9(0.16,0.32)	9(0.84,3.82)	10(3.37,16.7)	10(13.4,69.6)
$tILU_0(0.5, 0.5)$	9(0.04,0.05)	9(0.16,0.31)	9(0.81,3.63)	10(3.36,17.2)	10(14.0,75.7)
$Jacobi(0.5)$	9(0.02,0.05)	9(0.08,0.26)	9(0.44,3.99)	10(1.81,19.5)	10(7.23,73.5)
GaussSeidel	7(0.02,0.04)	7(0.09,0.18)	7(0.43,3.29)	8(1.84,14.0)	8(7.25,55.0)

(c) Parallel AMG

Smoother \ P	1	2	4	8	16
$ILU_0(0.5)$	8(13.0,28.90)	9(6.71,17.7)	9(3.43,9.20)	9(1.88,5.25)	10(1.05,2.76)
$tILU_0(0.5, 0.25)$	10(11.2,24.0)	11(6.33,14.4)	11(3.10,7.56)	11(1.88,4.80)	11(1.04,2.35)
$tILU_0(0.5, 0.5)$	10(11.5,24.1)	11(6.03,14.2)	11(3.09,7.27)	11(1.74,4.56)	11(0.96,2.26)
$Jacobi(0.5)$	10(5.64,21.1)	11(3.16,11.5)	11(1.61,6.22)	11(0.91,3.77)	11(0.43,1.74)
GaussSeidel	8(5.10,14.7)	8(3.20,11.4)	8(1.61,8.43)	8(0.90,7.20)	8(0.44,6.38)

cases we examine the efficiency of MG preconditioning using a range of smoothers. We report the convergence results for two different levels of the truncation parameter ($\alpha = 0.25$ and $\alpha = 0.5$) and the damping parameter $\gamma = 0.5$. As the discrete Poisson operator in 2D has a constant matrix stencil for each mesh size h, the truncation procedure (3.43) will produce standard ILU_0 smoothing for all $\alpha = [0, 0.125)$[1] and produce Jacobi smoothing for all $\alpha = [0.125, 1]$ (this applies to the GMG case only as the coarse level operators in AMG case may differ). In Table 4.1(a) we therefore only represent $\text{tILU}_0(0.5, 0.25)$ as a smoother.

For comparison, we report the preconditioner effectiveness when using two limiting cases of tILU smoother: ILU_0 (obtained for $\alpha = 0$) and damped Jacobi (obtained for $\alpha = 1$) and using the standard Gauss-Seidel smoother.

The results in Table 4.1 suggest that ILU_0 smoother is h-robust, in this context, with a small number of iterations. Truncated versions of the ILU_0 smoother show a

[1]The cutoff value of $\alpha = 0.125$ applies to Q_1 discretisation (see Section 2.4) and will be different for other discretisations.

performance equal to that of the damped Jacobi smoother. It should be emphasised, however, that in this case the smallest number of iterations and the shortest execution time (for both GMG and AMG cases) was observed when Gauss-Seidel smoother is used. MG is known to be an optimal preconditioner with respect to the discrete problem size for the Poisson problem, and this property is clearly demonstrated in Table 4.1 for all smoothers.

From part(a) and (b) of Table 4.1 it can be seen that the setup phase for GMG takes considerably longer than that of AMG when applied to the same discrete problem. This can be explained by the way GMG is implemented in OOMPHLIB. GMG is constructed by uniformly or adaptively un-refining the initial fine grid. The coefficient matrix is then constructed using a Q1 FEM discretisation associated with each mesh in the hierarchy. This operation corresponds to a substantial proportion of the overall setup time (around 80%). A small addition to the overall setup time is the construction of the hierarchy of interpolation matrices (Appendix B.6), this accounts for approximately 9% of the setup time. Within this interpolation setup time, half of it is taken up by setting up the connections between the coarse and fine elements in the refinement. Within the total setup time there is also the small cost of setting up the smoother. This time will vary depending on the choice of smoother. Since the setup time for coarse-level operators and interpolation matrices is equal in all cases, the difference between the total setup times within each sub-table represents the differences in the time needed to assemble the smoothers. In the case of the Jacobi and Gauss-Seidel method there is no additional overhead associated with assembling the smoothers, hence the preconditioner setup times should be equal in these two cases. In the case of ILU_0 method during the setup phase a copy of the original matrix at every MG level needs to be made, before performing the incomplete factorisation of the copied matrix. For tILU_0, we need to make copies of the original matrices at all MG levels, followed by the static analysis of the non-zero pattern to truncate small off-diagonal entries, before performing ILU_0 factorisation on truncated matrices.

In part(c) Table 4.1 we report the performance of AMG preconditioned GMRES solver when the preconditioner is parallelised on P processors. The results are reported only for the largest problem size ($N = 1046529$). From these results it can be concluded that parallelisation of the classical AMG preconditioner can have further detrimental effect to the numerical efficiency in some cases.

For the Gauss-Seidel smoother parallelisation is achieved by generalised red-black ordering of unknowns. This can have a negative effect on the numerical efficiency of the method, compared to the standard Gauss-Seidel method. Jacobi is naturally parallel, therefore it does not lead to any deterioration in the smoother's efficiency. In the case of incomplete factorisation (both ILU_0 and tILU_0), parallelisation of smoothers

is done in a block Jacobi fashion. That is, by applying the standard ILU_0 method to the related diagonal blocks. The block Jacobi method assumes truncation of the non-zero entries in the off-diagonal blocks. This truncation inevitably affects numerical efficiency of the algorithm as the number of processors increases. The requirement to truncate off-diagonal non-zero entries to facilitate parallelisation of ILU_0 factorisation resembles the idea used in $tILU_0$. The difference between truncation performed in $tILU_0$ method and truncation performed in parallel ILU_0 method is that the former method is done with regard to the numerical efficiency of the resulting algorithm (based on the magnitude of the non-zero entries in each row), while the latter is centred around achieving good load balancing. This implies that when $tILU_0$ smoothing is applied in parallel, potentially there could be further truncation in the coefficient matrices (due to blocking of these matrices). This would lead to a further deterioration in the numerical effectiveness of the resulting smoother. This effect is, to a small extent, presented in Table 4.1(c) (there is an increase by 1 iteration moving from 1 to 2 processors), but will become significantly more pronounced in cases of complicated convection flow. A further consideration that may cause deterioration in the iteration counts is that no communication between the processors is attributed to the classical AMG parallel coarsening procedure.

Having in mind that preconditioning is often computationally the most expensive part of the solution algorithm (accounting for 80-90% of the overall solution time) it is realistic to expect a decent parallel performance. Although this is the case when Jacobi and ILU_0 smoothers are used, there is a considerable loss of parallel efficiency when Gauss-Seidel smoother is used (for example although the sequential execution time of the AMG preconditioned GMRES with Gauss-Seidel smoother is by far the shortest (around 40% shorter than its nearest competitor AMG with $tILU_0(0.5, 0.25)$ smoother), when using 16 processors, the same method is 3.6 times slower than the best method (GMRES/AMG/damped Jacobi in this case). Parallel efficiency (3.42) of GMRES/AMG/Gauss-Seidel on 16 processors is only 0.14, while parallel efficiency of GMRES/AMG damped Jacobi is 0.76. For comparison, parallel efficiency for this case of GMRES/AMG/ILU_0 is 0.65 and that of GMRES/AMG/$tILU_0(0.5, 0.25)$ is 0.64. Thus, in parallel, $tILU_0$ smoother leads to a MG preconditioner of similar parallel efficiency as Jacobi smoothing.

4.2 Geometric multigrid preconditioning of the convection-diffusion problem

In this section we present the convergence results, for solving the system of linear equations obtained from the SUPG FEM discretisation of the following convection-diffusion problems. The linear solvers used in this section are the GMRES method preconditioned with GMG which uses several different smoothers (Solver Strategy 4.2). The efficiency of the preconditioners are examined on a range of different problems and different discretisations.

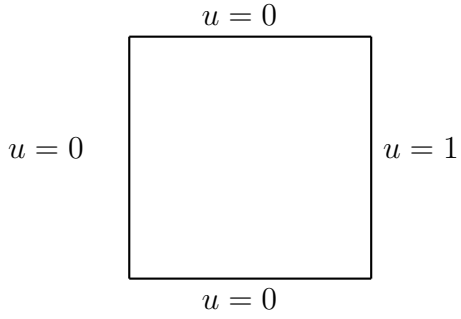
Solver Strategy 4.2

Krylov Method: Right preconditioned GMRES (tolerance 10^{-6})
Preconditioner: GMG
  Cycle: $V(2,2)$
Smoothers:
  $Gauss - Seidel$
  $ILU_0(0.5)$
  $tILU_0(0.5, \alpha)$
  $Jacobi(0.5)$

4.2.1 Constant uni-directional wind

This case is a model for temperature distribution on a square domain $\Omega = [-1, 1]^2$, where a uni-directional convection field flows from the heated boundary. In addition, at all other boundaries of the square domain the temperature is kept at zero. This is formally represented with the following Dirichlet BCs:

$$\begin{aligned} &u(x = 1, -1 \leq y \leq 1) = 1 && \text{(hot wall)} \\ &u = 0\,, \text{ elsewhere on } \partial\Omega && \text{(cold wall).} \end{aligned} \tag{4.2}$$



The convection field flows in the negative x-direction and has a constant ($\vec{w}$ =

$(-1, 0)$), with a source term equal to zero ($f = 0$). This is represented by a vector plot in Figure 4.2(a). Figure 4.2 also represents the solution u as a contour plot

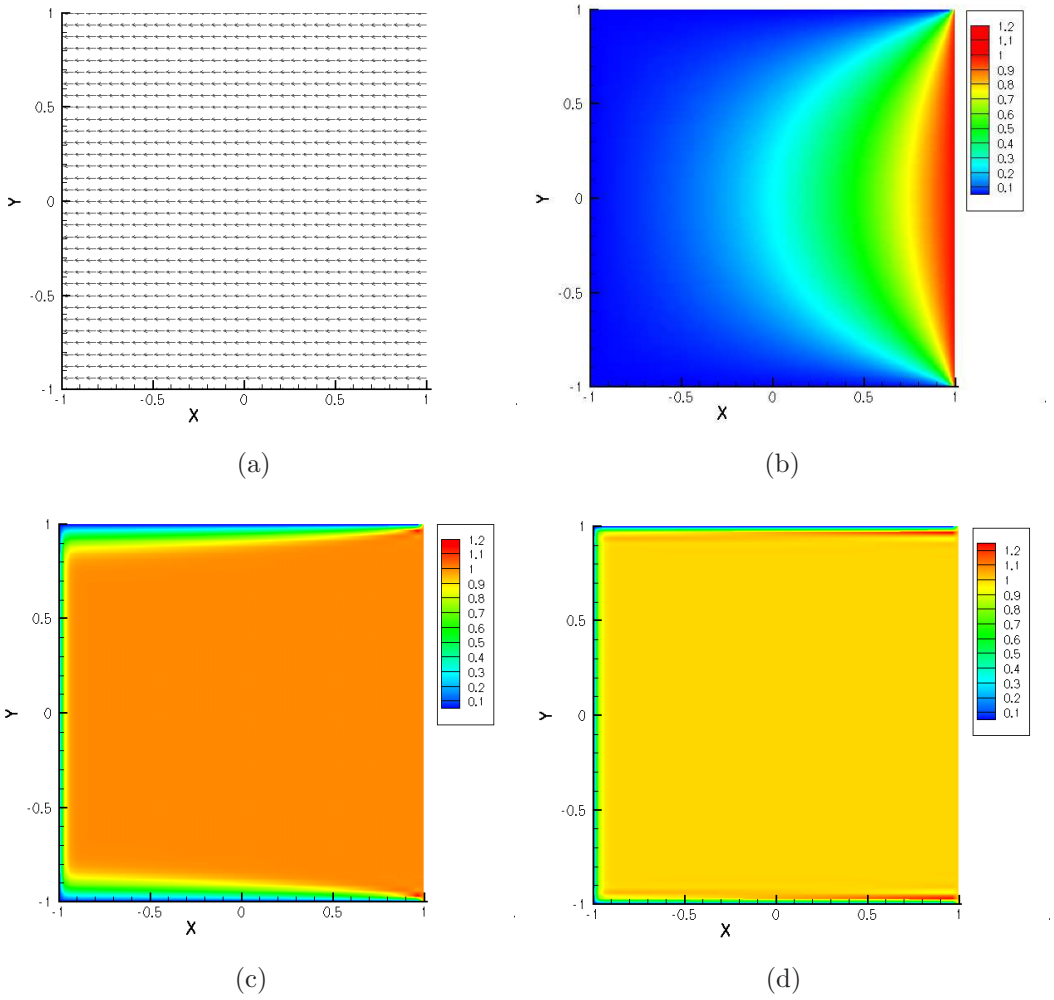


Figure 4.2: The convection-diffusion problem (Case study 4.2.1): constant, unidirectional wind $\vec{w} = (-1, 0)$, no source term ($f = 0$); (a) Arrow plot of the wind. (b) the solution for $Pe = 0$, (c) the solution for $Pe_* = 500$, (d) the solution for $Pe_* = 10,000$.

for three different values if $Pe_* = 0$, 500 and $10,000$. In Figure 4.2(b) we see a diffusion model ($Pe_* = 0$) where heat from the hot wall diffuses to the more colder areas of the domain leading to a smooth transition of reducing heat throughout the domain (red representing hot and blue representing cold parts of the domain). In Figure 4.2(c) we see the solution u as a contour plot when $Pe_* = 500$. The model now represents a convection-diffusion model. With these parameters the transfer of heat is much more intense throughout, with a large proportion of the domain being heated, represented in orange. In addition it is also important to note that the

solution shows steep boundary layers close to the cold boundaries, due to the homogeneous Dirichlet BC at $u(x = -1, -1 \leq y \leq 1)$, $u(-1 \leq x \leq 1, y = -1)$ and $u(-1 \leq x \leq 1, y = 1)$ intercepting the horizontal convection flow.

Finally Figure 4.2(d) represents the solution for highly convective flow, $Pe_* = 10,000$. An increase in the relative convection strength has lead to a larger proportion of the domain being heated, and to increasingly steeper boundary layers near the cold boundaries.

This model, along with Case study 4.1, due to their simplicity, will serve as a benchmark in terms of iteration count and solve times. Also, Case study 4.2.1 is the only scenario where performing some analysis of the smoothing operator is possible, as a constant matrix stencil is required for this purpose.

Uniform grid refinement

We start with the discretisation of Case study 4.2.1 on a sequence of uniformly refined grids. The default grid structure in OOMPHLIB is constructed through a tree based data structure which naturally induces a tree structured enumeration of the unknowns. Global nodal numbering of a sequence of uniformly refined grids, enumerated by a tree-based method are presented in Figure 4.3.

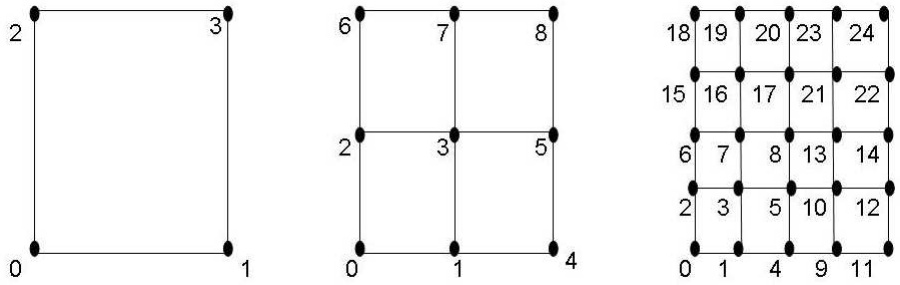


Figure 4.3: An example of OOMPHLIB [45] global tree structure nodal ordering for uniformly refined grids.

Table 4.2 summarises the convergence results of Solver Strategy 4.2. The table show that using $\text{ILU}_0(0.5)$ as a smoother results consistently in the smallest iteration count regardless of the diffusion parameter or grid size. Also, GMRES/GMG/$\text{ILU}_0(0.5)$ can be considered Pe,h-robust and an optimal solver in this case. However, the computational cost of assembling and applying the $\text{ILU}_0(0.5)$ smoother makes other less computationally expensive smoothers an attractive choice.

The use of the Gauss-Seidel smoother in this case also proved Pe,h-robust, following asymptotically the same convergence pattern as the $\text{ILU}_0(0.5)$ smoother. However this is not the case for coarser grids. Here we find Gauss-Seidel to be dependent on

Table 4.2: The iteration counts and the (setup,total) execution time (in seconds) required for the convergence of the GMRES solver right-preconditioned by geometric multigrid with several smoothers, when applied to the solution of the discrete convection-diffusion problem (Case study 4.2.1) obtained from $Q1$ SUPG FEM on uniform grids. Natural (tree-based) ordering of the unknowns, with each sub-table representing a different diffusion parameter.

(a) Pe_*=500

Smoother \ N	3969	16129	65025	261121	1046529
$ILU_0(0.5)$	5(0.89,1.04)	5(4.09,4.87)	5(17.6,22.1)	6(72.9,97.1)	6(247.0,318.3)
$tILU_0(0.5, 0.25)$	7(0.89,0.98)	7(4.11,4.56)	8(17.7,20.7)	9(73.3,90.7)	10(247.9,303.7)
$tILU_0(0.5, 0.5)$	7(0.89,0.98)	7(4.09,4.55)	8(17.7,20.7)	10(73.2,91.0)	11(247.7,307.7)
$Jacobi(0.5)$	28(0.88,1.22)	24(4.10,5.71)	22(17.7,26.8)	22(73.0,118.)	21(245.8,372.6)
$GaussSeidel$	6(0.88,0.95)	5(4.09,4.38)	5(17.6,19.6)	5(73.0,83.3)	6(254.9,293.2)

(b) Pe_*=1,000

Smoother \ N	3969	16129	65025	261121	1046529
$ILU_0(0.5)$	5(0.88,1.03)	5(4.08,4.86)	5(17.6,22.1)	5(73.1,93.8)	6(253.7,339.2)
$tILU_0(0.5, 0.25)$	9(0.88,1.00)	8(4.10,4.61)	8(17.7,20.7)	9(73.4,90.7)	10(288.0,373.5)
$tILU_0(0.5, 0.5)$	9(0.89,1.00)	8(4.10,4.61)	8(17.7,20.6)	9(73.8,90.6)	10(292.0,371.3)
$Jacobi(0.5)$	36(0.89,1.34)	31(4.10,6.07)	29(17.7,29.8)	28(73.8,132.)	28(299.4,557.8)
$GaussSeidel$	8(0.89,0.97)	6(4.08,4.40)	6(17.6,20.0)	6(73.1,85.0)	6(251.3,286.7)

(c) Pe_*=2,000

Smoother \ N	3969	16129	65025	261121	1046529
$ILU_0(0.5)$	5(0.89,1.04)	5(4.09,4.88)	5(17.7,22.1)	5(73.4,94.0)	6(296.9,396.7)
$tILU_0(0.5, 0.25)$	10(0.89,1.02)	9(4.08,4.67)	9(17.7,21.0)	9(73.6,90.6)	9(304.9,381.3)
$tILU_0(0.5, 0.5)$	10(0.89,1.01)	9(4.11,4.71)	9(17.7,21.0)	9(73.7,90.8)	9(289.6,358.7)
$Jacobi(0.5)$	45(0.88,1.45)	48(4.10,7.60)	41(17.6,35.4)	38(73.5,156.)	36(273.2,558.6)
$GaussSeidel$	11(0.88,0.99)	9(4.08,4.55)	7(17.6,20.3)	6(73.2,85.1)	6(303.7,354.2)

the diffusion parameter. This can possibly be explained by the inaccurate approximation of the solution for such coarse discretisations. The total solve time of an iterative solver, in the case of convection-dominated problems, that uses a Gauss-Seidel smoother is roughly 10% smaller than a solver that uses a damped ILU_0 smoother. This is because of the larger computational overhead associated with the assembly and application of the ILU_0 smoother.

The Jacobi smoother continuously shows poor performances compared to the other methods. Although performing h-robust, the Jacobi smoother leads to a GMG preconditioner which exhibits a considerable dependence on Pe_*. Based on the results from Table 4.2 the asymptotic behaviour of Jacobi appears to be $O(\sqrt{Pe_*})$. This considerable dependence on Pe_* and a relatively high iteration count make Jacobi smoothing not competitive in this context, despite having the lowest computational cost.

By contrast, the tILU_0 smoother demonstrates Pe,h-robust behaviour for this problem. There is an increase in iteration count by 50% compared to the cases when ILU_0 and Gauss-Seidel are used. However, by allowing a truncation of the original matrix before factorising we are reducing the storage cost of matrix M (3.8), when

compared to ILU_0. This can help boost the caching properties of the resulting algorithm as it has to deal with much smaller datasets. Due to the significant amount of truncated entries (in Table 4.10 with $\alpha = 0.25$ and $\alpha = 0.5$ we observe the truncation of 78% of the off-diagonal entries across all MG levels), there is a reasonable gain in the execution time in the case $Pe_* = 2000$ compared to $\text{ILU}_0(0.5)$ smoother and a comparable performance to that when Gauss-Seidel smoother is used. As truncation plays an important role in reducing storage costs and as a consequence the total solve time, it is reasonable to believe that further truncation will result in a further reduction in total solve time. This is computationally true per iteration as the limiting case $\text{tILU}_0(0.5,1)$ is equivalent to the Jacobi(0.5) smoother. However, there is a value of α that will give a minimum execution time for each problem, if this value is exceeded the execution time will grow again (this can be seen when using Jacobi as a smoother). The results in Table 4.2 indicate that tILU_0 smoother can offer a suitable alternative to computationally expensive, memory intensive, but robust ILU_0 smoother and inexpensive but potentially ineffective Jacobi smoother in the context of MG preconditioning of strongly convection-dominated problems.

To summarise, in Table 4.2 we find that the Gauss-Seidel method is the preferred chosen smoother for Solver Strategy 4.2 when applied to Case study 4.2.1, closely followed by $\text{tILU}_0(0.5,0.5)$.

Lexicographical nodal ordering in a negative x-direction

The Gauss-Seidel algorithm approximates the matrix by its lower triangular part. This means that its efficiency will be compromised if there are large (in magnitude) off-diagonal entries in the upper triangular part of the matrix. The aim of downwind ordering of the unknowns is to bring large off-diagonal entries to the lower triangular part of the matrix and thus improve the effectiveness of the Gauss-Seidel smoother [44]. By ordering the unknowns in the direction of the wind this leads to excellent results, as ideally A becomes a block lower triangular matrix.

The ILU_0 algorithm (3.18) can also benefit from ordering the entries. Different orderings of the unknowns may lead to completely different ILU_0 factorisations, which may have a profound effect on its effectiveness as a smoother. Depending on the amount and magnitude of the "fill-in" entries the resulting smoothers can have significantly different properties. As the components of a Jacobi algorithm are independently updated, ordering the nodes in the mesh will have no effect.

For convection-dominated problems (with Gauss-Seidel as a smoother) a good convergence rate requires ordering of the unknowns [44]. In [85, p.236, p.237] the authors study the effect of nodal ordering on various point and line smoothers. The findings show that ordering in the direction of the wind leads to a more efficient

Gauss-Seidel smoother.

In the cases of constant uni-directional wind it is possible to analyse the smoothing efficiency of a particular iterative method using Fourier analysis. In addition, the problem needs to be discretised on a uniform regular tensor product grid. For Case study 4.2.1 we order the nodes in a lexicographical negative x-direction.

Fourier smoothing analysis

Wesseling [93, Chapter 7] discusses the analytical procedure, based on Fourier analysis, to determine the contraction factors that correspond to different components of the solution error in the direction of the Fourier vectors.

Denote

$$E_{amp}\phi(\theta) = \lambda(\theta)\phi(\theta),$$

the eigenvalue problem associated with the amplification matrix E_{amp} (3.14), where $\lambda(\theta)$ is the eigenvalue of the matrix E_{amp} and $\phi(\theta)$ is the associated eigenvector represented by,

$$\phi_{\mathbf{j_1},\mathbf{j_2}}(\theta) = e^{\mathbf{i}(\mathbf{j_1}\theta_1+\mathbf{j_2}\theta_2)}. \tag{4.3}$$

In (4.3) $\mathbf{j_1}, \mathbf{j_2}$ are the nodal coordinates in Cartesian space and $\mathbf{i} = \sqrt{-1}$. Define $\Theta(\theta)$ as the Fourier space with θ_1, θ_2 coordinates of the points in discrete Fourier grid $\Theta(\theta) = [-\pi, \pi]^2$.

The eigenvalues of E_{amp} (3.14) are calculated by [93, p.113] [81, p.14]:

$$\lambda(\theta) = \frac{\sum_j N(j)e^{\mathbf{i}(\mathbf{j_1}\theta_1+\mathbf{j_2}\theta_2)}}{\sum_j M(j)e^{\mathbf{i}(\mathbf{j_1}\theta_1+\mathbf{j_2}\theta_2)}} \qquad \theta \in \Theta(\theta), \tag{4.4}$$

where $j = (\mathbf{j_1}, \mathbf{j_2})$ and $\theta = (\theta_1, \theta_2)$. The eigenfunctions $\phi(\theta)$ belong to two main spaces [93, p.108] [81, p.128]:

$$\begin{aligned} &\theta \in \Theta_S \quad \text{Smooth eigenfunctions} : \Theta_S(\theta) = \Theta(\theta) \cap (-\tfrac{\pi}{2}, \tfrac{\pi}{2})^2 \\ &\theta \in \Theta_R \quad \text{Oscillatory eigenfunctions} : \Theta_R(\theta) = \Theta(\theta) \backslash \Theta_S(\theta). \end{aligned}$$

The FE discretisation of Case study 4.2.1 using SUPG method with Q_1 approximation defined on a uniform tensor product grid results in a discrete operator that can be

represented in a nine point stencil notation:

$$\begin{array}{c} A \\ \left(\begin{array}{ccccccc} d & q & & & f & g & p \\ c & d & q & & & f & g & p \\ & c & d & q & & & f & g \\ & & c & d & q & & & f \\ b & & & c & d & q & & \\ a & b & & & c & d & q & \\ z & a & b & & & c & d & q \\ & z & a & b & & & c & d \end{array} \right) \end{array} \sim \begin{array}{c} Stencil \\ \left[\begin{array}{ccc} f & g & p \\ c & d & q \\ z & a & b \end{array} \right] \end{array}.$$

To simplify the notation we denote:

$$E = fe^{\mathbf{i}(-\theta_1+\theta_2)} + ge^{\mathbf{i}\theta_2} + pe^{\mathbf{i}(\theta_1+\theta_2)} + qe^{\mathbf{i}(\theta_1)},$$

$$F = ce^{-\mathbf{i}(\theta_1)} + ze^{-\mathbf{i}(\theta_1+\theta_2)} + ae^{-\mathbf{i}(\theta_2)} + be^{\mathbf{i}(\theta_1-\theta_2)}.$$

Using right to left, bottom to top lexicographical ordering of the nodal points, by substituting (4.4) into relevant stencil notation we can calculate the eigenvalues (amplification factor) for each smoother as follows:

Gauss-Seidel

$$\lambda(\theta) = \frac{-E}{d+F}. \tag{4.5}$$

Jacobi

$$\lambda(\theta) = \frac{-(E+F)}{d}. \tag{4.6}$$

ILU$_0$

For the case of ILU$_0$ a new stencil notation must be introduced to take into consideration the matrix residual. This stencil notation takes the following form:

$$\underbrace{LD^{-1}U}_{M} - A = N = \begin{bmatrix} \xi & 0 & 0 & 0 & 0 \\ \omega & 0 & 0 & 0 & \varsigma \\ 0 & 0 & 0 & 0 & \vartheta \end{bmatrix},$$

$$N = \frac{\xi e^{\mathbf{i}(-\mathbf{2}\theta_1+\theta_2)} + \omega e^{-\mathbf{2i}(\theta_1)} + \varsigma e^{\mathbf{2i}(\theta_1)} + \vartheta e^{\mathbf{i}(\mathbf{2}\theta_1-\theta_2)}}{\widetilde{d}},$$

where $\widetilde{d}$ is computed from a recursion algorithm [93, p.142]. Then the eigenvalues of E_{amp} are given by

$$\lambda(\theta) = \frac{N}{N + (E + F + d)}. \tag{4.7}$$

tILU$_0$

To find the eigenvalues of our new tILU$_0$ method, we initially treat our truncated method in a similar fashion to that of ILU$_0$. However given that a truncation has been performed statically before the application of the incomplete factorisation, our modified stencils in M have only taken into account the truncated part of the matrix. This means that we need to place all truncated entries into the matrix N_t and derive the modified formulas for the coefficients of N_t:

$$\underbrace{(LD^{-1}U)_t}_{M_t} - A_t = N_t = \begin{bmatrix} \xi & 0 & 0 & 0 & 0 \\ \omega & 0 & 0 & 0 & \varsigma \\ 0 & 0 & 0 & 0 & \vartheta \end{bmatrix}.$$

The statically removed stencil entries from matrix A during the truncation process are represented as N_t^*, and will be represented as

$$N_t^* = (A - A_t) = \begin{bmatrix} f_t^* & g_t^* & p_t^* \\ c_t^* & 0 & q_t^* \\ z_t^* & a_t^* & b_t^* \end{bmatrix}.$$

The residual error of matrix A is therefore

$$N = N_t + N_t^* = N_t - (E_t^* + F_t^*).$$

The new equation to calculating the eigenvalues is therefore

$$\lambda(\theta) = \frac{N}{N_t + (E_t + F_t + d)}. \tag{4.8}$$

In the case of convection-diffusion it is not possible to evaluate the eigenvalues of the matrix E_{amp} analytically. This is why we resort to a numerical method. Given the discretisation of the Fourier space by a tensor product grid of points with uniform spacing h_θ we can numerically calculate the absolute value of the complex eigenvalues $|\lambda(\theta)|$ from (4.5), (4.6), (4.7) and (4.8).

In Table 4.3 we summarise the values of $\sigma = \sup|\lambda(\theta)|$ at different Pe_* for two different Fourier space mesh sizes h_θ. A smoother will converge if $0 < \sigma < 1$ [31,

		Gauss-Seidel	ILU_0	$tILU_0$ (0.25)	Jacobi
Pe_*=0	$h_\theta=\frac{2\pi}{64}$	0.4087	0.1377	0.5	0.5
Pe_*=0	$h_\theta=\frac{2\pi}{128}$	0.4203	0.1466	0.5	0.5
Pe_*=500	$h_\theta=\frac{2\pi}{64}$	0.5314	0.3822	0.5613	1.751
Pe_*=500	$h_\theta=\frac{2\pi}{128}$	0.5459	0.3822	0.5811	1.751
Pe_*=2000	$h_\theta=\frac{2\pi}{64}$	0.8294	0.9739	0.8726	1.9336
Pe_*=2000	$h_\theta=\frac{2\pi}{128}$	0.8455	0.9739	0.8811	1.9336

Table 4.3: The modulus of the maximum eigenvalue $\sigma = \sup|\lambda(\theta)|$ for different smoothers for Case study 4.2.1 with negative x-directional ordering as a function of Pe_*. A matrix size of $N = 3,969$ is used.

p.98]. For a diffusion problem, a value of $\sup|\lambda(\theta)|$ for the Jacobi smoother is found analytically to be 0.5 [31, p.100]. This is consistent with that reported in Table 4.3, for $Pe_* = 0$. For ILU_0, when $Pe_* = 0$ Wesseling [93, p.143] finds numerically that $\sigma = 0.13$ (this cannot be achieved analytically; this is also the case when $Pe_* > 1$). This is consistent with Table 4.3. The smoothing factor for a Gauss-Seidel seven point stencil when $Pe_* = 0$ [93, p.124] is $\sigma = |\lambda(\frac{\pi}{2}, cos^{-1}(\frac{4}{5})| = 0.5$. We would expect a nine point stencil to have a smaller value σ, which we found to be $\sigma \approx 0.42$.

In a case study similar to this Syamsudhuha [81, p.45] shows that for a seven-point stencil using Gauss-Seidel with lexicographical ordering of the unknowns, σ initially ($\sigma = 0.5$ at $Pe_* = 0$) reduced as Pe_* increases, up until $Pe_* = 10^{2.5}$. However, with a further increase in Pe_*, the value of σ increases considerably with $\sigma \approx 0.49$ for $Pe_* = 10^3$ and $\sigma \approx 0.85$ for $Pe_* = 10^4$ indicating a considerable deterioration in the efficiency of the smoother. This efficiency pattern coincides with our findings in Table 4.3.

For the discrete convection-diffusion problem arising in Case study 4.2.1 the truncation of the coefficient matrix $N = 3969$ is exactly the same for both values of the truncation parameters ($\alpha = 0.25$ and $\alpha = 0.5$), therefore we will only report the results for $\alpha = 0.25$ in Table 4.3.

From Table 4.3 we observe that for a fixed problem size ($N = 3969$) all smoothers exhibit strong dependence on Pe_*. A further observation from Table 4.3 is that it is necessary to introduce damping into the Jacobi method, to obtain a convergent smoother, as convergence is only guaranteed if $\sigma < 1$.

The results in Table 4.3 suggest that for $Pe_* = 2000$ the $tILU_0$ smoother has lower σ than the ILU_0 smoother, and has a comparable error reduction to the Gauss-Seidel smoother. The results from Table 4.3 use the Fourier space defined in [93, p.108].

However Wesseling [93, p.111] states that when using a Fourier series for smoothing analysis with Dirichlet BC, better agreement with practical results are found when $\theta = 0$ is not included in the Fourier space [93, p.112]. This has a profound effect only on the ILU_0 smoother when $Pe_* = 2000$ where we obtain for $h_\theta = \frac{2\pi}{64}, \sigma = 0.4102$ and for $h_\theta = \frac{2\pi}{128}, \sigma = 0.6634$.

In Figure 4.4 we plot the eigenvalues $\lambda(\theta)$ in the Fourier space $[-\pi, \pi]^2$ computed for four different smoothers (Jacobi, Gauss-Seidel, ILU_0, $\text{tILU}_0(\alpha = 0.25)$). These results offer a more complete picture of the smoothing properties of each method than Table 4.3, which represents the worst case scenario. From these plots we can see that in the case of the ILU_0 smoother (part(a)) most of the oscillatory error modes

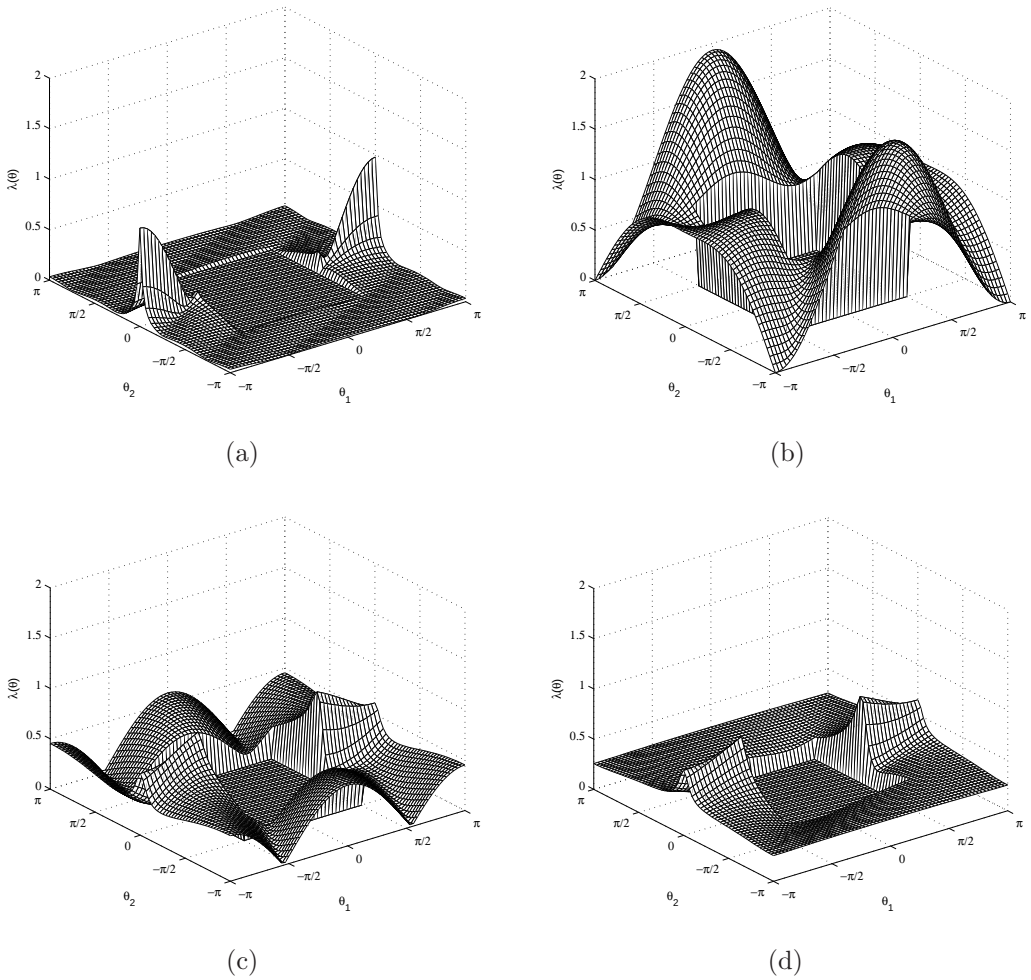


Figure 4.4: Plots of the eigenvalues $\lambda(\theta)$ of the matrix E_{amp} for Case study 4.2.1 with $Pe_* = 2,000$ discretised by SUPG FEM with Q_1 approximation (discrete problem size $N = 3969$). The Fourier space $[-\pi, \pi]^2$ is discretised by a uniformly spaced grid of points with $h_\theta = \frac{2\pi}{64}$. (a) ILU_0 smoother. (b) Jacobi smoother. (c) $\text{tILU}_0(0.25)$ smoother. (d) Gauss-Seidel smoother.

are strongly reduced, except the modes around $|\theta_1| = \pi$, $\theta_2 = 0$.

Figure 4.4(b) suggests that in the case of Jacobi smoother a substantial portion of oscillatory modes have their corresponding eigenvalues well above one, inducting that this would be an inefficient choice of smoother (unless damped) for Case study 4.2.1.

Figure 4.4(c) represents the plot of $\lambda(\theta)$ for the tILU$_0(\alpha = 0.25)$ smoother. The interesting detail here is to compare this plot with those in part(a) and (b). This reveals the fact that tILU$_0$ smoother exhibits in part the features of ILU$_0$ smoother (spikes close to $|\theta_1| = \pi$, $\theta_2 = 0$) and in part the features of the Jacobi smoother (the curved part for $-\frac{\pi}{2} < \theta_1 < \frac{\pi}{2}$). This practically demonstrates the hybrid nature of the tILU$_0$ smoother.

It is interesting to note that ILU$_0$ and Gauss-Seidel have similar smoothing properties (judging from the shapes of the curves $\lambda(\theta)$ for part(a) and part(d) of Figure 4.4), with the observation that ILU$_0$ shows to be better in reducing a wider spectrum of oscillatory modes. In conclusion of Table 4.3 and Figure 4.4, we observe that ILU$_0$ would be expected to have the fastest convergence regardless of the diffusion parameter.

Having analysed smoothing properties of various methods, we examine the convergence of Solver Strategy 4.2 using lexicographical ordering of the unknowns in the negative x-direction. These results are summarised in Table 4.4.

In Table 4.4 the performance of our smoothers directly complies with our Fourier analysis findings in Table 4.3, when Dirichlet BC are imposed on the Fourier space (for $N = 3969$).

The results in Table 4.4 are broadly the same as those reported in Tables 4.2. From this we can conclude that for Case Study 4.2.1 ordering the unknowns seems to have virtually no effect to the performance of the smoothers and therefore the GMG preconditioner. This is a positive result as we are looking to create a preconditioner that is robust and which does not require any special ordering of the unknowns. As this is especially important if we are to further develop the smoother for AMG instead of GMG.

Table 4.4: The iteration counts and the (setup,total) execution time (in seconds) required for the convergence of the GMRES solver right-preconditioned by geometric multigrid with several smoothers, when applied to the solution of the discrete convection-diffusion problem (Case study 4.2.1) obtained from $Q1$ SUPG FEM on uniform grids. The lexicographical ordering of the unknowns in the negative x-direction, with each sub-table representing a different diffusion parameter.

(a) Pe_*=500

Smoother \ N	3969	16129	65025	261121	1046529
$ILU_0(0.5)$	5(0.89,1.14)	5(3.91,5.16)	6(17.3,24.9)	6(71.1,102.7)	6(294.6,422.8)
$tILU_0(0.5, 0.25)$	7(0.87,0.99)	7(3.92,4.48)	8(17.3,20.8)	9(71.9,90.4)	10(295.9,370.0)
$tILU_0(0.5, 0.5)$	7(0.87,0.99)	7(3.90,4.46)	8(17.3,20.8)	10(71.2,88.3)	11(295.3,373.9)
$Jacobi(0.5)$	28(0.88,1.22)	24(4.10,5.71)	22(17.7,26.8)	22(73.0,118.1)	21(245.8,372.6)
$GaussSeidel$	6(0.90,0.96)	5(3.92,4.18)	5(17.2,19.0)	6(71.3,79.0)	6(293.2,327.5)

(b) Pe_*=1,000

Smoother \ N	3969	16129	65025	261121	1046529
$ILU_0(0.5)$	5(0.86,1.12)	5(3.92,5.15)	5(17.3,23.8)	6(71.4,103.2)	6(304.1,434.4)
$tILU_0(0.5, 0.25)$	9(0.87,1.02)	8(3.90,4.52)	8(17.4,20.8)	9(71.5,90.0)	10(305.6,397.7)
$tILU_0(0.5, 0.5)$	9(0.88,1.02)	8(3.92,4.55)	8(15.9,18.8)	9(72.5,92.1)	10(300.7,379.7)
$Jacobi(0.5)$	36(0.89,1.34)	31(4.10,6.07)	29(17.7,29.8)	28(73.8,131.7)	28(299.4,557.8)
$GaussSeidel$	7(0.87,0.93)	6(3.90,4.19)	6(17.2,19.0)	6(71.8,79.7)	6(297.6,332.4)

(c) Pe_*=2,000

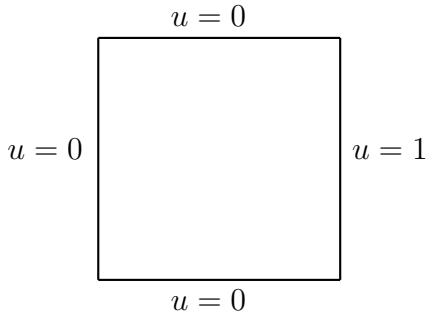
Smoother \ N	3969	16129	65025	261121	1046529
$ILU_0(0.5)$	5(0.87,1.12)	5(3.90,5.15)	5(16.0,21.7)	5(71.7,99.8))	6(306.8,441.9)
$tILU_0(0.5, 0.25)$	10(0.88,1.03)	9(3.91,4.59)	9(17.1,21.0)	9(73.6,93.2)	9(309.0,394.1)
$tILU_0(0.5, 0.5)$	10(0.87,1.03)	9(3.93,4.63)	9(17.3,21.2)	9(73.7,93.0)	9(305.0,384.8)
$Jacobi(0.5)$	45(0.88,1.45)	48(4.10,7.60)	41(17.6,35.4)	38(73.5,156.1)	36(273.2,558.6)
$GaussSeidel$	9(0.87,0.95)	8(3.92,4.31)	7(17.0,19.1)	7(74.4,84.0)	7(314.0,357.4)

4.2.2 Double glazing problem - recirculating wind

In this case we study the convection-diffusion problem known as the double glazing problem [31, p.119]. It is a well-known benchmark problem for testing the efficiency of linear solvers aimed at discrete convection problems. The main difficulty in this problem arises from the nature of the wind, which is cyclical (recirculating). At the same time, this is a physically relevant example of a wide class of enclosed flow problems that occur frequently in fluid mechanics (driven cavity problem [31, p.316, p.317]).

We consider the convection-diffusion problem on a unit square domain $\Omega = [-1, 1]^2$ with the following Dirichlet BC:

$$\begin{aligned} &u(x = 1, -1 \leq y \leq 1) = 1 && \text{(hot wall)} \\ &u = 0\,, \text{ elsewhere on } \partial\Omega && \text{(cold wall).} \end{aligned} \tag{4.9}$$



This is a model for a heat exchange system; for example when air circulates between glass panels of a double glazed window, or a nuclear reactor cooling system. The convection flow used in this case is

$$\vec{w} = (2y(1 - x^2), -2x(1 - y^2)),$$

shown in Figure 4.5(a). Figure 4.5 also presents the solution u as a contour plot for $Pe_* = 0$, 500 and $10,000$, using a zero source term $f = 0$.

In Figure 4.5(b) we see a diffusion model ($Pe_* = 0$) where heat from the hot wall diffuses to more colder areas of the domain, shown by the heat colour spectrum. In Figure 4.5(c), where $Pe_* = 500$, the transfer of heat by convection becomes dominant. The solution exhibits two boundary layers, one close to the "hot" wall, and the other close to the "cold" wall $y = -1$. This is due to the shape of the wind function, which moves hot fluid close to the "cold" wall at $y = -1$. The width of the boundary layers is thought to be proportional to $\sqrt{(\frac{1}{Pe_*})}$ [31, p.119], which can be seen in Figure 4.5(d) obtained for $Pe_* = 10,000$.

Following the results from [44], where improved performance of point or block Gauss-Seidel smoothers is obtained using a "black-box" downwind ordering of the unknowns, it would be of interest to examine the potential benefits when the same reordering is used for tILU$_0$ smoothing. Certain ordering algorithms aim to minimise the amount of "fill-in" [78] (for example reverse Cuthill McKee [23, Section 8] and minimum degree ordering [17, Section 7.1]). Intuitively this implies that ILU$_0$ factorisation of a reordered matrix would be closer to LU factorisation than the ILU$_0$ factorisation of the matrix without reordering. This should make the smoother more robust and effective than for any other ordering. The authors of [5] show the performance of these ordering strategies when ILU$_0$ is used as a preconditioner to GMRES(20). However these reordering algorithms work only on information based on static sparsity patterns and do not take into account the numerical values of the matrix coefficients.

We will now take a closer look at the effect that ordering of the unknowns can have on the performance of smoothers within the GMG preconditioner. In Table 4.5

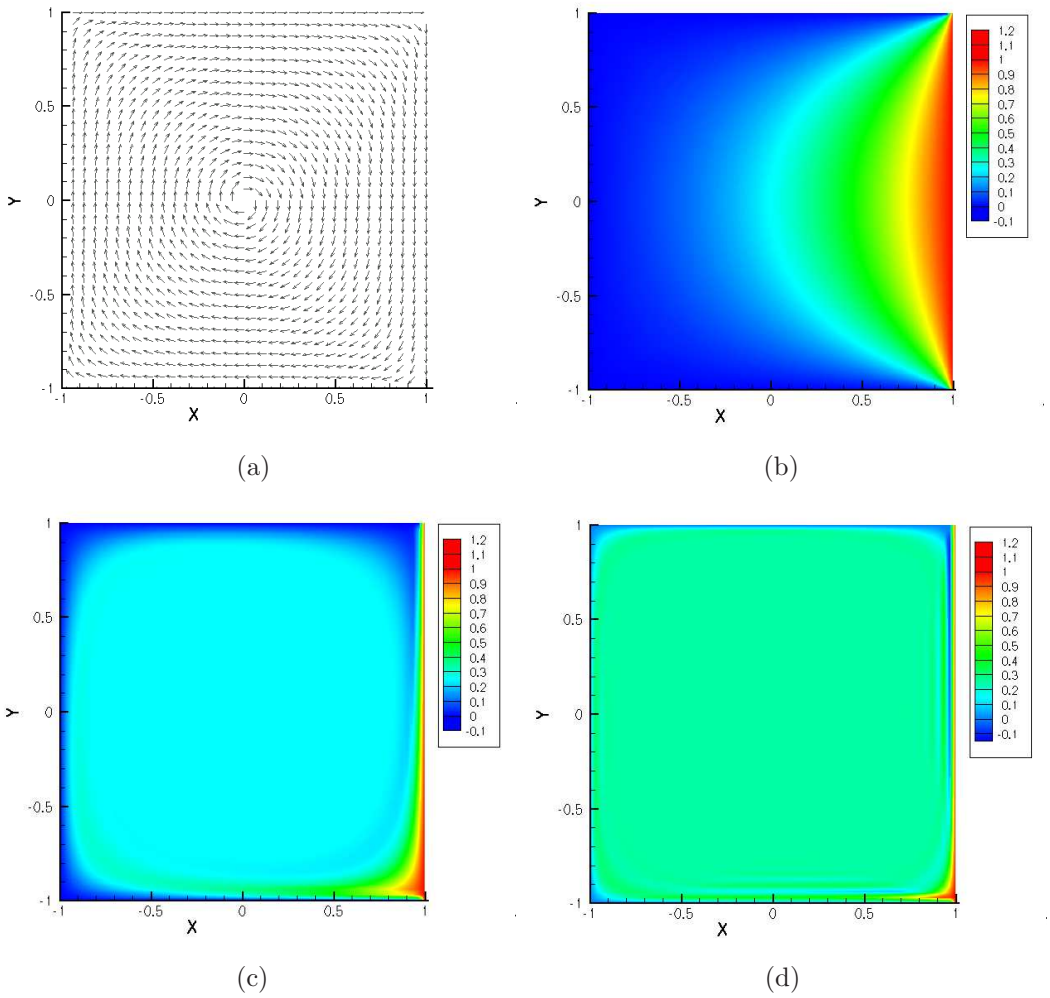


Figure 4.5: The convection-diffusion problem (Case study 4.2.2): recirculating wind $\vec{w} = (2y(1-x^2), -2x(1-y^2))$, no source term ($f = 0$); (a) Arrow plot of the wind. (b) the solution for $Pe_* = 0$, (c) the solution for $Pe_* = 500$, (d) the solution for $Pe_* = 10,000$.

we examine five different methods of unknown ordering, for Case study 4.2.2 with $Pe_* = 500$.

In Table 4.5(a) we use a MG V-cycle $V(2,2)$ with default ordering of the unknowns. The results in this table show an increase in the total number of iterations that is required to solve the linear system of Case study 4.2.2, in comparison to Case studies 4.2.1 and 4.1. This is, in part, expected as the convection field in Case study 4.2.2 is considerably more complex than in Case study 4.2.1.

The default ordering of the Gauss-Seidel smoother in comparison to Case study 4.2.1 grows from 6 to 12 iterations. By contrast, the performance of GMG with the ILU_0 smoother is $\vec{w}$-robust, with an increase of 1 to 2 iterations. This implies that ILU_0 has

Table 4.5: The iteration counts and the (setup,total) execution time (in seconds) required for the convergence of the GMRES solver right-preconditioned by geometric multigrid with several smoothers, when applied to the solution of the discrete convection-diffusion problem (Case study 4.2.2) obtained from $Q1$ SUPG FEM on uniform grids. A constant diffusion parameter ($Pe_* = 500$), with each sub-table representing a different ordering of the unknowns.

(a) OOMPHLIB natural ordering; MG V-cycle V(2,2)

Smoother \ N	3969	16129	65025	261121	1046529
$ILU_0(0.5)$	10(0.77,1.02)	9(3.49,4.53)	9(15.1,21.0)	8(60.2,82.6)	7(256.6,346.7)
$tILU_0(0.5, 0.25)$	18(0.80,1.03)	16(3.47,4.39)	15(15.2,19.6)	14(60.5,79.4)	13(266.6,354.1)
$tILU_0(0.5, 0.5)$	28(0.77,1.09)	27(3.49,4.92)	25(15.3,22.0)	23(60.5,89.1)	21(265.1,399.2)
$Jacobi(0.5)$	75(0.77,1.72)	69(3.45,7.56)	64(5.19,37.3)	59(60.7,155.)	54(262.4,676.6)
$GaussSeidel$	16(0.80,0.95)	14(3.47,4.05)	13(15.1,18.7)	12(60.3,75.7)	12(257.3,330.2)

(b) x-directional ordering ($\breve{D}^{(\rightarrow)}$, $\breve{D}^{(\rightarrow)}$,$\breve{D}^{(\rightarrow)}$, $\breve{D}^{(\rightarrow)}$) ; MG V-cycle V(4,4)

Smoother \ N	3969	16129	65025	261121	1046529
$ILU_0(0.5)$	6(0.81,1.26)	6(3.74,5.97)	5(17.2,27.5)	5(75.3,120.3)	5(343.0,541.4)
$tILU_0(0.5, 0.25)$	11(0.83,1.12)	10(3.77,4.95)	10(17.3,23.6)	9(75.5,102.7)	8(343.9,451.3)
$tILU_0(0.5, 0.5)$	17(0.82,1.18)	16(3.78,5.34)	15(17.3,24.9)	14(75.8,109.9)	13(345.1,495.0)
$Jacobi(0.5)$	45(0.82,1.67)	41(3.79,7.51)	38(16.2,38.3)	35(67.0,168.5)	32(277.2,677.4)
$GaussSeidel$	12(0.81,0.95)	11(3.76,4.36)	10(17.2,21.0)	10(75.1,92.59)	9(340.4,415.1)

(c) xy-forward-directional ordering ($\breve{D}^{(\rightarrow)}$, $\breve{D}^{(\uparrow)}$,$\breve{D}^{(\rightarrow)}$, $\breve{D}^{(\uparrow)}$); MG V-cycle V(4,4)

Smoother \ N	3969	16129	65025	261121	1046529
$ILU_0(0.5)$	6(0.86,1.34)	6(3.94,6.18)	5(21.2,34.0)	5(75.6,120.3)	5(338.6,533.4)
$tILU_0(0.5, 0.25)$	11(0.87,1.18)	10(3.96,5.18)	9(21.1,28.7)	9(75.4,102.3)	8(340.6,456.6)
$tILU_0(0.5, 0.5)$	17(0.88,1.25)	16(3.93,5.54)	15(21.3,31.2)	14(75.3,109.7)	13(376.9,548.1)
$Jacobi(0.5)$	45(0.82,1.67)	41(3.79,7.51)	38(16.2,38.3)	35(67.0,168.5)	32(277.2,677.4)
$GaussSeidel$	12(0.87,1.01)	11(3.95,4.55)	10(21.0,26.2)	9(75.1,90.9)	9(341.5,413.3)

(d) xy-forward-xy-backward-directional ordering ($\breve{D}^{(\rightarrow)}$, $\breve{D}^{(\uparrow)}$,$\breve{D}^{(\leftarrow)}$, $\breve{D}^{(\downarrow)}$); MG V-cycle V(4,4)

Smoother \ N	3969	16129	65025	261121	1046529
$ILU_0(0.5)$	6(0.81,1.27)	6(3.76,6.11)	6(17.4,29.6)	5(75.2,121.5)	5(327.4,519.7)
$tILU_0(0.5, 0.25)$	11(0.81,1.10)	10(4.05,5.33)	9(17.4,23.5)	9(75.6,103.1)	8(326.7,426.4)
$tILU_0(0.5, 0.5)$	17(0.81,1.18)	16(3.99,5.66)	15(17.4,25.3)	14(75.4,110.1)	13(331.6,471.5)
$Jacobi(0.5)$	45(0.82,1.67)	41(3.79,7.51)	38(16.2,38.3)	35(67.0,168.5)	32(277.2,677.4)
$GaussSeidel$	11(0.81,0.93)	9(3.74,4.23)	9(17.4,20.5)	8(75.1,88.96)	7(329.5,383.1)

(e) "Black-box" ordering (HSL_MC13 Tarjan's algorithm); MG V-cycle V(2,2)

Smoother \ N	3969	16129	65025	261121	1046529
$ILU_0(0.5)$	21(0.88,1.83)	19(4.60,8.94)	18(28.4,51.4)	17(261.4,359.4)	15(3882,4209)
$tILU_0(0.5, 0.25)$	28(0.89,1.40)	26(4.60,6.93)	24(28.5,39.9)	22(261.5,313.0)	20(3888,4058)
$tILU_0(0.5, 0.5)$	35(0.89,1.43)	33(4.60,7.10)	31(28.4,40.7)	29(261.6,319.3)	26(3751,3942)
$Jacobi(0.5)$	75(0.77,1.72)	69(3.45,7.56)	64(5.19,37.3)	59(60.7,155.)	54(262.4,676.6)
$GaussSeidel$	26(0.89,1.13)	23(4.58,5.66)	21(28.3,34.7)	20(261.0,290.4)	18(3993,4091)

the potential to be a robust smoother for MG preconditioning of discrete convection-dominated problems. Also, there is a noticeable increase in the iteration count between Case study 4.2.1 and 4.2.2 (by approximately a factor of 3) when damped Jacobi is used as a smoother.

The results obtained for tILU$_0$(0.5,α) show a clear mesh independence for both

levels of truncation ($\alpha = 0.25$ and $\alpha = 0.5$). However $\alpha = 0.25$ gives a more h-robust smoother at a smaller iteration count. The increase in the iteration count of Case study 4.2.2 compared to 4.2.1 is approximately 30% for $\alpha = 0.25$ and approximately double for $\alpha = 0.5$. Moreover, the behaviour of tILU$_0$(0.5,0.25) smoother is fairly similar to that of ILU$_0$

Next we turn our attention to the effects that different node orderings have on the performance of our smoothers. In Figure 4.3 we see how OOMPHLIB globally indexes the nodes in a uniform 2D grid, based on a tree structure strategy. We can obtain a standard lexicographical ordering of the nodes, by taking a permutation of the unknowns. This is achieved by introducing a lookup vector, which for a simple example shown in Figure 4.3 has the form

$$\begin{array}{lll}
\text{Forward X-Directional:} & \breve{D}^{(\rightarrow)} & = (0\ \ 1\ \ 4\ \ 2\ \ 3\ \ 5\ \ 6\ \ 7\ \ 8), \\
\text{Forward Y-Directional:} & \breve{D}^{(\uparrow)} & = (0\ \ 2\ \ 6\ \ 1\ \ 3\ \ 7\ \ 4\ \ 5\ \ 8), \\
\text{Backward X-Directional:} & \breve{D}^{(\leftarrow)} & = (8\ \ 7\ \ 6\ \ 5\ \ 3\ \ 2\ \ 4\ \ 1\ \ 0), \\
\text{Backward Y-Directional:} & \breve{D}^{(\downarrow)} & = (8\ \ 5\ \ 4\ \ 7\ \ 3\ \ 1\ \ 6\ \ 2\ \ 0).
\end{array}$$

The lookup vector is used to indirectly access the correct entries in the coefficient matrix, which is assembled using default ordering. This approach is more memory efficient than explicitly reordering the coefficient matrix. However indirect addressing will result in a deterioration in the execution performance.

In the case of non-uniform wind with variable directions, it is difficult to order the nodes in the direction of the wind. This is the reason why a single downwind ordering is replaced by multiple sweeps of lexicographical ordering in different directions.

In Table 4.5(b)(c)(d) we report the iteration counts of Solver Strategy 4.2 with V$(4, 4)$ cycle. That is, 4 sweeps of a smoother. The increase in the number of sweeps is necessary in order to perform a fair comparison with each table (that have different node orderings), to approximately have the same amount of computational work associated with the preconditioner in each case.

In Table 4.5(b) forward x-directional lexicographical ordering is used. Once again we find that the ILU$_0$ smoother leads to the smallest iteration counts and a preconditioner that is h-robust. However due to the computational complexity of ILU$_0$ smoothing, the total execution times do not look competitive. As is expected there is a small increase in the iteration count of tILU$_0(0.5, 0.25)$ in comparison to ILU$_0$. Although the difference between the iteration counts of ILU$_0$ and tILU$_0(0.5, 0.25)$ is slightly smaller when x-directional lexicographical ordering is used, compared to the case of default ordering. This implies that tILU$_0(0.5, 0.25)$ smoothing is slightly more sensitive to the ordering than ILU$_0$ smoothing. However tILU$_0(0.5, 0.25)$ has a considerable reduction in the total solve time compared to ILU$_0$.

Both $\text{tILU}_0(0.5, 0.25)$ and Gauss-Seidel lead to roughly the same iteration counts, with the execution time of GMRES/GMG/Gauss-Seidel being $\approx 10\%$ better than that of GMRES/GMG/$\text{tILU}_0(0.5, 0.25)$ in Table 4.5(a) and (b).

Instead of having one or a few sweeps of a smoother in a single Cartesian direction, one can perform sweeps of a smoother applied to alternating lexicographical ordering of the unknowns. This way, instead of smoothing the error components in a single direction, we smooth the components of the error over multiple directions.

In Table 4.5(c) alternating forward Cartesian directional sweeps ($\breve{D}^{(\rightarrow)}$, $\breve{D}^{(\uparrow)}$,$\breve{D}^{(\rightarrow)}$, $\breve{D}^{(\uparrow)}$) are used. Comparing the results from part(b) and (c) of Table 4.5, we conclude that forward alternating directional ordering and forward x-directional ordering of the unknowns have no difference in performance for the smoothers we consider. Extending the ordering to include all four Cartesian directions (see Table 4.5(d)) leads to a further reduction in iteration counts for the case of Gauss-Seidel smoothing, but appears to have no effect when other smoothers are used. The iteration counts obtained by tILU_0 for $\alpha = 0.25$ and $\alpha = 0.5$ in Table 4.5(b), (c) and (d) appear to have no dependence on the ordering of the unknowns.

Next we want to examine whether nodal ordering, using algorithms designed to reduce "fill-in" during sparse Gauss-elimination, will have any beneficial impact to our smoothers in the case of convection-dominated problems. The authors [44] suggest that "black-box" reordering, based on variants of the minimum degree ordering algorithm may coincide with downwind ordering and therefore improve the efficiency of smoothers during convection-dominated problems. This is a very useful concept when no geometric information is available to perform downwind reordering. Such circumstances arise during AMG preconditioning, and thus a "black-box" reordering is the only available option, apart from default ordering.

In this context we experimented with Tarjan's algorithm [82]. As was the case for the previous directional orderings we use a lookup vector to implement a permutation to the coefficient matrix through indirect addressing. However this lookup vector is now created from a permutation of the convection matrix only (Definition 2.4.8). A permutation of the convection matrix is achieved by passing the non-zero matrix entries' coordinates into the `MC13` HSL algorithm [1]. This is applied to all matrices in the MG hierarchy. In Table 4.5(e) we report the results of Solver Strategy 4.2 with V(2,2) cycle, ordered using Tarjan's algorithm. Comparing these results with the results from Table 4.5(a), we observe a significant deterioration in iteration counts, with the iteration counts doubling for the ILU_0 smoother. However, this is not the case for the Jacobi smoother, as it is unaffected by ordering.

In Table 4.6 we present the same set of results as in Table 4.5, but for a larger value of $Pe_* = 2000$. Comparing the corresponding cases between the two tables,

Table 4.6: The iteration counts and the (setup,total) execution time (in seconds) required for the convergence of the GMRES solver right-preconditioned by geometric multigrid with several smoothers, when applied to the solution of the discrete convection-diffusion problem (Case study 4.2.2) obtained from $Q1$ SUPG FEM on uniform grids. A constant diffusion parameter ($Pe_* = 2000$), with each sub-table representing a different ordering of the unknowns.

(a) OOMPHLIB natural ordering; MG V-cycle V(2,2)

Smoother \ N	3969	16129	65025	261121	1046529
$ILU_0(0.5)$	17(0.78,1.19)	17(3.44,5.32)	16(15.1,24.8)	14(60.3,97.6)	13(259.8,415.9)
$tILU_0(0.5, 0.25)$	43(0.78,1.36)	41(3.45,6.08)	38(15.2,27.3)	35(60.7,114.8)	32(262.8,492.5)
$tILU_0(0.5, 0.5)$	61(0.79,1.58)	65(3.46,7.47)	62(15.3,34.7)	58(60.5,150.6)	53(260.9,660.0)
$Jacobi(0.5)$	189(0.77,4.39)	194(3.46,20.7)	180(15.3,99.6)	166(60.3,416.1)	153(260.1,1939)
$GaussSeidel$	39(0.79,1.16)	36(3.45,5.05)	35(15.2,25.1)	29(60.3,98.0)	27(263.4,445.3)

(b) x-directional ordering ($\breve{D}^{(\rightarrow)}$, $\breve{D}^{(\rightarrow)}$,$\breve{D}^{(\rightarrow)}$, $\breve{D}^{(\rightarrow)}$); MG V-cycle V(4,4)

Smoother \ N	3969	16129	65025	261121	1046529
$ILU_0(0.5)$	10(0.82,1.52)	9(3.75,6.94)	9(17.1,34.2)	8(75.4,142.7)	7(372.9,662.4)
$tILU_0(0.5, 0.25)$	27(0.82,1.57)	25(3.77,6.90)	22(17.2,31.4)	20(75.8,141.3)	19(375.5,696.7)
$tILU_0(0.5, 0.5)$	37(0.81,1.62)	38(3.75,7.64)	36(17.2,36.3)	33(75.4,167.1)	31(371.2,834.1)
$Jacobi(0.5)$	116(0.83,3.52)	116(3.78,17.0)	107(16.2,87.1)	98(67.1,401.8)	89(278.6,1611.)
$GaussSeidel$	29(0.81,1.14)	28(3.74,5.31)	29(17.1,27.7)	24(75.5,117.2)	22(377.2,579.4)

(c) xy-forward-directional ordering ($\breve{D}^{(\rightarrow)}$, $\breve{D}^{(\uparrow)}$,$\breve{D}^{(\rightarrow)}$, $\breve{D}^{(\uparrow)}$); MG V-cycle V(4,4)

Smoother \ N	3969	16129	65025	261121	1046529
$ILU_0(0.5)$	11(0.87,1.67)	13(3.92,8.41)	20(17.5,53.6)	20(75.6,233.5)	18(374.7,1016)
$tILU_0(0.5, 0.25)$	26(0.87,1.64)	24(3.96,7.09)	21(17.5,31.0)	19(75.7,138.0)	18(357.3,664.5)
$tILU_0(0.5, 0.5)$	37(0.86,1.70)	38(4.00,8.08)	36(17.3,36.3)	33(75.5,168.8)	31(355.7,785.0)
$Jacobi(0.5)$	116(0.83,3.52)	116(3.78,17.0)	107(16.2,87.1)	98(67.1,401.8)	89(278.6,1611.)
$GaussSeidel$	30(0.86,1.22)	42(4.05,6.57)	75(21.2,60.0)	84(75.6,242.2)	75(373.8,1191.)

(d) xy-forward-xy-backward-directional ordering ($\breve{D}^{(\rightarrow)}$, $\breve{D}^{(\uparrow)}$,$\breve{D}^{(\leftarrow)}$, $\breve{D}^{(\downarrow)}$); MG V-cycle V(4,4)

Smoother \ N	3969	16129	65025	261121	1046529
$ILU_0(0.5)$	11(0.81,1.62)	13(3.94,8.51)	20(17.5,54.4)	20(75.4,238.4)	18(362.2,1012.)
$tILU_0(0.5, 0.25)$	26(0.82,1.57)	24(3.89,7.11)	21(17.4,31.4)	19(75.8,138.9)	18(351.3,623.3)
$tILU_0(0.5, 0.5)$	37(0.83,1.63)	38(3.76,7.64)	36(17.4,37.0)	33(75.7,165.5)	31(356.1,757.4)
$Jacobi(0.5)$	116(0.83,3.52)	116(3.78,17.0)	107(16.2,87.1)	98(67.1,401.8)	89(278.6,1611.)
$GaussSeidel$	31(0.82,1.16)	41(3.93,6.33)	67(17.3,42.5)	74(75.3,220.6)	69(367.5,1038.)

(e) "Black-box" ordering (HSL_MC13 Tarjan's algorithm); MG V-cycle V(2,2)

Smoother \ N	3969	16129	65025	261121	1046529
$ILU_0(0.5)$	47(0.89,3.05)	47(4.61,15.3)	44(28.4,84.0)	41(261.6,494.3)	38(3926.,4794.)
$tILU_0(0.5, 0.25)$	67(0.91,2.40)	67(4.61,11.7)	62(28.4,61.8)	58(261.5,414.7)	54(3901.,4478.)
$tILU_0(0.5, 0.5)$	82(0.92,2.40)	85(4.59,12.4)	80(28.4,66.7)	74(261.5,435.4)	68(3919.,4646.)
$Jacobi(0.5)$	189(0.77,4.39)	194(3.46,20.7)	180(15.3,99.6)	166(60.3,416.1)	153(260.1,1939.)
$GaussSeidel$	65(0.89,1.58)	63(4.58,8.15)	57(28.4,47.0)	53(261.2,341.7)	49(4011.,4421.)

(f) OOMPHLIB natural ordering; MG V-cycle V(1,1)

Smoother \ N	3969	16129	65025	261121	1046529
$GaussSeidel$	62(0.87,1.44)	60(3.81,6.82)	66(16.8,35.9)	54(69.2,142.4)	51(290.4,642.2)

(g) x-directional ordering ($\breve{D}^{(\rightarrow)}$); MG V-cycle V(1,1)

Smoother \ N	3969	16129	65025	261121	1046529
$GaussSeidel$	68(0.85,1.49)	74(4.04,7.98)	96(17.2,49.2)	99(72.4,244.5)	90(304.0,1044.)

we observe that in all cases there is a considerable increase in iteration counts when Pe_* is increased. This growth appears the smallest in the case of the ILU_0 smoother. When default ordering is used (part(a)), the iteration counts roughly doubled when the ILU_0 smoother is used and triple when the Jacobi smoother is used (for this case the number of iterations is already impractical when $Pe_* = 2000$). In the case of tILU_0 and Gauss-Seidel smoothers there is an increase in the iteration counts by a factor of 2.5 to 3. We emphasise that the shortest execution time is obtained when the ILU_0 smoother is used, due to the fact that the incomplete factorisation time is compensated by a considerably smaller number of GMRES iterations, compared to the cases when other smoothers are used. In Table 4.6(f) we present the results for the Gauss-Seidel smoother with V(1,1) MG cycle, which demonstrates that the iteration counts have increased by a factor of 2 when compared to a V(2,2) MG cycle. It is also clear that in Table 4.6(f) there is a small dependence on h, however, by increasing the number of smoothing iterations in a cycle this dependence is reduced.

Next we turn our attention to the case when different ordering strategies are used. In Table 4.6(b) we note a considerable improvement in the performance of Gauss-Seidel smoother (by contrast, using V(1,1) MG cycle in Table 4.6(g) the preconditioner exhibits considerable h-dependence before the iteration count asymptotically settles to a constant). However, the asymptotic cost of V(1,1) and V(4,4) preconditioners is consistent (in that the former requires 4 times as many iterations, being roughly 4 times computationally cheaper). What may come as a surprising result is that the application of V(4,4) GMG cycle with Gauss-Seidel smoother performs considerably better when default or x-directional ordering of unknowns is used, than when alternating ordering of the unknowns is used. Although, the h-dependence shown in Table 4.6(g) seems to have a direct correlation.

In the context of directional ordering, tILU_0 smoothers show virtually no difference in performance with respect to the ordering of the unknowns, and, except in the case of x-directional ordering, MG preconditioning with tILU_0(0.5,0.25) smoother outperform significantly the MG preconditioner with Gauss-Seidel smoothing.

In summary it is clear that an appropriate ordering of the nodes can have the desired effects of reducing the iteration counts. However, as demonstrated in Table 4.6 the orderings need to be paired with the adequate number of relaxation sweeps. On occasions $\text{tILU}_0(0.5, 0.25)$ was shown to have the smallest total solve time and out performed Gauss-Seidel on iteration counts. The disadvantage in using ordering nodes is the extra computational cost of making a lookup vector. Also, in the case of ILU, changing the nodal ordering meant re-factorising the matrices to take into consideration the new nodal ordering. A more efficient implementation of the ILU smoother with multi-directional ordering would require a considerably larger memory.

In the 1980s MG preconditioning of anisotropic elliptic problem appears to be the main challenge for demonstrating the robustness of a smoother, with respect to anisotropic diffusion. In [83] Thole and Trottenberg use MG with standard full coarsening and simple-point relaxation. For highly anisotropic diffusion this results in smoothing only the dominant direction. Such a combination will produce an inefficient MG preconditioner. To improve the robustness of this preconditioning methodology a combination of semi-coarsening and line smoothing is proposed in this context. The idea is to coarsen the grid only in the direction of dominant diffusion or to apply line smoothing in the same direction. In [31, pp.178–197] the use of alternating directional line smoothers is demonstrated to give a robust GMG preconditioner for the convection-diffusion problem. However, for the case of AMG preconditioning directional line smoothing cannot be achieved as there is no geometric information. In Tables 4.5 and 4.6 a prominent feature of tILU_0 is that the smoother gives a similar performance of the MG preconditioner irrespectively of the nodal ordering. Within the context of MG semi-coarsening, AMG uses standard Ruge-Stüben coarsening which generates semi-coarsening in the characteristic directions [97], which for the convection-diffusion problem is the downwind direction.

Stretched grids

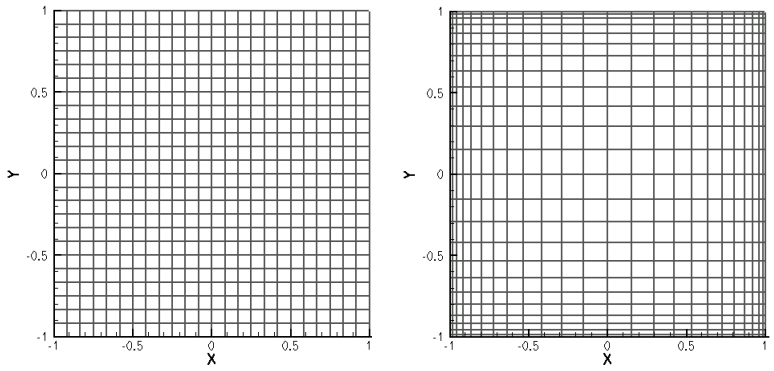
The accuracy of the discrete solution of a problem depends on the quality of the underlying grid and the order of approximation. This especially applies to convection-diffusion problems where boundary layers can occur. In the case when the structure of the solution is known in advance, a grid can be constructed in such a way that good accuracy of the discrete solution is achieved, even with moderate grid resolution and standard Galerkin FEM [31, p.140, p.141] [42, p.133].

The concept behind using stretched grids in the context of the convection-diffusion problem is that, using a priori knowledge or simply a guess about the nature of the solution (e.g. part of the domain close to the Dirichlet boundary), we can construct a grid which will have nodes concentrated in these areas of the domain and fewer nodes in areas of the domain where there are no significant changes in the solution.

Grid stretching introduces elements of distorted shape (in the case of triangular grids) or large aspect ratios (in the case of quadrilateral grids). This situation leads to discrete convection-diffusion operators that have considerably different matrix stencils than was obtained from the discretisation found in uniform grids (Section 2.4). This, in turn, causes problems for MG solvers due to greater positive off-diagonal entries, reduced diagonal dominance and further loss of symmetry. Having a robust MG smoother, if standard coarsening is used, is key to tackling these problems.

In Figure 4.6(a) a uniform quadrilateral mesh is shown. In Figure 4.6(b) we

present a stretched grid with the same number of points, and more nodes concentrated towards the domain boundary.



(a) (b)

Figure 4.6: Discretisation of the domain $\Omega = [-1, 1]^2$ with (a) uniform grid; (b) stretched grid with a stretching parameter $a = 0.9$ ($N = 529$ in both cases).

The stretching of a mesh in Figure 4.6(b) is done by initially uniformly refining the mesh to the desired resolution. The transformation (4.10) is then applied to recalculate the position of each node in the domain interior:

$$\begin{aligned} &\text{if (not on } x \text{ boundary)} \\ &\qquad\qquad x = x - (1+x)|xa| \quad (x < 0) \\ &\qquad\qquad x = x + (1-x)|xa| \quad (x \geq 0), \\ &\text{if (not on } y \text{ boundary)} \\ &\qquad\qquad y = y - (1+y)|ya| \quad (y < 0) \\ &\qquad\qquad y = y + (1-y)|ya| \quad (y \geq 0), \end{aligned} \tag{4.10}$$

where $a \in [0, 1]$ is the parameter that determines the amount of stretching.

In Table 4.7 we present the convergence characteristics of Solver Strategy 4.2, where the convection-diffusion problem (Case study 4.2.2) is discretised using SUPG FEM with Q_1 approximation on a sequence of stretched grids with the parameter a set to 0.9. From the table it can be seen that ILU_0 smoother performs h-robust, however the solver shows moderate dependence on Pe_* (as $O(\sqrt{Pe_*})$). What is interesting is that the solver with the tILU_0(0.5,0.25) smoother shows the same robustness, but has up to 15% shorter execution time. Also, tILU(0.5,0.25) smoother shows competitive execution times, compared to other smoothers considered in the project. When the damped Jacobi smoother is used, the resulting iterative solver exhibits a much worse asymptotic behaviour with respect to Pe_* than the iterative solver with ILU_0/tILU_0 smoothers (the iteration counts grow essentially as $O(Pe_*)$). In the cases when the

Table 4.7: The iteration counts and the (setup,total) execution time (in seconds) required for the convergence of the GMRES solver right-preconditioned by geometric multigrid V(2,2) cycle with several smoothers, when applied to the solution of the discrete convection-diffusion problem (Case study 4.2.2) obtained from $Q1$ SUPG FEM on stretched grids. Natural (tree-based) ordering of the unknowns, with each sub-table representing a different diffusion parameter.

(a) Pe_*=500

Smoother \ N	2209	9025	36481	146689
$ILU_0(0.5)$	9(0.43,0.55)	8(1.99,2.52)	8(8.13,10.7)	7(33.7,45.1)
$tILU_0(0.5, 0.25)$	10(0.43,0.51)	10(1.99,2.33)	9(8.15,9.57)	8(35.2,42.2)
$tILU_0(0.5, 0.5)$	20(0.45,0.57)	20(2.03,2.61)	19(8.16,10.7)	19(33.6,45.3)
$Jacobi(0.5)$	36(0.43,0.64)	34(1.98,3.00)	30(8.10,12.7)	27(33.6,53.7)
$GaussSeidel$	12(0.43,0.50)	11(1.91,2.18)	10(8.11,9.46)	11(33.5,40.7)

(b) Pe_*=1,000

Smoother \ N	2209	9025	36481	146689
$ILU_0(0.5)$	12(0.43,0.59)	11(1.97,2.66)	10(8.00,11.1)	9(35.3,49.7)
$tILU_0(0.5, 0.25)$	14(0.43,0.53)	13(2.00,2.43)	12(8.02,9.82)	11(35.1,44.1)
$tILU_0(0.5, 0.5)$	29(0.43,0.61)	30(1.98,2.89)	28(8.09,11.9)	26(34.0,51.7)
$Jacobi(0.5)$	56(0.42,0.79)	56(1.97,3.93)	50(8.03,16.2)	44(36.6,78.1)
$GaussSeidel$	17(0.44,0.52)	21(2.01,2.50)	21(8.02,10.7)	18(35.5,49.0)

(c) Pe_*=2,000

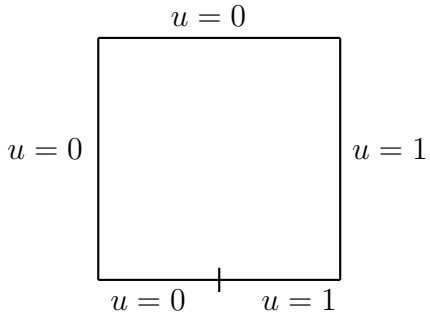
Smoother \ N	2209	9025	36481	146689
$ILU_0(0.5)$	15(0.43,0.62)	15(1.97,2.92)	14(8.01,12.3)	13(38.5,61.8)
$tILU_0(0.5, 0.25)$	20(0.44,0.58)	20(1.98,2.68)	17(8.25,10.9)	15(39.2,52.7)
$tILU_0(0.5, 0.5)$	39(0.43,0.69)	44(2.01,3.46)	43(8.30,14.7)	40(35.8,64.8)
$Jacobi(0.5)$	86(0.43,1.05)	91(1.99,5.84)	84(8.66,26.2)	73(34.9,97.8)
$GaussSeidel$	24(0.43,0.56)	43(2.09,3.19)	71(8.05,18.6)	102(34.0,122)

Gauss-Seidel smoother is used, the iterative solver loses its independence on the problem size in highly convective cases.

4.2.3 Combined uni-directional and recirculating wind

Here we combine the convection fields studied in Case study 4.2.1 and 4.2.2 to create a new complex convection flow. In this case we solve the convection-diffusion problem (2.7) on a square domain $\Omega = [-1, 1]^2$, subject to the following combination of the Dirichlet BCs:

$$\begin{aligned} u = 0 \quad &\text{for} \quad [-1 \leq x < 0, y = -1] \bigcup [-1 \leq x \leq 1, y = 1] \bigcup [x = -1, -1 \leq y \leq 1] \\ u = 1 \quad &\text{for} \quad [0 \leq x \leq 1, y = -1] \bigcup [x = 1, -1 \leq y < 1]. \end{aligned} \tag{4.11}$$



The convection field $\vec{w}$ used for this case study consists of two components:

$$\begin{aligned} &\vec{w}_u = (\cos(\tfrac{2\pi}{3}), \sin(\tfrac{2\pi}{3})) && \text{(uni-directional)}, \\ &\vec{w}_c = (2y(1-x^2), -2x(1-y^2)) && \text{(circular)}, \\ &\vec{w} = \vec{w}_u - \vec{w}_c, \end{aligned} \tag{4.12}$$

and is shown in Figure 4.7(a). This case study may be considered as a simplified model of costal modelling, where we have a uni-directional $\vec{w}_u$ inflow of a river with a concentration u of a pollutant into a bay with a circular $\vec{w}_c$ (sweeping) sea current.

In Figure 4.7(b),(c) and (d) the contour plots represent the solution u for $Pe_* = 0$, 500 and 10,000. This combination of BCs and convective field results in a rather complicated flow, with steep boundary layers close to $y = 1$ and a concave shaped layer in the domain interior. Thus, getting accurate numerical solutions of the problem can be difficult. Discretisation by uniform grid would require very fine resolution (especially for large Pe_*), and using stretched grids would only resolve the boundary layers at $y = 1$. The best choice here (unless using SUPG method) is to have a FEM based on adaptively refined grids. For this case study we restrict ourselves to Q_1 SUPG FEM discretisation on uniform grids and will explore adaptively refined grids in the next case study.

In Table 4.8 we present the performance of Solver Strategy 4.2 for Case study 4.2.3. For a relatively small inverse diffusion parameter $Pe_* = 500$ (Table 4.8(a)) all smoothers are h-robust. As in the previous examples, the ILU_0 smoother leads to the smallest iteration count, although it has essentially a larger solve time in comparison to the cases when Gauss-Seidel and $\text{tILU}_0(0.5,0.25)$ smoothers are used. As the inverse diffusion parameter increases, the iteration count for all methods increases moderately. In all cases the Gauss-Seidel and $\text{tILU}_0(0.5,0.25)$ smoothers show mesh independence and consistently outperform the other smoothers.

In Table 4.9 we summarise the truncation statistics obtained using (3.43) for several values of α with coefficient matrices at all levels of the MG hierarchy obtained in Case study 4.2.3, for the case of uniform grid refinement. The storage requirements of a MG preconditioner is proportional to number of non-zero entries (NNZ) at each

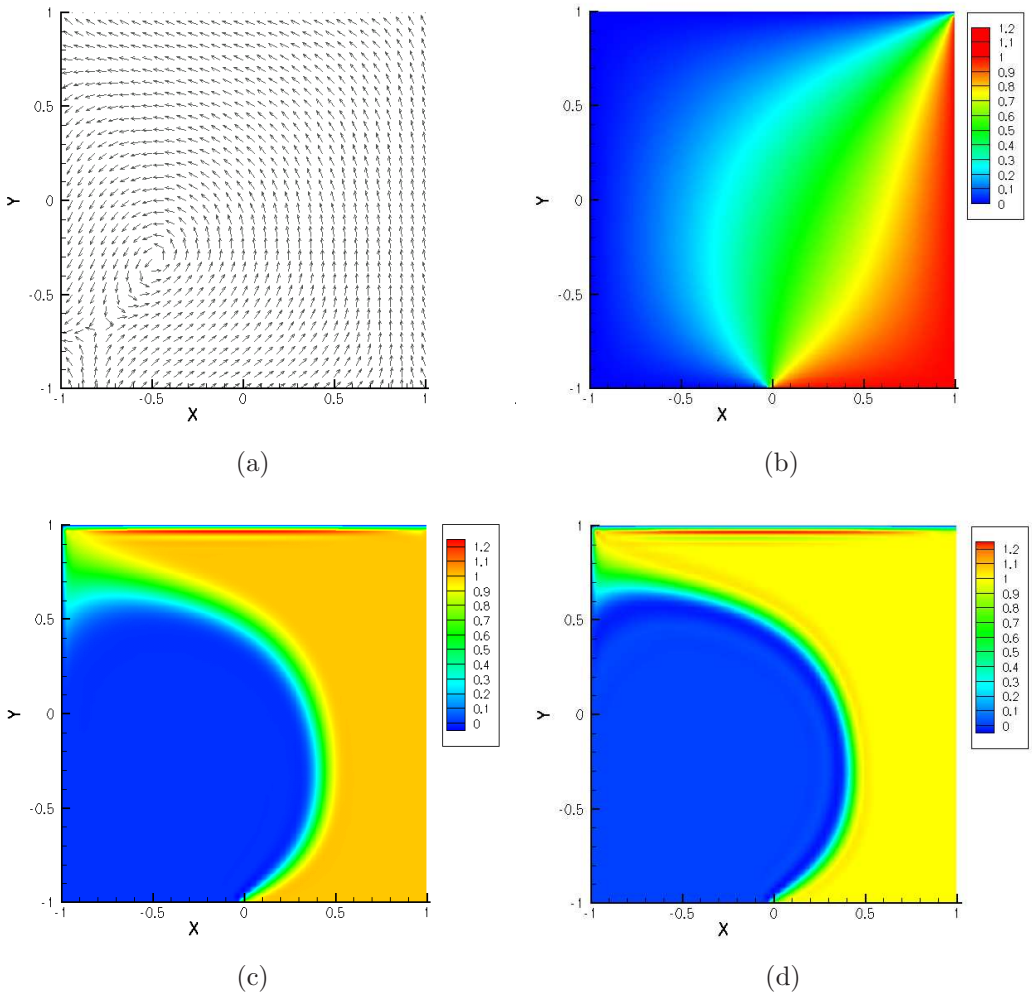


Figure 4.7: The convection-diffusion problem (Case study 4.2.3): no source term ($f = 0$); (a) Arrow plot of the wind, defined by (4.12), (b) the solution for $Pe_* = 0$, (c) the solution for $Pe_* = 500$, (d) the solution for $Pe_* = 10,000$.

hierarchical level (we do not take into consideration the storage requirements of the interpolation operators as they are independent to the level of truncation).

The results in Table 4.9 are presented for three different values of Pe_*. For the extreme case $\alpha = 0$ we have no truncation. Hence the numbers of non-zero entries in matrices $\widetilde{A}_l$ and A_l are the same for $l = 1, ..., L$, and we have the standard ILU_0 method. At the other extreme value of the truncation parameter ($\alpha = 1$) we have Jacobi smoothing, where only the diagonal entries are kept. As we use Q_1 approximation in our discretisation, which results in a 9-point stencil, the asymptotic level of truncation when $\alpha = 1$ should be $\frac{1}{9} \approx 0.11$. This is clearly visible at fine grids, while on coarse grids, due to the larger influence of the boundary nodes this asymptotic ratio is marginally different. The expectation is that by changing the truncation parameter between 0 and 1 we can find a smoother with low cost similar

Table 4.8: The iteration counts and the (setup,total) execution time (in seconds) required for the convergence of the GMRES solver right-preconditioned by geometric multigrid V(2,2) cycle with several smoothers, when applied to the solution of the discrete convection-diffusion problem (Case study 4.2.3) obtained from $Q1$ SUPG FEM on uniform grids. Natural (tree-based) ordering of the unknowns, with each sub-table representing a different diffusion parameter.

(a) Pe_*=500

Smoother \ N	3969	16129	65025	261121
$ILU_0(0.5)$	9(0.80,1.02)	9(3.46,4.49)	8(14.5,19.2)	8(61.5,84.3)
$tILU_0(0.5, 0.25)$	14(0.79,0.98)	11(3.48,4.13)	10(14.5,17.4)	10(61.7,76.9)
$tILU_0(0.5, 0.5)$	21(0.79,1.03)	21(3.47,4.56)	20(14.5,19.4)	20(61.7,87.7)
$Jacobi(0.5)$	32(0.78,1.14)	32(3.48,5.14)	31(14.4,23.1)	29(61.6,104.0)
$GaussSeidel$	10(0.79,0.89)	10(3.49,3.93)	9(14.5,16.7)	8(61.7,72.1)

(b) Pe_*=1,000

Smoother \ N	3969	16129	65025	261121
$ILU_0(0.5)$	10(0.79,1.04)	11(3.48,4.73)	10(14.4,20.1)	9(61.6,86.8)
$tILU_0(0.5, 0.25)$	17(0.79,1.02)	19(3.52,4.66)	14(14.5,18.6)	13(61.8,81.6)
$tILU_0(0.5, 0.5)$	23(0.79,1.05)	30(3.46,5.07)	29(14.5,21.8)	27(61.8,98.3)
$Jacobi(0.5)$	37(0.79,1.20)	41(3.47,5.67)	41(14.4,26.4)	39(61.5,121.0)
$GaussSeidel$	12(0.79,0.90)	14(3.50,4.10)	12(14.4,17.3)	11(61.7,76.0)

(c) Pe_*=2,000

Smoother \ N	3969	16129	65025	261121
$ILU_0(0.5)$	10(0.79,1.03)	13(3.45,4.90)	14(14.5,22.3)	12(61.6,94.5)
$tILU_0(0.5, 0.25)$	18(0.80,1.04)	26(3.49,5.10)	26(14.5,22.3)	18(62.1,90.0)
$tILU_0(0.5, 0.5)$	25(0.80,1.08)	37(3.45,5.52)	42(14.5,25.7)	39(62.1,118.4)
$Jacobi(0.5)$	41(0.79,1.26)	50(3.44,6.20)	54(14.4,30.8)	53(61.6,147.9)
$GaussSeidel$	14(0.80,0.93)	17(3.50,4.26)	19(14.4,19.0)	20(61.5,87.6)

to that of the Jacobi method, but with sufficient robustness (ideally, the same as the ILU_0 method). The amount of entries held at each MG level will depend, in general, on the nature of the convection field $\vec{w}$, Pe_* and the problem itself.

The two values of the parameter α in Table 4.9 are those used in our experiments reported in Table 4.8. The interesting findings from Table 4.9 are that for both $\alpha = 0.25$ and $\alpha = 0.5$ the percentage of truncated entries ($\frac{NNZ(\widetilde{A}_l)}{NNZ(A_l)}$) varies between the refinement levels. This is the consequence of h-dependence in the convection matrix (see matrix stencils in Section 2.4) resulting in the convection part of the operator becoming more prominent at coarser levels. What makes the suggested methodology effective is the fact that most of the non-zero entries at fine levels are truncated. For example, in the case when $Pe_* = 20$ and $\alpha = 0.25$, the truncated matrices at levels $l = 8, 7, 6$ are diagonal matrices resulting in a Jacobi smoother on these levels. This is exactly the opposite from previous work [9] where an expensive (line Gauss-Seidel) smoother is applied at the finest level, followed by the damped Jacobi method at all coarser levels.

Keeping a much larger proportion of the non-zero entries at coarse levels appears to be key to the success of the $tILU_0$ smoother when compared to the simple Jacobi

Table 4.9: Truncation statistics for the convection-diffusion matrices obtained from Q1 SUPG FEM discretisation of Case study 4.2.3 as a function of Pe_* with uniform grid refinement. $\widetilde{A}_l$ denotes the truncated matrix and A_l is the original matrix at the refinement level l. NNZ$(\cdot)$ denotes the number of non-zero entries and $\frac{NNZ(\widetilde{A}_l)}{NNZ(A_l)}[\%]$ the percentage of the non-zero entries kept in the truncated matrices.

(a) Pe_*=20

Level l	8	7	6	5	4	3	2	1	$\sum_l(\text{NNZ}(\widetilde{A}_l))$
N	261121	65025	16129	3969	961	225	49	9	347489
NNZ(A_l)	2343961	582169	143641	34969	8281	1849	361	49	3115281
$\alpha = 0$									
NNZ($\widetilde{A}_l$)	2343961	582169	143641	34969	8281	1849	361	49	3115281
$\frac{NNZ(\widetilde{A}_l)}{NNZ(A_l)}[\%]$	100	100	100	100	100	100	100	100	η =100
$\alpha = 0.25$									
NNZ($\widetilde{A}_l$)	261121	65025	16129	5203	1776	538	122	20	349935
$\frac{NNZ(\widetilde{A}_l)}{NNZ(A_l)}[\%]$	11	11	11	15	21	29	34	41	η =11
$\alpha = 0.5$									
NNZ($\widetilde{A}_l$)	261121	65025	16129	3969	979	298	76	14	347612
$\frac{NNZ(\widetilde{A}_l)}{NNZ(A_l)}[\%]$	11	11	11	11	12	16	21	29	η =11
$\alpha = 1.0$									
NNZ($\widetilde{A}_l$)	261121	65025	16129	3969	961	225	49	9	347489
$\frac{NNZ(\widetilde{A}_l)}{NNZ(A_l)}[\%]$	11	11	11	11	12	12	14	18	η =11

(b) Pe_*=500

Level l	8	7	6	5	4	3	2	1	$\sum_l(\text{NNZ}(\widetilde{A}_l))$
$\alpha = 0.25$									
NNZ($\widetilde{A}_l$)	609501	179900	47298	12514	3183	741	153	21	853312
$\frac{NNZ(\widetilde{A}_l)}{NNZ(A_l)}[\%]$	26	31	33	36	38	40	42	43	η =27
$\alpha = 0.5$									
NNZ($\widetilde{A}_l$)	315940	96176	26391	6843	1698	395	85	14	447543
$\frac{NNZ(\widetilde{A}_l)}{NNZ(A_l)}[\%]$	13	17	18	20	21	21	24	29	η =14

(c) Pe_*=2000

Level l	8	7	6	5	4	3	2	1	$\sum_l(\text{NNZ}(\widetilde{A}_l))$
$\alpha = 0.25$									
NNZ($\widetilde{A}_l$)	769968	207318	54737	13888	3369	759	154	21	1050215
$\frac{NNZ(\widetilde{A}_l)}{NNZ(A_l)}[\%]$	33	36	38	40	41	41	43	43	η =34
$\alpha = 0.5$									
NNZ($\widetilde{A}_l$)	427453	112813	28882	7189	1738	397	85	14	578572
$\frac{NNZ(\widetilde{A}_l)}{NNZ(A_l)}[\%]$	18	19	20	21	21	21	24	29	η =19

smoother. However, for a fixed value of α the amount of truncated entries is reduced as Pe_* is increased. But even for highly convective flows ($Pe_* = 2000$) only an average of 20% of the non-zero entries are kept for $\alpha = 0.5$. This is also true for all case studies introduced in this chapter (Table 4.10).

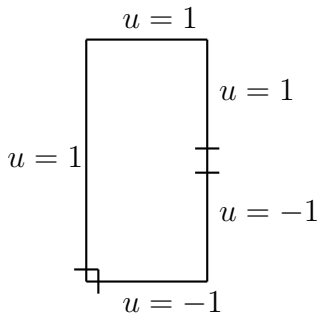
Table 4.10: Comparison of truncation statistics for Case studies 4.2.1, 4.2.2, 4.2.3 and 4.2.4 as a function of Pe_* and the truncation parameter α. The discrete convection-diffusion operators are obtained from Q_1 SUPG FEM discretisation using a uniform grid hierarchy. The results present the total number of non-zero entries in the entire MG hierarchy and, in brackets, the percentage of the total number of non-zero entries in the entire MG hierarchy after truncation.

Pe_*	Case Study	$\alpha = 0$	$\alpha = 0.25$	$\alpha = 0.5$	$\alpha = 1.0$
500	4.2.1	3115281(100)	693965(22)	433355(14)	347489(11)
	4.2.2	3115281(100)	727545(23)	440533(14)	347489(11)
	4.2.3	3115281(100)	853312(27)	447543(14)	347489(11)
	4.2.4	3115281(100)	814001(26)	497052(16)	347489(11)
2000	4.2.1	3115281(100)	693965(22)	693965(22)	347489(11)
	4.2.2	3115281(100)	951577(31)	619169(20)	347489(11)
	4.2.3	3115281(100)	1050215(34)	578572(19)	347489(11)
	4.2.4	3115281(100)	956168(31)	592205(19)	347489(11)

4.2.4 Multiple recirculating wind

As a final case, we consider the convection-diffusion problem defined on a rectangular domain $\Omega = [0, 1] \times [0, 2]$ with a convection field $\vec{w}$ consisting of multiple recirculations. This configuration is physically relevant as it occurs in models of thermally buoyed flows when Rayleigh-Bénard convection is studied [22, Chapter 6]. The specification for this problem is taken from the OOMPHLIB library [46]. The boundary conditions are taken from the analytical solution calculated on $\partial\Omega$:

$$u = \tanh(1.0 - 50 * (x - y)). \tag{4.13}$$



The convection field used in this case is

$$\vec{w} = (\sin(6y), \cos(6x)),$$

shown in Figure 4.8(a), where we can see three complete recirculations, two moving in an anti-clockwise direction and the third in a clockwise direction. Figure 4.8 also represents the solution u as a contour plot for $Pe_* = 0$, 500 and $10,000$, with a zero source term $f = 0$. This combination of BCs and convection field results in a steep solution layer in the interior of the domain. Initially for Case study 4.2.4 we use Q_1

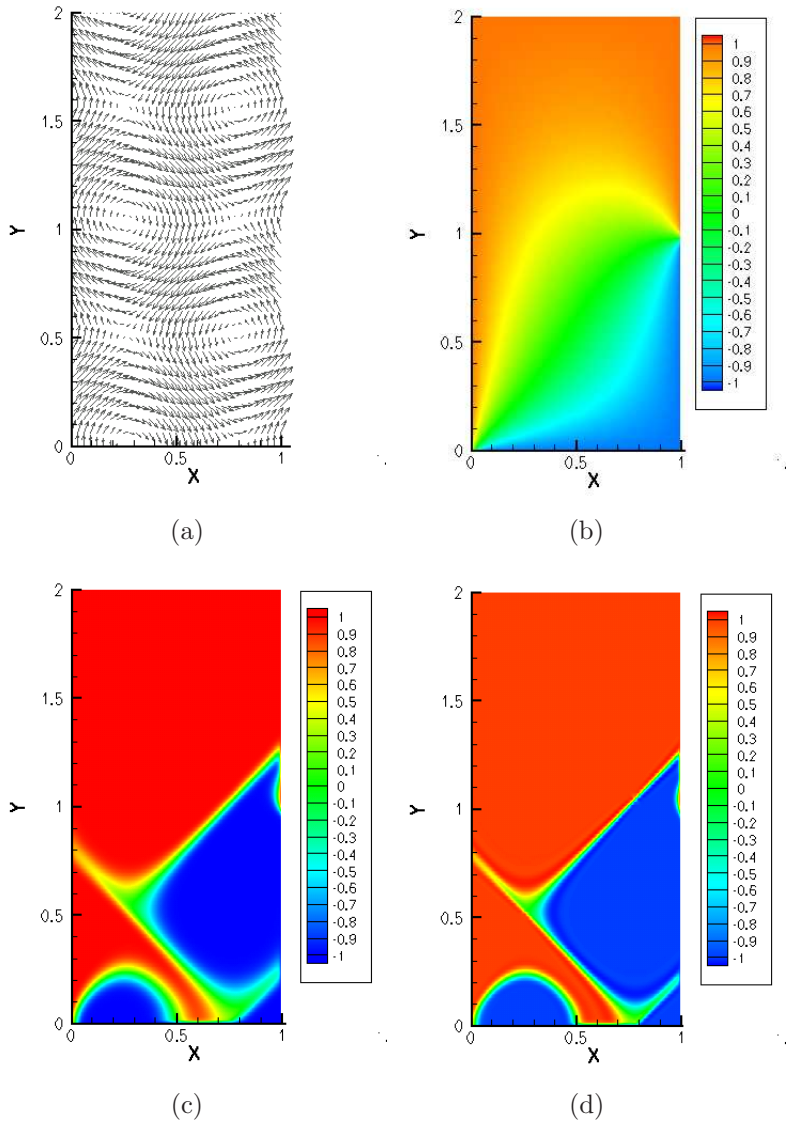


Figure 4.8: The convection-diffusion problem (Case study 4.2.4): no source term ($f = 0$); (a) Arrow plot of the wind (b) the solution for $Pe_* = 0$, (c) the solution for $Pe_* = 500$, (d) the solution for $Pe_* = 10,000$.

SUPG FEM discretisation on a uniform grid.

In Table 4.11 we present the iteration count and the execution times of Solver Strategy 4.2 for Case study 4.2.4. Comparing these results with the previous cases, we can observe consistency both in terms of absolute iteration counts and performance of different smoothers relative to each other.

For all cases in Table 4.11 we observed that the asymptotic performance of GMRES remains h-robust. However, there is a modest increase in iteration counts as a function of Pe_*. This increase is the smallest for the case of ILU_0 and $\text{tILU}_0(0.5, 0.25)$ smoothers. The shortest execution time is obtained when Gauss-Seidel smoother is

Table 4.11: The iteration counts and the (setup,total) execution time (in seconds) required for the convergence of the GMRES solver right-preconditioned by geometric multigrid V(2,2) cycle with several smoothers, when applied to the solution of the discrete convection-diffusion problem (Case study 4.2.4) obtained from $Q1$ SUPG FEM on uniform grids. Natural (tree-based) ordering of the unknowns, with each sub-table representing a different diffusion parameter.

(a) Pe_*=500

Smoother \ N	3969	16129	65025	261121	1046529
$ILU_0(0.5)$	9(0.79,1.01)	10(3.74,5.05)	9(14.7,20.2)	9(60.3,84.8)	8(264.8,371.5)
$tILU_0(0.5, 0.25)$	15(0.81,1.01)	15(3.77,4.72)	15(14.8,19.0)	15(60.6,81.7)	14(258.9,361.4)
$tILU_0(0.5, 0.5)$	23(0.81,1.07)	23(3.81,5.19)	23(14.8,20.8)	22(61.0,88.2)	21(251.4,374.6)
$Jacobi(0.5)$	35(0.81,1.19)	36(3.74,5.80)	36(14.7,25.7)	35(60.5,112.1)	32(245.3,452.2)
$GaussSeidel$	14(0.79,0.93)	14(3.76,4.43)	13(14.7,18.1)	12(60.3,75.1)	11(248.3,308.6)

(b) Pe_*=2,000

Smoother \ N	3969	16129	65025	261121	1046529
$ILU_0(0.5)$	14(0.79,1.12)	15(3.47,5.34)	16(14.7,24.1)	15(60.3,99.8)	14(258.8,423.6)
$tILU_0(0.5, 0.25)$	24(0.79,1.11)	26(3.52,5.26)	27(14.8,22.8)	26(60.8,99.4)	25(261.8,434.6)
$tILU_0(0.5, 0.5)$	39(0.79,1.26)	45(3.50,6.10)	47(14.7,28.1)	45(60.6,123.8)	42(255.9,566.4)
$Jacobi(0.5)$	60(0.80,1.57)	71(3.48,7.95)	73(14.8,39.4)	71(60.7,170.2)	69(259.7,851.6)
$GaussSeidel$	25(0.79,1.03)	27(3.53,4.79)	27(14.7,21.8)	25(60.3,91.9)	23(256.1,397.3)

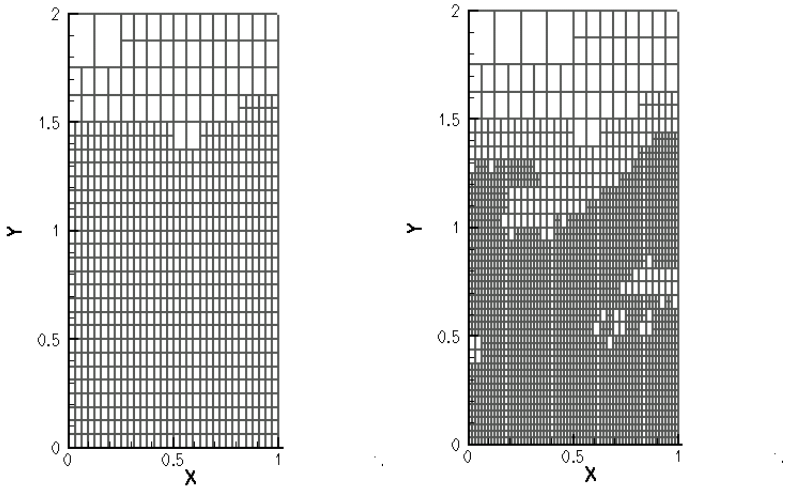
used, closely followed by the tILU$_0(0.5, 0.25)$ smoother.

Adaptively refined grids

Case study 4.2.4 is an example where discretisation by uniform or stretched grids would not be advantageous, due to the structure of the solution (there exists a steep layer in the interior of the domain, with the position depending upon the structure and strength of the wind). For this reason we use adaptive grid refinement as our discretisation strategy and test our solvers for this case.

The OOMPHLIB library [45] has a built in adaptive refinement function. Adaptive mesh refinement targets areas of interest by solving the problem at each level of refinement and using a posteriori error estimator function to calculate the solution error estimate elementwise. The elements in which the error estimate exceeds the maximum threshold are refined, and the elements in which the error estimate is smaller than the minimum threshold are unrefined as long as the desired min/max level is not achieved in all elements. The refinement procedure is stopped when the estimated solution error satisfies either a posteriori error in the interval $[10^{-3}, 10^{-5}]$ or a maximum of 6 levels of refinement.

In Figure 4.9 level 3 and 4 of the refinement process is presented for Case study 4.2.4. We can see that in parts of the domain where the solution does not change significantly (for $y > 1.5$) some of the elements are un-refined (coarsened), while parts of the domain with the most rapid changes in the solution have most of its elements refined.



(a) (b)

Figure 4.9: Adaptive mesh refinement for Case study 4.2.4 with (a) 3 levels of refinement, $N = 768$. (b) 4 levels of refinement, $N = 2411$.

The consequence of using adaptively refined grids is that the same level of accuracy in the discrete solution can be achieved with much fewer degrees of freedom than in the case of uniform refinement. However, spectral properties of coefficient matrices obtained from discretisation on adaptively refined grids are considerably different from the matrices obtained from uniform grids (in particular, the condition number depends on the size of the smallest elements in the grid). Also if the matrix stencils are h-dependent then the matrix coefficients will differ between rows (i.e. the coefficient matrix can potentially be badly scaled). These two facts can have potentially negative impact on the performance on the smoother and, as a consequence, the MG preconditioner. Table 4.12 summarises the convergence results of Solver Strategy 4.2, applied to Case study 4.2.4 discretised on adaptively refined grids. The main conclusion is that the results are consistent with those obtain on uniform grids (reported in Table 4.11) with the iteration counts generally lower. The best execution times are obtained from Gauss-Seidel and $\text{tILU}_0(0.5,0.25)$ smoothers.

In Table 4.13 we present the truncation statistics for the matrices obtained from the discretisation of Case study 4.2.4 on a sequence of adaptively refined grids. From Table 4.13 it follows that proportionately more entries are kept on the few finest levels than in the case of uniformly refined grids. Also, there is no substantial difference in the amount of truncated entries when Pe_* is increased.

Finally, for comparison reasons, in Table 4.14 we present the truncation statistics for coefficient matrices obtained by Q_1 SUPG FEM discretisation of Case studies 4.2.1, 4.2.2, 4.2.3 and 4.2.4 on adaptive grids for $Pe_* = 500$ and 2000. From this

Table 4.12: The iteration counts and the (setup,total) execution time (in seconds) required for the convergence of the GMRES solver right-preconditioned by geometric multigrid V(2,2) cycle with several smoothers, when applied to the solution of the discrete convection-diffusion problem (Case study 4.2.4) obtained from $Q1$ SUPG FEM on an adaptive grid refinement. Natural (tree-based) ordering of the unknowns, with each sub-table representing a different diffusion parameter.

(a) Pe_*=500

Smoother \ Level	2	3	4	5	6
$ILU_0(0.5)$	7(0.04,0.05)	8(0.15,0.18)	8(0.47,0.58)	8(1.23,1.52)	6(1.22,1.43)
$tILU_0(0.5, 0.25)$	10(0.04,0.05)	13(0.14,0.18)	13(0.47,0.56)	12(1.24,1.46)	10(1.22,1.41)
$tILU_0(0.5, 0.5)$	12(0.04,0.05)	18(0.15,0.19)	20(0.47,0.60)	17(1.24,1.51)	15(1.22,1.46)
$Jacobi(0.5)$	19(0.04,0.05)	28(0.14,0.20)	30(0.46,0.65)	26(1.25,1.66)	20(1.23,1.52)
$GaussSeidel$	9(0.04,0.05)	12(0.15,0.17)	11(0.46,0.52)	10(1.23,1.37)	8(1.21,1.31)

(b) Pe_*=2,000

Smoother \ Level	2	3	4	5	6
$ILU_0(0.5)$	8(0.04,0.05)	11(0.15,0.20)	12(0.50,0.67)	11(1.34,1.76)	12(1.21,1.61)
$tILU_0(0.5, 0.25)$	11(0.04,0.05)	17(0.15,0.19)	20(0.49,0.65)	19(1.34,1.74)	17(1.21,1.52)
$tILU_0(0.5, 0.5)$	13(0.04,0.05)	24(0.16,0.21)	31(0.49,0.70)	31(1.37,1.93)	26(1.23,1.64)
$Jacobi(0.5)$	22(0.04,0.06)	42(0.15,0.25)	49(0.49,0.82)	43(1.36,2.13)	36(1.22,1.74)
$GaussSeidel$	10(0.04,0.05)	17(0.15,0.18)	19(0.49,0.59)	16(1.37,1.59)	15(1.21,1.39)

table we can conclude that in the case of uniform wind (Case study 4.2.1) the amount of truncation is almost independent of Pe_* and is almost the same for both values values for α. For $\alpha = 0.5$ the percentage of truncated entries appears to be almost the same for all four problems and also appears almost independent of Pe_*. For the smaller value of $\alpha = 0.25$ complexity of the convection field and Pe_*, however, does have an impact to the amount of truncation. That is, more entries are being kept in cases of more complicated convection (Case studies 4.2.2 and 4.2.3) and when Pe_* is increased.

Table 4.13: Truncation statistics for the convection-diffusion operator obtained from Q1 SUPG FEM discretisation of Case study 4.2.4 as a function of Pe_* with adaptive grid refinement. $\widetilde{A}_l$ denotes the truncated matrix and A_l is the original matrix at the refinement level l. NNZ$(\cdot)$ denotes the number of non-zero entries and $\frac{NNZ(\widetilde{A}_l)}{NNZ(A_l)}[\%]$ the percentage of the non-zero entries kept in the truncated matrices.

(a) Pe_*=20

Level l	6	5	4	3	2	1	$\sum_l(\text{NNZ}(\widetilde{A}_l))$
N	3,994	1,077	308	87	28	9	
NNZ(A_l)	35,620	9,449	2,622	685	196	49	48,621
$\alpha = 0$							
NNZ($\widetilde{A}_l$)	35,620	9,449	2,622	685	196	49	48,621
$\frac{NNZ(\widetilde{A}_l)}{NNZ(A_l)}[\%]$	100	100	100	100	100	100	η =100
$\alpha = 0.25$							
NNZ($\widetilde{A}_l$)	11,688	2,960	754	198	70	22	15,692
$\frac{NNZ(\widetilde{A}_l)}{NNZ(A_l)}[\%]$	33	31	29	29	36	51	η =32
$\alpha = 0.5$							
NNZ($\widetilde{A}_l$)	4,018	1,111	343	124	41	12	5,649
$\frac{NNZ(\widetilde{A}_l)}{NNZ(A_l)}[\%]$	11	12	13	19	21	35	η =12
$\alpha = 1.0$							
NNZ($\widetilde{A}_l$)	3,994	1,077	308	87	28	9	5,503
$\frac{NNZ(\widetilde{A}_l)}{NNZ(A_l)}[\%]$	11	11	12	13	14	18	η =11

(b) Pe_*=500

Level l	6	5	4	3	2	1	$\sum_l(\text{NNZ}(\widetilde{A}_l))$
N	5,053	1,376	384	119	32	9	6,973
NNZ(A_l)	45,303	12,204	3,314	971	230	49	62,071
$\alpha = 0.25$							
NNZ($\widetilde{A}_l$)	13,951	4,098	1,211	385	103	22	19,770
$\frac{NNZ(\widetilde{A}_l)}{NNZ(A_l)}[\%]$	31	34	37	40	45	45	η =32
$\alpha = 0.5$							
NNZ($\widetilde{A}_l$)	8,675	2,426	711	223	67	12	12,114
$\frac{NNZ(\widetilde{A}_l)}{NNZ(A_l)}[\%]$	20	20	21	23	29	24	η =20

(c) Pe_*=2000

Level l	6	5	4	3	2	1	$\sum_l(\text{NNZ}(\widetilde{A}_l))$
N	4,937	1,352	376	112	32	9	6,818
NNZ(A_l)	44,269	11,995	3,242	908	230	49	60,692
$\alpha = 0.25$							
NNZ($\widetilde{A}_l$)	15,572	4,348	1,232	364	105	22	21,643
$\frac{NNZ(\widetilde{A}_l)}{NNZ(A_l)}[\%]$	35	36	38	40	46	45	η =36
$\alpha = 0.5$							
NNZ($\widetilde{A}_l$)	9,016	2,534	715	217	68	12	12,562
$\frac{NNZ(\widetilde{A}_l)}{NNZ(A_l)}[\%]$	20	21	22	24	30	24	η =21

Table 4.14: Comparison of truncation statistics for Case studies 4.2.1, 4.2.2, 4.2.3 and 4.2.4 as a function of Pe_* and the truncation parameter α. The discrete convection-diffusion operators are obtained from Q_1 SUPG FEM discretisation using adaptive meshing. The results present the total number of non-zero entries in the entire MG hierarchy and, in brackets, the percentage of the total number of non-zero entries in the entire MG hierarchy after truncation.

Pe_*	Case Study	$\alpha = 0$	$\alpha = 0.25$	$\alpha = 0.5$	$\alpha = 1.0$
500	4.2.1	13,640(100)	3,456(25)	3,260(24)	1,624(12)
	4.2.2	21,970(100)	7,159(33)	4,962(23)	2,560(12)
	4.2.3	21,927(100)	8,053(37)	4,604(21)	2,511(11)
	4.2.4	62,071(100)	19,770(32)	12,114(20)	6,973(11)
2000	4.2.1	13,339(100)	3,386(25)	3,194(24)	1,589(12)
	4.2.2	22,787(100)	8,468(37)	5,211(23)	2,655(12)
	4.2.3	23,188(100)	9,166(40)	4,945(21)	2,650(11)
	4.2.4	60,692(100)	21,643(36)	12,562(21)	6,818(11)

4.2.5 Summary

In this section we tested the performance of the GMRES solver preconditioned by GMG, where a variety of smoothers were used on a number of case studies. In all cases the tILU_0 smoother was asymptotically optimal with respect to the discrete problem size and exhibited only moderate dependence on Pe_*. In cases of complicated convection fields this dependence was no worse than when the standard ILU_0 smoother was used. When tested with multiple-directional nodal ordering, the tILU_0 smoother showed little sensitivity in its performance with respect to the ordering of the unknowns. This leads to the assumption that tILU_0 is an appropriate smoother for AMG with arbitrary nodal ordering.

In terms of the execution times, the preconditioned iterative solver with the $\text{tILU}_0(0.5,0.25)$ smoother is better than the solver obtained when using ILU_0 or damped Jacobi smoothing. Furthermore, the execution times for the tILU_0 smoother were comparable to the execution times obtained from the Gauss-Seidel smoother. Having said that, the tILU_0 smoother shows a much better Pe, h-robustness than the Gauss-Seidel smoother in important cases such as recirculating wind problems discretised by stretched grids.

Damping parameters

For completeness we will now look at alternative damping parameters γ that are used within the context of simple point smoothers. In [9] a damping parameter of $\gamma = 0.5$ is used for the damped Jacobi smoother, on the double-glazing problem with stretched grids. For the 1D Laplacian operator, the damped Jacobi smoother with $\gamma = \frac{2}{3}$ is the optimal damping parameter for a uniform square grid [8]; $\gamma = 0.8$ in the case of a 5-point 2D Laplacian operator, and $\gamma = \frac{6}{7}$ for 7-point approximation in 3D. Furthermore, [31, p.100] states that for Q1 approximation on a uniform square

grid the optimal damping parameter for Jacobi is $\gamma = \frac{8}{9}$. Since the preconditioner setup time does not depend on γ, we only report iteration count below. The total execution times in each case can be estimated by taking the total execution times in Tables 4.15(a) or 4.16(a) scaled with the number of iterations.

Table 4.15: The iteration counts and the (setup,total) execution time (in seconds) required for the convergence of the GMRES solver right-preconditioned by geometric multigrid V(2,2) cycle with several smoothers, when applied to the solution of the discrete convection-diffusion problem (Case study 4.2.2) obtained from $Q1$ SUPG FEM on uniform grids. Natural (tree-based) ordering of the unknowns and Pe_*=2000, with each sub-table representing a different damping parameter.

(a) $\gamma = 0.5$

Smoother \ N	3969	16129	65025	261121
$ILU_0(0.5)$	17(0.78,1.19)	17(3.44,5.32)	16(15.1,24.8)	14(60.3,97.6)
$tILU_0(0.5, 0.25)$	43(0.78,1.36)	41(3.45,6.08)	38(15.2,27.3)	35(60.7,114.8)
$tILU_0(0.5, 0.5)$	61(0.79,1.58)	65(3.46,7.47)	62(15.3,34.7)	58(60.5,150.6)
$Jacobi(0.5)$	189(0.77,4.39)	194(3.46,20.7)	180(15.3,99.6)	166(60.3,416.1)
$GaussSeidel$	39(0.79,1.16)	36(3.45,5.05)	35(15.2,25.1)	29(60.3,98.0)

(b) $\gamma = 0.666\dot{}$

Smoother \ N	3969	16129	65025	261121
$ILU_0(2/3)$	15	15	13	12
$tILU_0(2/3, 0.25)$	36	36	31	28
$tILU_0(2/3, 0.5)$	50	50	50	46
$Jacobi(2/3)$	161	161	151	137

(c) $\gamma = 0.8$

Smoother \ N	3969	16129	65025	261121
$ILU_0(0.8)$	13	13	12	11
$tILU_0(0.8, 0.25)$	32	30	27	25
$tILU_0(0.8, 0.5)$	44	47	44	40
$Jacobi(0.8)$	×	×	×	×

(d) $\gamma = 1.0$

Smoother \ N	3969	16129	65025	261121
$ILU_0(1.0)$	13	13	12	12
$tILU_0(1.0, 0.25)$	28	26	23	34
$tILU_0(1.0, 0.5)$	40	44	42	37
$Jacobi(1.0)$	×	×	×	×

Table 4.15 shows the dependence of the iteration counts with respect to damping parameters and different smoothers for Case study 4.2.2 with $Pe_* = 2000$. In the case of damped Jacobi smoothing we observe that the GMG preconditioned solver fails to converge for large values of the damping parameter. That is, convergence when $\gamma < 0.8$, however damped Jacobi is still not a competitive smoothing strategy. For ILU_0 smoothing, the smallest iteration counts are seen for $\gamma = 0.8$. It is therefore clear that the tILU_0 smoother, in the extreme cases, benefit from damping. However, for each case the smallest iteration counts are obtained for different values of damping parameters. In Table 4.16, Case study 4.2.4, we find that the smallest number of

Table 4.16: The iteration counts and the (setup,total) execution time (in seconds) required for the convergence of the GMRES solver right-preconditioned by geometric multigrid V(2,2) cycle with several smoothers, when applied to the solution of the discrete convection-diffusion problem (Case study 4.2.4) obtained from $Q1$ SUPG FEM on uniform grids. Natural (tree-based) ordering of the unknowns and Pe_*=2000, with each sub-table representing a different damping parameter.

(a) $\gamma = 0.5$

Smoother \ N	3969	16129	65025	261121
$ILU_0(0.5)$	14(0.79,1.12)	15(3.47,5.34)	16(14.7,24.1)	15(60.3,99.8)
$tILU_0(0.5, 0.25)$	24(0.79,1.11)	26(3.52,5.26)	27(14.8,22.8)	26(60.8,99.4)
$tILU_0(0.5, 0.5)$	39(0.79,1.26)	45(3.50,6.10)	47(14.7,28.1)	45(60.6,123.8)
$Jacobi(0.5)$	60(0.80,1.57)	71(3.48,7.95)	73(14.8,39.4)	71(60.7,170.2)
$GaussSeidel$	25(0.79,1.03)	27(3.53,4.79)	27(14.7,21.8)	25(60.3,91.9)

(b) $\gamma = 0.666\dot{}$

Smoother \ N	3969	16129	65025	261121
$ILU_0(2/3)$	12	13	13	12
$tILU_0(2/3, 0.25)$	20	22	23	22
$tILU_0(2/3, 0.5)$	34	39	40	38
$Jacobi(2/3)$	51	59	60	58

(c) $\gamma = 0.8$

Smoother \ N	3969	16129	65025	261121
$ILU_0(0.8)$	11	12	12	11
$tILU_0(0.8, 0.25)$	18	19	20	19
$tILU_0(0.8, 0.5)$	30	35	35	34
$Jacobi(0.8)$	×	×	×	×

(d) $\gamma = 1.0$

Smoother \ N	3969	16129	65025	261121
$ILU_0(1.0)$	9	10	12	10
$tILU_0(1.0, 0.25)$	16	17	18	×
$tILU_0(1.0, 0.5)$	28	39	157	×
$Jacobi(1.0)$	×	×	×	×

iterations for ILU_0 is when no damping is performed. This leads to the assumption that the damping parameter of ILU_0 is problem dependent. On the other hand when using a damped Jacobi smoother we find that for Case study 4.2.4 a damping parameter of $\gamma = \frac{2}{3}$ leads to the smallest iteration count. This is consistent with Case study 4.2.2.

Numerical evidence shows that $\alpha = 0.25$ is the most appropriate value for the truncation parameter when tILU_0 smoother is used for GMG preconditioning. For completeness, we now look at the relationship of $\text{tILU}_0(\gamma, 0.25)$ as a function of the damping parameter. In Tables 4.15 and 4.16 we find that the smallest iteration count and therefore the smallest solve time, and subsequently computational cost, is found when $\gamma = 0.8$. In Table 4.15 we observe a reduction of 29% in the number of iterations between the cases $\gamma = 0.5$ and $\gamma = 0.8$, and in Table 4.16 a reduction of 27%. This decrease in the iteration count makes tILU_0 a significant competitor to

Gauss-Seidel in terms of solve time. Also, it is worth noting that for $\gamma = 1$, the GMG preconditioner with tILU$_0$ smoother show some difficulties with convergence.

Number of smoothing sweeps

We now look at the impact that changing the number of smoothing sweeps, of the preconditioned GMG V-cycle, has on the solver's iteration counts. We would expect the iteration count to decrease as the number of pre and post smoothing sweeps within a V-cycle increase, as the residual equations (3.5) at each MG level are solved more accurately. However, a more important criterion is the overall solve time, and its relationship to the number of pre and post smoothing iterations.

The results for Case study 4.2.2 are summarised in Table 4.17. The increase in the

Table 4.17: The iteration counts and the (setup,total) execution time (in seconds) required for the convergence of the GMRES solver right-preconditioned by geometric multigrid with several smoothers, when applied to the solution of the discrete convection-diffusion problem (Case study 4.2.2) obtained from $Q1$ SUPG FEM on uniform grids. A constant Pe_*=2000 and natural (tree-based) ordering of the unknowns, with each sub-table representing a different number of pre and post smoothing operations.

(a) $V(1,1)$

Smoother \ N	3969	16129	65025	261121
$ILU_0(0.5)$	25(0.91,1.66)	26(3.67,6.92)	25(15.18,28.42)	23(60.11,112.1)
$tILU_0(0.5, 0.25)$	66(0.91,2.26)	65(3.72,8.79)	60(15.34,34.19)	56(61.91,137.9)
$tILU_0(0.5, 0.5)$	96(0.91,2.73)	107(3.69,11.96)	105(15.39,49.27)	98(61.28,203.0)
$Jacobi(0.5)$	×	×	×	×
$GaussSeidel$	62(0.91,1.77)	60(3.64,6.84)	66(15.08,29.98)	54(60.24,111.7)

(b) $V(2,2)$

Smoother \ N	3969	16129	65025	261121
$ILU_0(0.5)$	17(0.78,1.19)	17(3.44,5.32)	16(15.1,24.8)	14(60.28,97.56)
$tILU_0(0.5, 0.25)$	43(0.78,1.36)	41(3.45,6.08)	38(15.2,27.3)	35(60.66,114.8)
$tILU_0(0.5, 0.5)$	61(0.79,1.58)	65(3.46,7.47)	62(15.3,34.7)	58(60.52,150.6)
$Jacobi(0.5)$	189(0.77,4.39)	194(3.46,20.7)	180(15.3,99.6)	166(60.35,416.1)
$GaussSeidel$	39(0.79,1.16)	36(3.45,5.05)	35(15.2,25.1)	29(60.26,97.99)

(c) $V(4,4)$

Smoother \ N	3969	16129	65025	261121
$ILU_0(0.5)$	12(0.88,1.46)	11(3.67,5.82)	11(13.91,23.30)	10(60.40,96.62)
$tILU_0(0.5, 0.25)$	27(0.91,1.77)	26(3.72,6.86)	23(15.29,26.65)	21(61.83,105.6)
$tILU_0(0.5, 0.5)$	37(0.88,1.92)	38(3.74,7.85)	36(15.26,31.59)	33(61.49,126.2)
$Jacobi(0.5)$	116(0.91,5.10)	116(3.72,20.51)	107(15.16,78.83)	98(61.52,317.1)
$GaussSeidel$	26(0.91,1.56)	24(3.67,6.11)	22(15.08,24.44)	20(60.42,97.01)

number of smoothing sweeps leads to an overall reduction in execution time for tILU$_0$ and damped Jacobi. This is not the case when ILU$_0$ and Gauss-Seidel smoothers are used (in these two cases increasing the number of smoothing sweeps beyond two does not lead to any significant improvement in the overall execution time).

Table 4.18 contains results for Case study 4.2.4. We see that roughly the same pattern of behaviour as in Table 4.17. When $\text{tILU}_0(0.5, 0.25)$ smoothing is used, we

Table 4.18: The iteration counts and the (setup,total) execution time (in seconds) required for the convergence of the GMRES solver right-preconditioned by geometric multigrid with several smoothers, when applied to the solution of the discrete convection-diffusion problem (Case study 4.2.4) obtained from $Q1$ SUPG FEM on uniform grids. A constant Pe_*=2000 and natural (tree-based) ordering of the unknowns, with each sub-table representing a different number of pre and post smoothing operations.

(a) $V(1,1)$

Smoother \ N	3969	16129	65025	261121
$ILU_0(0.5)$	21(0.35,0.59)	23(1.45,2.55)	25(6,11.1)	24(23.89,44.58)
$tILU_0(0.5, 0.25)$	33(0.35,0.59)	38(1.45, 2.53)	41(6.01,10.8)	41(24.45,44.86)
$tILU_0(0.5, 0.5)$	56(0.36,0.72)	69(1.45,3.29)	75(6.04,14.62)	74(24.12,62.11)
$Jacobi(0.5)$	96(0.36,0.99)	118(1.45,4.71)	124(5.98,20.47)	124(24.28,91.51)
$GaussSeidel$	36(0.35,0.52)	40(1.44,2.2)	41(5.64,8.62)	39(23.76,36.77)

(b) V(2,2)

Smoother \ N	3969	16129	65025	261121
$ILU_0(0.5)$	14(0.35,0.54)	15(1.44,2.3)	16(6.1,9.99)	15(23.99,39.53)
$tILU_0(0.5, 0.25)$	24(0.36,0.57)	26(1.46,2.35)	27(6.03,9.78)	26(24.46,39.65)
$tILU_0(0.5, 0.5)$	39(0.35,0.65)	45(1.46,2.83)	47(6.04,12.11)	45(24.21,49.61)
$Jacobi(0.5)$	60(0.35,0.84)	71(1.46,3.83)	73(5.98,16.13)	71(24.27,68.05)
$GaussSeidel$	25(0.36,0.52)	27(1.43,2.11)	27(6.12,8.9)	25(23.71,34.76)

(c) $V(4,4)$

Smoother \ N	3969	16129	65025	261121
$ILU_0(0.5)$	10(0.35,0.53)	10(1.45,2.21)	10(6.01,9.27)	9(23.87,36.41)
$tILU_0(0.5, 0.25)$	16(0.35,0.55)	17(1.44, 2.25)	17(6.03,9.26)	16(24.4,37.14)
$tILU_0(0.5, 0.5)$	26(0.35,0.62)	29(1.45, 2.64)	30(6.01,11.16)	28(24.22,44.9)
$Jacobi(0.5)$	40(0.35,0.85)	45(1.44,3.71)	45(5.98,15.33)	44(24.21,63.86)
$GaussSeidel$	17(0.35,0.52)	17(1.44,2.1)	17(5.95,8.71)	16(23.75, 34.89)

see that an increase in the number of pre and post smoothing sweeps from 1 to 2 leads to a reduction in iteration counts by $\sim 36\%$ and a reduction in the total execution time by $\sim 12\%$. On the other hand, a further increase from 2 to 4 smoothing sweeps reduces the iteration counts by a further $\sim 25\%$, but the overall execution time is reduced by only $\sim 5\%$.

From Tables 4.17 and 4.18 we can conclude that V(2,2) is an appropriate choice for our V-cycle.

4.3 Algebraic multigrid preconditioning of the convection-diffusion problem

Geometric multigrid is based on the existence of a nested hierarchy of grids and the discretisation of the underlying problem on these grids (which assumes the knowledge of the geometric information associated with these grids). In many practical cases the domain of interest is fairly complicated. Such cases are usually discretised by non-structured grids, which makes the construction of a nested grid hierarchy difficult or even impossible. In such cases the application of MG depends on the ability to construct a sequence of coarse representations of the discrete operator. This can be achieved through an automatic coarsening procedure, based entirely on the matrix properties. This is the concept of AMG.

In this section we examine the effectiveness of the novel smoothing methodology, based on truncated ILU (tILU_0) factorisation in the AMG context, when applied to discrete convection-dominated problems. We use classical Ruge-Stüben AMG, where the coarsening procedure is based on matrix properties of the discrete Poisson operator [61, Chapter 4]. However, this coarsening procedure is also effective when applied to discrete convection-diffusion problems. For the case of GMG we were able to explore different ordering strategies of the unknowns, to get potentially a more robust smoothing strategy (Section 4.2.2). For the case of AMG there is no geometric information available, to help with ordering the unknowns. Therefore in the case of AMG we can only use default or "black-box" ordering. This is why having a smoothing strategy that shows little difference in its performance with respect to the ordering of the unknowns, as was the case for tILU_0, is important.

To solve the linear system (3.1) using AMG preconditioning, we use Solver Strategy 4.3.

Solver Strategy 4.3

Krylov Method: Hypre GMRES (tolerance 10^{-6})
Preconditioner: Hypre BoomerAMG
  Coarsening:
    Classical Ruge-Stüben
    Strength of dependance: $\theta = 0.25$
  Cycle: $V(2,2)$
Smoothers:
  $Gauss - Seidel$
  $ILU_0(0.5)$
  $tILU_0(0.5, \alpha)$
  $Jacobi(0.5)$

In Table 4.19 we show the convergence characteristics of Solver Strategy 4.3 applied to discrete problems from Case study 4.2.1. In the case of AMG we extend the

Table 4.19: The iteration counts and the (setup,total) execution time (in seconds) required for the convergence of the GMRES solver right-preconditioned by classical algebraic multigrid V(2,2) cycle with several smoothers, when applied to the solution of the discrete convection-diffusion problem (Case study 4.2.1) obtained from $Q1$ SUPG FEM on uniform grids. Each sub-table represents a different diffusion parameter. $\times$ denotes a lack of convergence within 100 GMRES iterations.

(a) Pe_*=500

Smoother \ N	3969	16129	65025	261121	1046529
$ILU_0(0.5)$	5(0.11,0.19)	5(0.49,1.23)	5(2.60,8.10)	5(14.3,47.0)	5(68.3,173.5)
$tILU_0(0.5, 0.25)$	7(0.09,0.16)	7(0.40,0.99)	8(1.87,6.86)	10(8.99,45.3)	8(29.8,126.9)
$tILU_0(0.5, 0.5)$	7(0.09,0.16)	8(0.40,1.01)	10(1.87,7.98)	10(8.99,43.4)	9(33.8,145.3)
$Jacobi(0.5)$	9(0.03,0.12)	7(0.14,0.62)	8(0.77,6.23)	9(6.76,42.2)	9(12.1,132.5)
$GaussSeidel$	8(0.03,0.10)	5(0.14,0.50)	6(0.77,4.84)	6(4.31,27.5)	5(11.8,78.7)

(b) Pe_*=2,000

Smoother \ N	3969	16129	65025	261121	1046529
$ILU_0(0.5)$	5(0.11,0.19)	6(0.48,1.30)	5(2.23,7.14)	8(7.97,30.7)	5(59.8,204.9)
$tILU_0(0.5, 0.25)$	8(0.09,0.18)	7(0.40,1.01)	12(1.82,9.07)	7(9.38,30.1)	8(38.7,174.4)
$tILU_0(0.5, 0.5)$	8(0.09,0.17)	8(0.39,1.01)	8(1.81,6.62)	8(7.91,30.1)	10(38.4,198.8)
$Jacobi(0.5)$	20(0.03,0.22)	13(0.13,1.00)	9(0.74,6.55)	7(3.70,27.0)	8(18.8,153.4)
$GaussSeidel$	12(0.03,0.19)	12(0.14,0.96)	8(0.72,5.57)	5(3.67,20.1)	6(28.6,124.9)

(c) Pe_*=10,000

Smoother \ N	3969	16129	65025	261121	1046529
$ILU_0(0.5)$	4(0.10,0.18)	5(0.47,1.19)	6(2.18,8.36)	6(10.0,37.8)	6(45.7,179.8)
$tILU_0(0.5, 0.25)$	14(0.09,0.23)	13(0.39,1.45)	12(1.78,9.35)	9(8.07,36.5)	$\times$
$tILU_0(0.5, 0.5)$	13(0.08,0.21)	13(0.38,1.36)	12(1.74,8.84)	10(7.95,37.4)	8(33.9,134.5)
$Jacobi(0.5)$	33(0.03,0.36)	40(0.13,3.02)	26(0.67,17.0)	15(3.56,49.1)	9(16.6,145.7)
$GaussSeidel$	19(0.02,0.20)	25(0.13,1.85)	22(0.67,13.8)	23(3.56,68.9)	14(17.4,192.9)

range of Pe_* for which we test the solver to $Pe_* = 10,000$ as the AMG preconditioner generally performs more robustly for highly convective cases. From Table 4.19 we can see that the AMG preconditioner has asymptotically optimal scaling with respect to the problem size for all smoothers and is Pe-robust for tILU$_0$ smoothers for all values of truncation parameter $\alpha \in [0, 1]$[1]. However, when using Gauss-Seidel as a smoother, there is a strong dependence on the iteration counts with respect to Pe_*. For the highly convective case, $Pe_* = 10,000$, the shortest execution time is recorded for the tILU$_0$(0.5,0.5) smoother.

In GMG all discrete operators at coarse levels are usually created directly by FE discretisation of the governing equations on these grids. This has an influence on the sparsity pattern of the coarse level operators. For example, in the case of a hierarchy of uniformly refined tensor product quadrilateral grids, Q_1 FE approximation, we can expect at all GMG levels that the matrices will retain a 9-point stencil, that is, each

[1]We observe a lack of convergence when using tILU$_0$(0.5,0.25) smoother for the case $N = 1046529$ and $Pe_* = 10,000$.

row will contain 9 non-zero entries. In the case of classical AMG coarsening this is frequently not the case. Regular coarsening patterns can only be expected for the Poisson problem discretised on regular grids. When non-structured grids are used and/or singular perturbations of the second-order elliptic operators are coarsened by classical AMG, one can expect that the generated coarse-level representations of the original matrix will have considerably different characteristics than the original matrix itself. An essential property of classical AMG coarsening is to generate coarse level matrices which are much denser than the original coefficient matrix. By inspection it can be seen that a substantial amount of the off-diagonal entries in coarse level matrices are very small in magnitude, and thus do not have a substantial contribution to the matrix properties. This makes them suitable candidates for truncation by (3.43). This situation is even more pronounced in 3D [8]. The difference in the matrix properties of the coarse levels in GMG and AMG has led to a switch in the optimal value for the truncation parameter from $\alpha = 0.25$ (the best value for the GMG case) to $\alpha = 0.5$ for classical AMG.

Table 4.20 shows the efficiency of Solver Strategy 4.3 for Case study 4.2.2. For

Table 4.20: The iteration counts and the (setup,total) execution time (in seconds) required for the convergence of the GMRES solver right-preconditioned by classical algebraic multigrid V(2,2) cycle with several smoothers, when applied to the solution of the discrete convection-diffusion problem (Case study 4.2.2) obtained from $Q1$ SUPG FEM on uniform grids. Each sub-table represents a different diffusion parameter. $\times$ denotes a lack of convergence within 100 GMRES iterations.

(a) Pe_*=500

Smoother \ N	3969	16129	65025	261121	1046529
$ILU_0(0.5)$	7(0.16,0.34)	6(0.58,1.62)	6(2.63,10.1)	6(12.1,47.2)	7(39.0,164.8)
$tILU_0(0.5, 0.25)$	8(0.12,0.24)	7(0.47,1.21)	9(2.05,8.54)	10(9.22,43.6)	10(32.7,155.8)
$tILU_0(0.5, 0.5)$	10(0.13,0.27)	9(0.46,1.32)	10(2.01,8.78)	12(9.03,48.5)	11(31.8,156.4)
$Jacobi(0.5)$	12(0.03,0.19)	10(0.13,0.98)	10(0.70,7.84)	11(3.67,44.5)	11(12.3,154.3)
$GaussSeidel$	10(0.03,0.16)	8(0.13,0.82)	7(0.73,6.17)	7(3.67,29.3)	8(12.3,106.5)

(b) Pe_*=2000

Smoother \ N	3969	16129	65025	261121	1046529
$ILU_0(0.5)$	11(0.14,0.41)	10(0.58,2.17)	9(2.44,12.0)	7(10.8,46.9)	7(46.1,191.2)
$tILU_0(0.5, 0.25)$	14(0.12,0.33)	12(0.47,1.68)	11(1.93,9.24)	11(8.30,43.1)	11(35.3,177.7)
$tILU_0(0.5, 0.5)$	16(0.11,0.33)	14(0.46,1.74)	13(1.90,10.0)	11(8.19,41.9)	12(34.9,185.1)
$Jacobi(0.5)$	24(0.028,0.35)	20(0.12,1.80)	16(0.61,11.2)	12(3.06,43.8)	11(14.2,169.2)
$GaussSeidel$	17(0.03,0.24)	16(0.12,1.40)	14(0.63,9.82)	10(3.05,35.0)	9(13.5,128.6)

(c) Pe_*=10,000

Smoother \ N	3969	16129	65025	261121	1046529
$ILU_0(0.5)$	18(0.14,0.55)	21(0.57,3.88)	33(2.46,36.3)	$\times$	$\times$
$tILU_0(0.5, 0.25)$	27(0.12,0.52)	33(0.48,4.15)	31(1.97,23.08)	29(8.34,102.6)	22(33.3,323.2)
$tILU_0(0.5, 0.5)$	30(0.11,0.54)	36(0.46,3.94)	30(1.93,21.3)	28(8.29,100.1)	19(33.3,268.2)
$Jacobi(0.5)$	48(0.03,0.72)	55(0.12,5.24)	47(0.62,32.8)	36(2.87,128.5)	25(12.6,337.9)
$GaussSeidel$	36(0.03,0.47)	51(0.12,4.55)	$\times$	$\times$	$\times$

$Pe_* = 500$ and 2000 the asymptotic performance of GMRES, for all smoothers, is

Pe, h-robust. For moderate values of Pe_* the shortest execution times are obtained when using Gauss-Seidel smoothing, but for highly-convective flow ($Pe_* = 10,000$) tILU$_0$(0.5,0.5) smoother has asymptotically the smallest iteration count and execution time with h-robust performance. A similar performance was also seen for Case study 4.2.3 reported in Appendix A. A noticeable feature in Table 4.20 when $Pe_* = 10,000$ is that tILU$_0$, $\alpha \in (0, 1]$, smoother is h-robust even when standard ILU$_0$ and Gauss-Seidel smoothers fail.

In Table 4.21 we present the iteration counts of Solver Strategy 4.3 applied to discrete problems from Case study 4.2.4. It can be seen that for moderate values of

Table 4.21: The iteration counts and the (setup,total) execution time (in seconds) required for the convergence of the GMRES solver right-preconditioned by classical algebraic multigrid V(2,2) cycle with several smoothers, when applied to the solution of the discrete convection-diffusion problem (Case study 4.2.4) obtained from $Q1$ SUPG FEM on uniform grids. Each sub-table represents a different diffusion parameter. $\times$ denotes a lack of convergence within 100 GMRES iterations.

(a) Pe_*=500

Smoother \ N	3969	16129	65025	261121	1046529
$ILU_0(0.5)$	6(0.12,0.25)	6(0.54,1.57)	6(2.35,8.67)	6(11.4,44.2)	6(44.2,170.5)
$tILU_0(0.5, 0.25)$	7(0.10,0.20)	8(0.44,1.21)	8(1.87,7.11)	10(8.62,41.4)	9(33.8,148.7)
$tILU_0(0.5, 0.5)$	9(0.10,0.21)	9(0.44,1.25)	10(1.83,7.95)	11(8.42,42.8)	10(34.0,152.5)
$Jacobi(0.5)$	9(0.03,0.14)	9(0.13,0.86)	9(0.66,6.80)	10(3.68,39.6)	9(15.0,144.5)
$GaussSeidel$	7(0.03,0.12)	7(0.13,0.70)	7(0.65,5.49)	7(3.66,28.6)	6(15.3,102.5)

(b) Pe_*=2,000

Smoother \ N	3969	16129	65025	261121	1046529
$ILU_0(0.5)$	8(0.13,0.31)	8(0.55,1.88)	8(2.42,10.9)	8(10.3,50.1)	8(39.2,187.5)
$tILU_0(0.5, 0.25)$	10(0.11,0.24)	9(0.46,1.41)	9(1.95,8.16)	9(8.16,37.3)	9(32.0,150.7)
$tILU_0(0.5, 0.5)$	12(0.11,0.26)	11(0.44,1.39)	12(1.91,9.58)	11(7.99,41.1)	9(31.9,148.9)
$Jacobi(0.5)$	12(0.03,0.18)	12(0.13,1.19)	12(0.64,8.85)	11(3.06,40.4)	10(12.6,142.6)
$GaussSeidel$	9(0.03,0.13)	9(0.12,0.88)	9(0.64,6.62)	8(3.19,31.2)	8(13.2,120.4)

(c) Pe_*=10,000

Smoother \ N	3969	16129	65025	261121	1046529
$ILU_0(0.5)$	12(0.13,0.37)	13(0.57,2.72)	15(2.47,18.5)	50(10.6,265.)	$\times$
$tILU_0(0.5, 0.25)$	15(0.11,0.31)	15(0.47,2.02)	14(2.00,12.2)	14(8.36,54.3)	12(33.0,190.4)
$tILU_0(0.5, 0.5)$	15(0.11,0.30)	17(0.46,2.23)	16(1.98,12.9)	16(8.16,57.0)	15(32.3,211.2)
$Jacobi(0.5)$	18(0.03,0.25)	17(0.12,1.66)	16(0.65,11.7)	16(3.05,57.2)	15(12.4,208.9)
$GaussSeidel$	14(0.03,0.19)	15(0.13,1.40)	14(0.64,10.1)	14(3.04,48.5)	12(12.4,161.3)

Pe_* all smoothers lead to an AMG preconditioner that is Pe, h-robust. The shortest execution time is observed when the Gauss-Seidel smoother is used. However, when Pe_* is increased further, the ILU$_0$ smoother fails for the largest problem, while other smoothers remain robust with moderate increase in iteration counts.

The AMG complexity measures C_G, C_A and C_S are now reported for the largest problem size ($N = 1,046,529$) obtained by discretising Case study 4.2.4. The results from Table 4.22 show that the grid complexity does not change considerably when Pe_* is increased, suggesting a fairly stable coarsening in this case. For the case of

Table 4.22: Coarsening statistics for Case study 4.2.4 with $N = 1,046,529$ as a function of Pe_*.

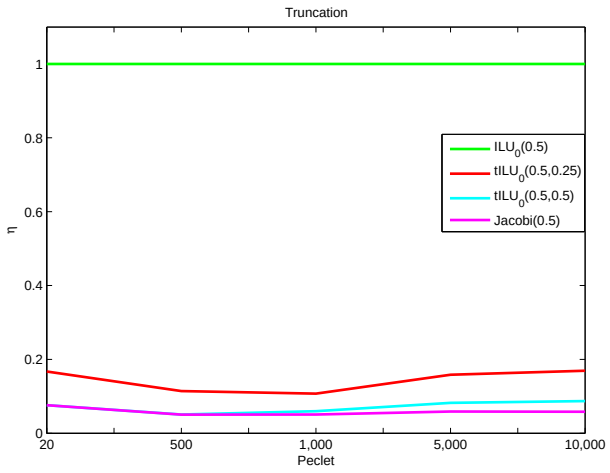
Pe_*	20	500	1,000	5,000	10,000
L	13	20	20	20	20
C_G	1.66553	2.12602	2.21301	2.30866	2.35954
C_A	2.43924	4.68124	4.85015	4.37539	4.49954
C_S	16.9796	30.8793	36.8793	33.4037	34.4232
$C_S^{(1)}$	8.98827	8.98827	8.98827	8.98827	8.98827

L is the number of MG levels, C_G is the grid complexity, C_A is the operator complexity, C_S is the average matrix stencil over all MG levels and $C_S^{(1)}$ is the average matrix stencil of A at the finest level.

uniform refinement with full coarsening GMG and direct discretisation of the coarse-level matrices, $C_G = C_A = \frac{4}{3}$ is expected [10, p.154]. It is clear from Table 4.22 that AMG has a considerably larger grid complexity than GMG.

In Table 4.22 we can also see that the operator complexity C_A depends strongly on Pe_* for $Pe_* < 500$, and after that it remains relatively constant. The values of $C_A \approx 4.6$ indicate that the coarse level matrices are considerably denser than the fine level matrix. This is clearly visible from the average stencil size C_S, for $Pe_* > 500$ (which is $C_S \approx 34$), compared to the average stencil size of A at the finest level where $C_S^{(1)} = 9$. The average stencil for GMG is consistent throughout all MG levels $C_S \approx 9$.

In Figure 4.10 we present the truncation ratio (3.40) for the matrix of dimension $N = 1,046,529$ obtained from Case study 4.2.4 as a function of Pe_*. We can see that



(a)

Figure 4.10: Truncation ratio (3.40) for Case study 4.2.4 with $N = 1,046,529$ as a function of Pe_*.

both tILU$_0$(0.5,0.25) and tILU$_0$(0.5,0.5) have only marginally more non-zero entries

than the damped Jacobi smoother. Also compared to the convergence results from Table 4.21, the performance is also remarkably similar.

Damping parameters

The tILU$_0$ smoother has two parameters, the truncation ratio (α) and damping (γ). Currently all tests using AMG preconditioning have had $\gamma = 0.5$ and we have tested different values of α on a number of problems and discretisations. We now examine the impact of the damping parameter on AMG preconditioning by looking at its performance. For this purpose we consider the following values of $\gamma = 0.5$, $\frac{2}{3}$, 0.8 and 1.0.

In Table 4.23 we summarise the convergence results for Case study 4.2.2 with $Pe_* = 10000$. From these results it is clear that ILU$_0$ gives poor performances regardless of the damping parameter chosen. In the cases of damped Jacobi and

Table 4.23: The iteration counts and the (setup,total) execution time (in seconds) required for the convergence of the GMRES solver right-preconditioned by algebraic multigrid V(2,2) cycle with several smoothers, when applied to the solution of the discrete convection-diffusion problem (Case study 4.2.2) obtained from $Q1$ SUPG FEM on uniform grids. Natural (tree-based) ordering of the unknowns and Pe_*=10000, with each sub-table representing a different damping parameter.

(a) $\gamma = 0.5$

Smoother \ N	3969	16129	65025	261121
$ILU_0(0.5)$	18(0.14,0.55)	21(0.57,3.88)	33(2.46,36.3)	×
$tILU_0(0.5, 0.25)$	27(0.12,0.52)	33(0.48,4.15)	31(1.97,23.1)	29(8.34,102.6)
$tILU_0(0.5, 0.5)$	30(0.11,0.54)	36(0.46,3.94)	30(1.93,21.3)	28(8.29,100.1)
$Jacobi(0.5)$	48(0.03,0.72)	55(0.12,5.24)	47(0.62,32.8)	36(2.87,128.5)
$GaussSeidel$	36(0.03,0.47)	51(0.12,4.55)	×	×

(b) $\gamma = 0.666\dot{}$

Smoother \ N	3969	16129	65025	261121
$ILU_0(0.5)$	17	20	43	X
$tILU_0(0.5, 0.25)$	25	30	28	32
$tILU_0(0.5, 0.5)$	27	32	29	25
$Jacobi(0.5)$	41	48	41	33

(c) $\gamma = 0.8$

Smoother \ N	3969	16129	65025	261121
$ILU_0(0.5)$	16	21	84	X
$tILU_0(0.5, 0.25)$	24	28	29	42
$tILU_0(0.5, 0.5)$	27	31	28	31
$Jacobi(0.5)$	38	44	43	88

(d) $\gamma = 1.0$

Smoother \ N	3969	16129	65025	261121
$ILU_0(0.5)$	16	35	X	X
$tILU_0(0.5, 0.25)$	23	27	35	62
$tILU_0(0.5, 0.5)$	34	42	80	X
$Jacobi(0.5)$	X	X	X	X

tILU$_0(\gamma, 0.5)$ the iteration counts improve marginally when $\gamma = \frac{2}{3}$ is used, compared to the default parameter ($\gamma = 0.5$). However any further increase results in a significant increase in the iteration count. The reduction in iteration count leads to a proportional reduction in the overall execution time (the preconditioner setup cost remains unchanged). Thus, for Case study 4.2.2 the use of tILU$_0(\frac{2}{3}, 0.5)$ smoother within V(2,2) AMG preconditioner leads to the shortest solution time.

In Table 4.24 we report the same experiment for Case study 4.2.4. We see

Table 4.24: The iteration counts and the (setup,total) execution time (in seconds) required for the convergence of the GMRES solver right-preconditioned by algebraic multigrid V(2,2) cycle with several smoothers, when applied to the solution of the discrete convection-diffusion problem (Case study 4.2.4) obtained from $Q1$ SUPG FEM on uniform grids. Natural (tree-based) ordering of the unknowns and Pe_*=10000, with each sub-table representing a different damping parameter.

(a) $\gamma = 0.5$

Smoother \ N	3969	16129	65025	261121
$ILU_0(0.5)$	12(0.13,0.37)	13(0.57,2.72)	15(2.47,18.5)	50(10.6,265.)
$tILU_0(0.5, 0.25)$	15(0.11,0.31)	15(0.47,2.02)	14(2.00,12.2)	14(8.36,54.3)
$tILU_0(0.5, 0.5)$	15(0.11,0.30)	17(0.46,2.23)	16(1.98,12.9)	16(8.16,57.0)
$Jacobi(0.5)$	18(0.03,0.25)	17(0.12,1.66)	16(0.65,11.7)	16(3.05,57.2)
$GaussSeidel$	14(0.03,0.19)	15(0.13,1.40)	14(0.64,10.1)	14(3.04,48.5)

(b) $\gamma = 0.666\dot{}$

Smoother \ N	3969	16129	65025	261121
$ILU_0(0.5)$	11	11	15	53
$tILU_0(0.5, 0.25)$	13	13	13	14
$tILU_0(0.5, 0.5)$	15	15	14	14
$Jacobi(0.5)$	16	15	14	14

(c) $\gamma = 0.8$

Smoother \ N	3969	16129	65025	261121
$ILU_0(0.5)$	10	12	16	70
$tILU_0(0.5, 0.25)$	12	13	13	15
$tILU_0(0.5, 0.5)$	14	14	14	13
$Jacobi(0.5)$	16	14	14	13

(d) $\gamma = 1.0$

Smoother \ N	3969	16129	65025	261121
$ILU_0(0.5)$	9	12	20	X
$tILU_0(0.5, 0.25)$	12	13	14	29
$tILU_0(0.5, 0.5)$	16	18	19	51
$Jacobi(0.5)$	16	17	18	91

again that ILU$_0$ smoother leads to an AMG preconditoner which performs poorly for large problem sizes, regardless of the value of damping parameter. By contrast, the tILU$_0(\gamma, 0.5)$ solving strategy performs consistently for a wide range of damping parameters $\gamma \in (0.5, 0.8)$.

In both cases tILU$_0$ can be considered competitive or better smoother than Gauss-Seidel.

Number of smoothing sweeps

Here we examine how the number of pre and post smoothing sweeps affects the convergence of AMG preconditioned GMRES solver. In all cases the damping parameter $\gamma = 0.5$ is used. In Table 4.25 we summarise the results for Case study 4.2.2 with $Pe_* = 10000$. For all smoothers, the iteration count decreases as the number of

Table 4.25: The iteration counts and the (setup,total) execution time (in seconds) required for the convergence of the GMRES solver right-preconditioned by algebraic multigrid with several smoothers, when applied to the solution of the discrete convection-diffusion problem (Case study 4.2.2) obtained from $Q1$ SUPG FEM on uniform grids. A constant Pe_*=10000 and natural (tree-based) ordering of the unknowns, with each sub-table representing a different number of pre and post smoothing operations.

(a) $V(1,1)$

Smoother \ N	3969	16129	65025	261121
$ILU_0(0.5)$	21(0.14,0.41)	30(0.57,3.32)	86(2.46,62.06)	×
$tILU_0(0.5, 0.25)$	35(0.12,0.48)	43(0.48,3.49)	41(1.97,22.11)	38(8.34,95.90)
$tILU_0(0.5, 0.5)$	40(0.11,0.50)	49(0.46,3.70)	46(1.93,23.35)	43(8.29,102.1)
$Jacobi(0.5)$	74(0.03,0.80)	83(0.12,6.20)	67(0.62,36.02)	57(2.87,146.3)
$GaussSeidel$	50(0.03,0.50)	69(0.12,4.98)	93(0.60,49.76)	×

(b) $V(2,2)$

Smoother \ N	3969	16129	65025	261121
$ILU_0(0.5)$	18(0.14,0.55)	21(0.57,3.88)	33(2.46,36.3)	×
$tILU_0(0.5, 0.25)$	27(0.12,0.52)	33(0.48,4.15)	31(1.97,23.1)	29(8.34,102.6)
$tILU_0(0.5, 0.5)$	30(0.11,0.54)	36(0.46,3.94)	30(1.93,21.3)	28(8.29,100.1)
$Jacobi(0.5)$	48(0.03,0.72)	55(0.12,5.24)	47(0.62,32.8)	36(2.87,128.5)
$GaussSeidel$	36(0.03,0.47)	51(0.12,4.55)	×	×

(c) $V(4,4)$

Smoother \ N	3969	16129	65025	261121
$ILU_0(0.5)$	15(0.14,0.68)	17(0.57,4.89)	21(2.46,42.67)	41(10.53,363.2)
$tILU_0(0.5, 0.25)$	22(0.12,0.66)	26(0.48,4.93)	25(1.97,33.98)	23(8.34,139.17)
$tILU_0(0.5, 0.5)$	24(0.11,0.65)	28(0.46,4.69)	26(1.93,31.96)	21(8.29,118.80)
$Jacobi(0.5)$	35(0.03,0.83)	42(0.12,6.83)	34(0.62,45.78)	29(2.87,171.19)
$GaussSeidel$	29(0.03,0.65)	42(0.12,6.60)	×	×

smoothing sweeps increases. This trend was also seen in GMG (Tables 4.17 and 4.18). However, the rate at which the iteration count decreases is not directly proportional to the to the increase in the number of sweeps; for $\text{tILU}_0(0.5, 0.5)$ an increase from 1 to 2 sweeps leads to a $\sim 35\%$ reduction in the iteration count. By increasing the number of sweeps to 4 this leads to a further reduction by only $\sim 25\%$. The increase in the number of sweeps results in an increase in the computational cost (execution time) of performing the preconditioning. Also, in cases when $\text{tILU}_0(0.5, 0.25)$ and damped Jacobi smoothers are used this decrease in iteration count is not reflected in the reduction of the overall solve time. For $\text{tILU}_0(0.5, 0.5)$ smoother the total execution time improves marginally when V(2,2) is used instead of V(1,1).

In Table 4.26 the results of the same experiment for Case study 4.2.4 are reported. From this table we see roughly the same pattern of behaviour as in Table 4.25. The

Table 4.26: The iteration counts and the (setup,total) execution time (in seconds) required for the convergence of the GMRES solver right-preconditioned by algebraic multigrid with several smoothers, when applied to the solution of the discrete convection-diffusion problem (Case study 4.2.4) obtained from $Q1$ SUPG FEM on uniform grids. A constant Pe_*=10000 and natural (tree-based) ordering of the unknowns, with each sub-table representing a different number of pre and post smoothing operations.

(a) $V(1,1)$

Smoother \ N	3969	16129	65025	261121
$ILU_0(0.5)$	15(0.13,0.31)	16(0.57,1.99)	19(2.47,14.05)	37(10.60,121.)
$tILU_0(0.5,0.25)$	18(0.11,0.29)	18(0.47,1.67)	17(2.00,9.70)	19(8.36,48.83)
$tILU_0(0.5,0.5)$	23(0.11,0.31)	24(0.46,1.97)	23(1.98,11.68)	24(8.16,57.76)
$Jacobi(0.5)$	27(0.03,0.26)	24(0.12,1.73)	23(0.65,11.54)	24(3.05,56.92)
$GaussSeidel$	18(0.03,0.17)	19(0.13,1.33)	18(0.64,8.65)	18(3.04,40.81)

(b) $V(2,2)$

Smoother \ N	3969	16129	65025	261121
$ILU_0(0.5)$	12(0.13,0.37)	13(0.57,2.72)	15(2.47,18.5)	50(10.6,265.)
$tILU_0(0.5,0.25)$	15(0.11,0.31)	15(0.47,2.02)	14(2.00,12.2)	14(8.36,54.3)
$tILU_0(0.5,0.5)$	15(0.11,0.30)	17(0.46,2.23)	16(1.98,12.9)	16(8.16,57.0)
$Jacobi(0.5)$	18(0.03,0.25)	17(0.12,1.66)	16(0.65,11.7)	16(3.05,57.2)
$GaussSeidel$	14(0.03,0.19)	15(0.13,1.40)	14(0.64,10.1)	14(3.04,48.5)

(c) $V(4,4)$

Smoother \ N	3969	16129	65025	261121
$ILU_0(0.5)$	10(0.13,0.48)	10(0.57,3.24)	11(2.47,23.31)	38(10.60,338.)
$tILU_0(0.5,0.25)$	12(0.11,0.40)	12(0.47,2.56)	11(2.00,15.59)	11(8.36,70.38)
$tILU_0(0.5,0.5)$	13(0.11,0.40)	13(0.46,2.46)	13(1.98,16.38)	12(8.16,71.03)
$Jacobi(0.5)$	15(0.03,0.35)	14(0.12,2.48)	13(0.65,17.73)	13(3.05,79.92)
$GaussSeidel$	12(0.03,0.29)	13(0.13,2.18)	12(0.64,15.23)	12(3.04,68.80)

data from Tables 4.25 and 4.26 suggest that V(2,2) cycle is an appropriate choice for a preconditioner (having in mind the requirements of robustness and minimal total execution time).

4.4 Parallelisation

The aim of numerical modelling is to produce simulations of increasingly more complex systems and phenomena. For example, modelling three-dimensional problems with greater accuracy, tackling complex multi-physics systems and performing simulations of time dependent problems. This aim can be achieved through the development of new, sophisticated numerical algorithms and/or increasing speed and memory capacity. However, progress in this area struggles to keep up with the ever more complex problems introduced. In such cases a natural progression is to use a parallel architecture. This approach puts additional requirements on the numerical

algorithm designers, that is, to develop algorithms that allow efficient parallelisation and effective implementation of such algorithms on modern parallel architectures. The most frequently used technique for parallelising numerical algorithms for the simulation of PDEs is domain decomposition [77, Chapter 14] [20, Section 8.2.7,9].

To solve the linear system (3.1) in parallel we use the solver configuration set out in Solver Strategy 4.4.

Solver Strategy 4.4

Krylov Method: Hypre GMRES (tolerance 10^{-6})
Preconditioner: Hypre BoomerAMG
  Coarsening:
    Classical Ruge-Stüben
    Strength of dependence: $\theta = 0.25$
  Cycle: $V(2,2)$
Smoothers:
  Parallel $Gauss - Seidel$
  Block Jacobi $ILU_0(0.5)$
  Block Jacobi $tILU_0(0.5, \alpha)$
  Parallel $Jacobi(0.5)$

In Solver Strategy 4.4 parallelism is considered only in the preconditioning step. A parallel AMG preconditioner BoomerAMG [50] is used from the Hypre library [2] [33]. The code is based on MPI and has several different options for parallel coarsening. Our experiments, unless stated otherwise, use a standard Ruge-Stüben method that neglects any connections between the unknowns across the subdomain boundaries. This approach is effective from an implementation point of view (as it does not require any interprocessor communication), however this can also result in a reduction in the efficiency of the preconditioner in some complex cases (e.g. strong convection). A solution to this is to use three-pass Ruge-Stüben coarsening [50], where a classical two-pass coarsening method is applied to the interior of each subdomain, followed by a third pass that affects only the nodes at the subdomain/processor boundaries (the nodes that have interconnections across the subdomain boundaries). The third pass of the coarsening algorithm requires interprocessor communication and may also lead to load imbalance [50]. BoomerAMG also offers other coarsening algorithms, such as CLJP [14] [50] and Falgout coarsening [50]. The CLJP coarsening algorithm has an advantage that it always generates the same sequence of coarse operators regardless of the number of subdomains. The disadvantage of the CLJP coarsening method is that it produces coarse level operators that are denser than those produced by the Ruge-Stüben coarsening. The Falgout coarsening is a trade-off between the two methods, it applies the standard Ruge-Stüben method in the interior of the subdomains (thus preserving the operator sparsity) and the CLJP coarsening to the

nodes at the subdomain boundaries (leading to better load balance than three-pass Ruge-Stüben coarsening) [50].

In the context of parallel smoothers, the Jacobi method is inherently parallel and the Gauss-Seidel method can be parallelised using red-black ordering [21]. Parallelisation of the ILU_0 method is achieved using a block Jacobi approach (see Page 93, 94 for further details) [77, p.378].

In Table 4.27 the convergence results of the above Solver Strategy applied to Case study 4.2.1, as a function of the number of processors for the problem size $N = 1046529$, is shown. From the table we see that parallelisation does not have a

Table 4.27: The iteration counts and the (setup,total) execution time (in seconds) required for the convergence of the parallel GMRES solver preconditioned by classical algebraic multigrid V(2,2) cycle with several smoothers, when applied to the solution of the discrete convection-diffusion problem (Case study 4.2.1) obtained from $Q1$ SUPG FEM on a uniform grid, with $N = 1,046,529$. Each sub-table represents a different diffusion parameter.

(a) Pe_*=500

Smoother \ P	1	2	4	8	16
$ILU_0(0.5)$	5(24.8,50.6)	5(13.6,28.5)	5(6.71,14.1)	5(3.72,7.56)	5(1.94,3.84)
$tILU_0(0.5, 0.25)$	8(18.9,39.5)	8(11.1,23.0)	8(5.69,11.7)	9(3.15,6.96)	8(1.62,3.49)
$tILU_0(0.5, 0.5)$	9(18.5,38.1)	10(10.7,22.5)	10(5.28,11.7)	10(3.06,7.22)	10(1.62,3.59)
$Jacobi(0.5)$	9(9.12,35.4)	9(5.89,19.4)	9(2.84,10.3)	10(1.66,6.11)	9(0.80,3.09)
$GaussSeidel$	5(9.03,27.4)	6(5.96,19.8)	6(2.94,12.0)	6(1.66,8.16)	6(0.85,6.70)

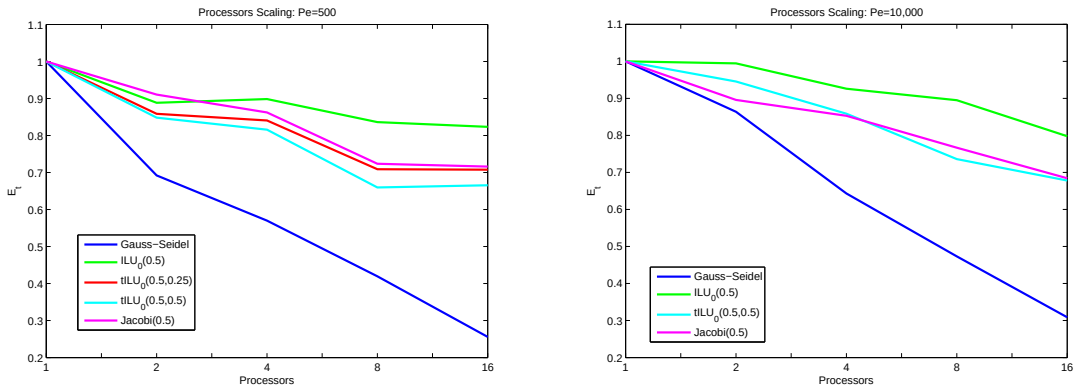
(b) Pe_*=10,000

Smoother \ P	1	2	4	8	16
$ILU_0(0.5)$	6(33.0,67.3)	6(16.6,33.8)	6(8.81,18.2)	6(4.81,9.40)	6(2.71,5.27)
$tILU_0(0.5, 0.5)$	8(23.2,43.8)	8(12.3,23.2)	8(6.74,12.8)	8(3.68,7.44)	8(2.21,4.04)
$Jacobi(0.5)$	9(14.0,41.3)	9(7.94,23.0)	9(4.39,12.1)	9(2.52,6.73)	9(1.41,3.77)
$GaussSeidel$	14(14.1,60.9)	14(7.93,35.2)	15(4.52,23.7)	15(2.44,16.1)	15(1.38,12.3)

negative impact to the iteration counts for all the smoothers considered. However, there is a lack of convergence in the solver for $Pe_* = 10,000$ when the $\text{tILU}_0(0.5, 0.25)$ smoother is used (this is consistent with the result from Table 4.19(c)). An important consideration in this case is the execution times. For $Pe_* = 500$ and $P = 1$ the best execution time is obtained when Gauss-Seidel smoother is used. However, parallel efficiency, of the Gauss-Seidel smoother implemented in BoomerAMG appears to be relatively poor, and for $P \geq 2$ the solver with damped Jacobi smoother performs well with the shortest execution time. For a larger number of processors both ILU_0 and tILU_0 smoothers' parallel performance exceeds that of the Gauss-Seidel smoother. For $Pe_* = 10,000$ the best execution times are obtained for the Jacobi smoother, irrespectively of the number of processors, closely followed by tILU_0(0.5,0.5) smoother. It is also noticeable that the actual GMRES solver time is shorter when tILU_0(0.5,0.5) smoother is used than when damped Jacobi smoother is used. However, the higher setup time involved in tILU_0(0.5,0.5) results in a marginally worse execution time

than in the damped Jacobi case. Furthermore, when the number of processors is increased, the solver with damped Jacobi smoother exhibits, as expected, marginally better efficiency than the solver with tILU_0 smoother.

In Fig 4.11 we present parallel efficiency of Solver Strategy 4.4 as a function of the number of processors. We can see that Gauss-Seidel smoothing results in a precondi-



(a) (b)

Figure 4.11: Parallel efficiency of GMRES solver with classical AMG preconditioning for Case study 4.2.1 with $N = 1,046,529$ (a) $Pe_* = 500$, (b) $Pe_* = 10,000$.

tioned solver that has very poor parallel scaling when used on more than 2 processors. By contrast, both Jacobi and ILU_0 smoothers show good parallel efficiency, which does not deteriorate substantially when the number of processors is increased. As expected, the parallel efficiency of the tILU_0 smoother generally lie close to damped Jacobi and ILU_0 smoothing.

The parallelisation of the preconditioned GMRES solver is not that successful for high values of Pe_* for Case study 4.2.2. The convergence results are summarised in Table 4.28. For the lower value of Pe_* there is a considerable growth in iteration counts when the number of processors is increased between 1 and 16, with the iteration counts roughly doubling overall for all smoothers. For $Pe_* = 10,000$, Table 4.28(b), there is a distinct increase in the iteration counts when multiple processors are used. In the case of Gauss-Seidel and ILU_0 smoothers, the increase in Pe_* and an increase in the number of processors, leads to an increase in the iteration count above 100, therefore these results have not been included in Table 4.28(b). The only smoother for which the solver converges is tILU_0. The reason for such poor convergence of the solver in parallel lies in the coarsening strategy used. A parallel Ruge-Stüben coarsening which disregards any connections between the nodes in different subdomains, without any method to compensate for the loss, will eventually begin to lose its efficiency. In the convection-diffusion cases, this effect is seen in Table 4.28. To

Table 4.28: The iteration counts and the (setup,total) execution time (in seconds) required for the convergence of the parallel GMRES solver preconditioned by classical algebraic multigrid V(2,2) cycle with several smoothers, when applied to the solution of the discrete convection-diffusion problem (Case study 4.2.2) obtained from $Q1$ SUPG FEM on a uniform grid, with $N = 1,046,529$. Each sub-table represents a different diffusion parameter.

(a) Pe_*=500

Smoother \ P	1	2	4	8	16
$ILU_0(0.5)$	7(27.5,63.2)	9(13.7,40.5)	11(7.40,25.1)	13(9.24,20.3)	13(2.22,8.31)
$tILU_0(0.5, 0.25)$	10(20.4,45.0)	14(11.1,30.2)	17(5.95,20.0)	20(40.8,49.3)	20(65.7,70.8)
$tILU_0(0.5, 0.5)$	11(20.3,45.9)	16(10.8,30.4)	20(6.22,19.2)	23(3.14,12.8)	22(1.85,6.67)
$Jacobi(0.5)$	11(9.87,41.7)	15(5.80,29.0)	18(3.23,20.8)	21(1.68,11.5)	21(0.94,6.94)
$GaussSeidel$	8(9.87,37.8)	11(5.87,29.1)	13(3.24,22.1)	14(1.73,16.0)	14(0.92,12.7)

(b) Pe_*=10,000

Smoother \ P	1	2	4	8	16
$tILU_0(0.5, 0.25)$	22(21.6,83.9)	61(11.2,108.6)	92(6.14,100.9)	89(41.7,85.9)	76(67.8,91.0)
$tILU_0(0.5, 0.5)$	19(21.0,70.8)	61(10.8,97.7)	95(6.08,82.1)	95(3.23,53.6)	83(1.99,23.9)
$Jacobi(0.5)$	25(9.82,85.3)	69(5.48,125.9)	X	X	93(1.05,32.1)

rectify the problem we apply a more sophisticated parallel coarsening method.

In Table 4.29 we repeat the tests performed previously for Case study 4.2.2 using a GMRES solver preconditioned by AMG with Falgout coarsening. The results in

Table 4.29: The iteration counts and the (setup,total) execution time (in seconds) required for the convergence of the parallel GMRES solver preconditioned by algebraic multigrid with Falgout coarsening and V(2,2) cycle with several smoothers, when applied to the solution of the discrete convection-diffusion problem (Case study 4.2.2) obtained from $Q1$ SUPG FEM on a uniform grid, with $N = 1,046,529$. Each sub-table represents a different diffusion parameter.

(a) Pe_*=500

Smoother \ P	1	2	4	8	16
$ILU_0(0.5)$	7(28.9,68.0)	8(14.1,38.2)	8(7.25,19.4)	8(3.97,10.9)	8(2.28,5.90)
$tILU_0(0.5, 0.25)$	11(21.6,51.1)	10(11.5,26.8)	10(5.92,13.0)	10(3.29,7.30)	10(1.95,4.15)
$tILU_0(0.5, 0.5)$	12(21.4,50.9)	11(11.3,27.1)	11(5.79,13.0)	11(3.21,7.33)	11(1.95,4.17)
$Jacobi(0.5)$	11(11.2,46.6)	11(6.30,24.9)	10(3.23,11.4)	11(1.82,6.79)	10(1.04,3.41)
$GaussSeidel$	8(11.3,37.4)	8(6.40,27.7)	8(3.23,22.5)	8(1.82,19.4)	8(1.06,17.7)

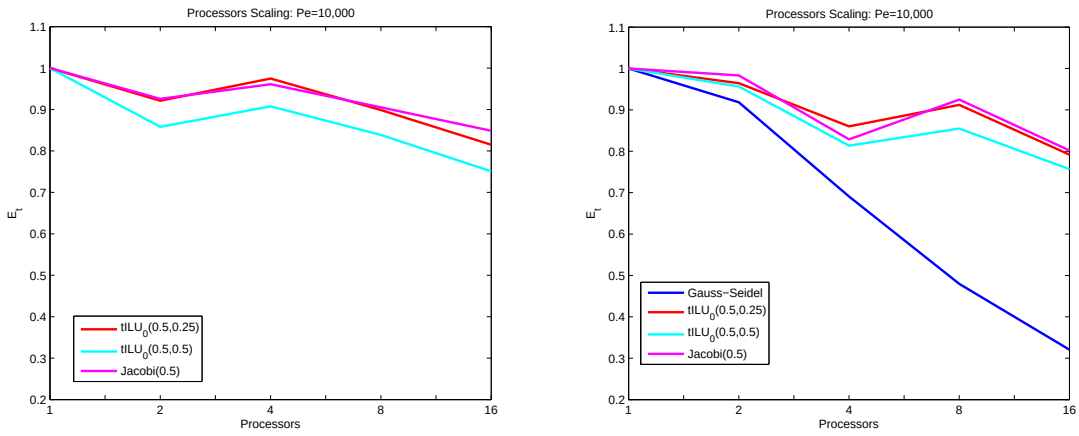
(b) Pe_*=10,000

Smoother \ P	1	2	4	8	16
$tILU_0(0.5, 0.25)$	22(22.7,90.7)	21(12.3,49.2)	20(6.81,23.3)	20(3.56,12.6)	20(2.22,6.96)
$tILU_0(0.5, 0.5)$	19(22.2,75.9)	20(12.0,44.2)	20(6.31,20.9)	19(3.58,11.3)	19(2.26,6.31)
$Jacobi(0.5)$	25(11.2,95.0)	25(6.66,51.3)	25(3.49,24.7)	24(2.05,13.1)	24(1.26,7.00)

Table 4.29 show a consistent iteration count in all the solvers with respect to an increasing number of processors. The shortest execution time for $Pe_* = 500$ is observed when the Jacobi smoother is used, regardless of the number of processors. In the highly convective case ($Pe_* = 10000$) presented in Table 4.29(b), once again Gauss-Seidel and ILU_0 smoothers are not shown as these smoothers have not facilitated GMRES to converge within an acceptable number of GMRES iterations (100

in our case). This raises the question as to the general capabilities of Gauss-Seidel as a smoother. In Table 4.29(b), it is clear that the smallest iteration count and shortest execution time is consistently observed for the $\text{tILU}_0(0.5, 0.5)$ smoother, regardless of the number of processors.

In Figure 4.12(a) we present parallel efficiency of the GMRES solver with AMG preconditioner using Falgout coarsening as a function of the number of processors for different parallel smoothers for Case study 4.2.2. The efficiency of tILU_0 and Jacobi



(a) (b)

Figure 4.12: Parallel efficiency of GMRES solver with AMG preconditioning using Falgout coarsening with $N = 1,046,529$ and $Pe_* = 10,000$ for (a) Case study 4.2.2, (b) Case study 4.2.4.

smoothers have improved in comparison to Figure 4.11(a). In Figure 4.12(a) there is a clear parallel relationship between tILU_0(0.5,0.5) and Jacobi. Furthermore the efficiency of tILU_0(0.5,0.25) is asymptotically identical to the Jacobi smoother with respect to the number of processors.

The convergence characteristics of GMRES/AMG/Gauss-Seidel method are significantly more Pe, h-robust for Case study 4.2.4 than for Case study 4.2.2 when a single processor is used. In Table 4.30 the parallel performance of the GMRES solver preconditioned with AMG using different smoothers for Case study 4.2.4, as a function of the number of processors for problem size $N = 1046529$ is shown. In Table 4.30(a) Ruge-Stüben coarsening is used. Once again we see a clear dependence in the iteration counts with respect to the number of processors for all smoothers. In the case of $P = 16$ the Gauss-Seidel smoother has the same iteration count as tILU_0(0.5,0.25). However the execution time of Gauss-Seidel is three times larger than tILU_0(0.5,0.25). We see that tILU_0 is more closely related to the Jacobi smoother. In Table 4.30(b) Falgout coarsening is used. We observe robust performance, with respect to the number of processors, which indicates that the problem

Table 4.30: The iteration counts and the (setup,total) execution time (in seconds) required for the convergence of the parallel GMRES solver preconditioned by algebraic multigrid V(2,2) with several smoothers, when applied to the solution of the discrete convection-diffusion problem (Case study 4.2.4) obtained from $Q1$ SUPG FEM on a uniform grid, with $N = 1,046,529$. A constant diffusion parameter ($Pe_* = 10,000$), with each sub-table representing a different coarsening strategy.

(a) Ruge-Stüben coarsening

Smoother \ P	1	2	4	8	16
$tILU_0(0.5, 0.25)$	12(22.9,58.5)	15(11.3,35.2)	21(6.91,29.3)	21(3.36,14.2)	38(1.95,11.8)
$tILU_0(0.5, 0.5)$	15(21.1,59.5)	19(11.1,37.3)	26(6.09,26.1)	26(3.12,12.7)	44(1.87,12.0)
$Jacobi(0.5)$	15(10.4,58.6)	18(5.81,36.1)	26(3.69,32.0)	26(1.78,17.0)	45(0.97,13.5)
$GaussSeidel$	12(10.4,55.9)	15(5.77,39.5)	21(3.40,35.7)	21(1.80,24.7)	38(0.98,34.7)

(b) Falgout coarsening

Smoother \ P	1	2	4	8	16
$tILU_0(0.5, 0.25)$	12(24.3,68.1)	12(12.5,35.3)	12(6.96,19.8)	11(3.66,9.33)	12(2.20,5.37)
$tILU_0(0.5, 0.5)$	14(23.5,69.5)	14(12.3,36.3)	15(6.84,21.3)	14(3.60,10.2)	15(2.10,5.74)
$Jacobi(0.5)$	14(12.4,68.5)	14(6.81,34.8)	15(4.00,20.6)	13(1.99,9.25)	15(1.14,5.34)
$GaussSeidel$	12(12.4,70.2)	12(6.81,38.2)	12(4.02,25.4)	12(1.98,18.3)	12(1.15,13.7)

of poor performance from Table 4.30(a) can be attributed to the poor coarsening strategy used in this case. The results in Table 4.30(b) show a clear robustness with respect to the number of processors for all smoothers. Although the solver that uses Gauss-Seidel smoother has a small iteration count, the execution time in comparison to all other smoothers is more than double, due to a poor parallel scaling of the Gauss-Seidel smoother.

In Figure 4.12(b) we present parallel efficiency of the GMRES solver using Falgout coarsening for Case study 4.2.4. Using Gauss-Seidel smoothing results in a preconditioned solver that has poor parallel scaling, this is consistent with our findings for Case study 4.2.1, Table 4.11(b). The parallel scaling of tILU_0 on the other hand is very encouraging, as it scales almost as the Jacobi smoother, which is inherently parallel.

4.5 Performance profile and summary

In summary of this chapter, we studied the iteration count, execution time and parallel scaling of MG preconditioned GMRES solver with different smoothers applied to a number of two-dimensional case studies.

In the case of the diffusion problem the GMG preconditioner generally led to a marginally lower iteration count, in comparison to AMG given the same parameters and choice of smoothers. For convection-dominated problems we observed that the new tILU_0 smoother was better in terms of execution time than ILU_0 and damped Jacobi smoothers and competitive with the case when Gauss-Seidel smoother is used.

The advantage of the tILU_0 smoother over Gauss-Seidel was apparent in the case of recirculating wind (Case study 4.2.2) when no special node ordering is used and in the case when this problem is discretised by stretched or adaptively refined girds.

When parallel performance of the iterative solver is considered, tILU_0 smoother (including its limiting case, damped Jacobi) exhibited much better scalability than the Gauss-Seidel smoother, and for the majority ILU_0.

To give an overall view of the performance of the smoothers when used in conjunction with MG preconditioned GMRES we now consider performance profiling [51, pp.320–327] based on total execution times from this chapter. The performance profile is plotted as a function of $\phi_j(\theta)$ against θ for all solver j, where θ represents a tolerance (factor of the fastest solver) and $\phi_j(\theta)$ the probability of being the fastest solver within a factor θ of the best performance over all solvers of the given set of test problems [51, p.320].

In Figure 4.13(a) we find the performance profile of GMG preconditioned GMRES solver for the largest problem size N with $Pe_* = 500$ and 2000 for all tables within this chapter (from Table 4.2 to Table 4.12). The performance profile shows that Gauss-Seidel is the fastest solver on 80% of the problems (shown when $\theta = 1$). However, $\text{tILU}_0(0.5, 0.25)$ reaches the probability $\phi_j(\theta) = 1$ first given a factor $\theta = 1.2$ of the fastest solver on every problem. This shows that $\text{tILU}_0(0.5, 0.25)$ is the more reliable smoother in the sense of being the least likely to be much slower than the fastest.

Figure 4.13(b) shows the performance profile of AMG preconditioned GMRES solver for the largest problem size N with $Pe_* = 500$ and 10000 for all tables within this chapter (from Table 4.19 to Table 4.21). For AMG we have concluded that $\alpha = 0.5$ is the most appropriate truncation parameter for the tILU_0 smoother. In Figure 4.13(b) we find that Gauss-Seidel is the fastest solver on roughly 60% of the problems and $\text{tILU}_0(0.5, 0.5)$ on the remaining problems, in the strictest sense (i.e $\theta = 1$). By comparing the performance profile of GMG and AMG preconditioned GMRES solver with Gauss-Seidel smoother, we observe a performance reduction in the AMG case (i.e Gauss-Seidel is less competitive over all problems). Initially we must note that we are using larger Pe_* for the AMG case ($Pe_* = 10000$) in comparison to GMG ($Pe_* = 2000$). An explanation for the performance reduction of AMG with Gauss-Seidel, is that for large Pe_* Gauss-Seidel fails to converge within the maximum number of iteration for Case studies 4.2.2 and 4.2.3 compared to tILU_0 and Jacobi. If we were to select a solver that has a 75% chance of being within a factor of roughly 1.5 of the fastest solver then all smoothers are equally successful, with the exception of ILU_0. The severe drop in performance of ILU_0 it not only down to ILU_0 failing to converge within the maximum number of iteration on a number of occasions, but also because the coarse level matrices generated by classical AMG

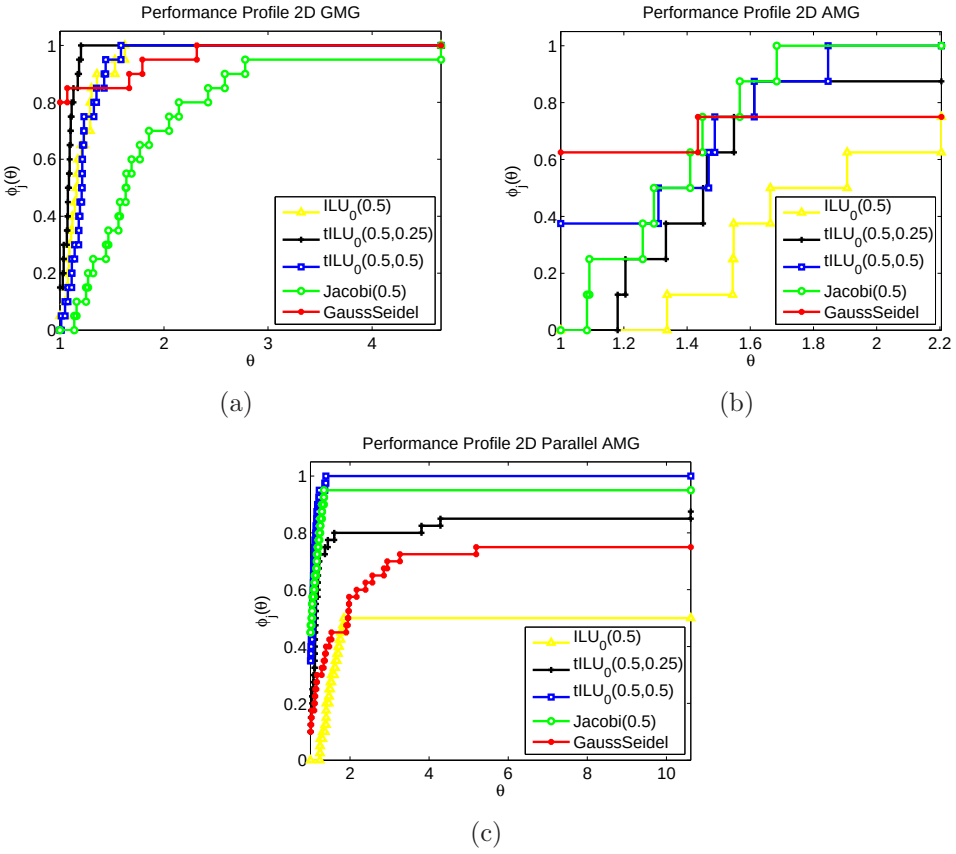


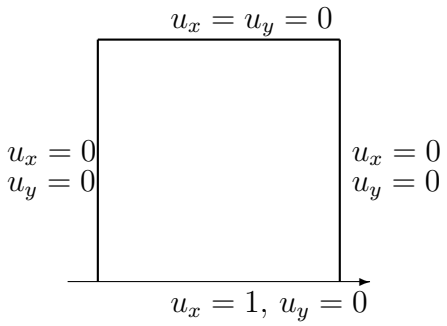
Figure 4.13: Performance profile for Case study 4.2.1, 4.2.2, 4.2.3 and 4.2.4, based on total execution time. GMRES solver with (a) GMG preconditioning (Pe_*=500, 2000), (b) AMG preconditioning (Pe_*=500, 10000) and (c) Parallel AMG preconditioning (Pe_*=500, 10000).

are considerably denser than the coefficient matrix (Table 4.22), which makes ILU_0 computationally expensive. As θ is increased we find that Jacobi(0.5) is the first smoother to reach $\phi_j(\theta) = 1$, within a factor of the fastest solver on every problem, followed by $\text{tILU}_0(0.5, 0.5)$. This further enforces that Gauss-Seidel is not the most robust of smoothers we tested and makes a further case for the use of $\text{tILU}_0(0.5, 0.5)$ smoother for the AMG preconditioner.

Figure 4.13(c) shows the performance profile of parallel AMG preconditioned GMRES solver for the largest problem size N and number of processors $P = 1, 2, 4, 8, 16$ with $Pe_* = 500$ and 10000 for all tables within this chapter (from Table 4.27 to Table 4.30). As is expected Gauss-Seidel performs poorly in parallel circumstances. We have seen from Section 4.4 that $\text{tILU}_0(0.5, 0.5)$ scales well. It is clear from Figure 4.13(c) that $\text{tILU}_0(0.5, 0.5)$ is robust with respect to the number of processors, however this is now reached for a very small factor θ. In conclusion it is clear that tILU_0 smoother is robust and robust with respect to the number of processors for two-dimensional problems.

4.6 Floor-driven cavity problem for the Navier-Stokes equations

In this section we test the effectiveness of the novel smoothing strategy in the context of block MG preconditioning of the Navier-Stokes equations. The implementation details of the "floor-driven cavity" problem is from OOMPHLIB [45]. This is an equivalent problem to lid-driven cavity [31, p.317]. The non leaky boundary conditions used for this case are



The solution of the Navier-Stokes equations (2.34) requires finding the unknown vector velocity field $\vec{u} = [u_x; u_y]$ and pressure p. In Figure 4.14 we present the solution of the floor-driven cavity for three different values of Re: (a) 20, (b) 200 and (c) 1000. The solution is presented as the pressure field (represent as a contour plot) and the velocity field (represented as streamlines [31, p.216]). In Figure 4.14(a) we see a recirculating flow in a anti-clockwise direction, with small eddies in the top corner of the domain. The pressure is fairly constant throughout the domain, with a large plume representing an increase in pressure around the bottom right corner, due to the movement of the floor. In Figure 4.14(b) we observe a distinct change in the position of the central eye of the recirculating flow, and in Figure 4.14(c) we see a further change in the position of the central eye of the recirculating flow towards the centre of the domain. In cases when $Re > 10^4$ for a two-dimensional case, the flow becomes unstable, and we begin to observe time periodic states [31, p.317]. That is, the flow loses its stability via the Hopf bifurcation [31, p.323].

A system of nonlinear equations obtained from FE discretisation of the problem (2.34) is solved iteratively using Newton's method (Algorithm 2.1). This linearisation procedure requires the solution of a linear system at every iteration (Newton step). The stopping criterion for Newton's method is implemented in OOMPHLIB as either the maximum number of Newton steps (10) or when $||(\overline{r}_N^k)||_\infty < 10^{-7}$ is reached.

When Navier-Stokes problems with high value of Re are solved by Newton's

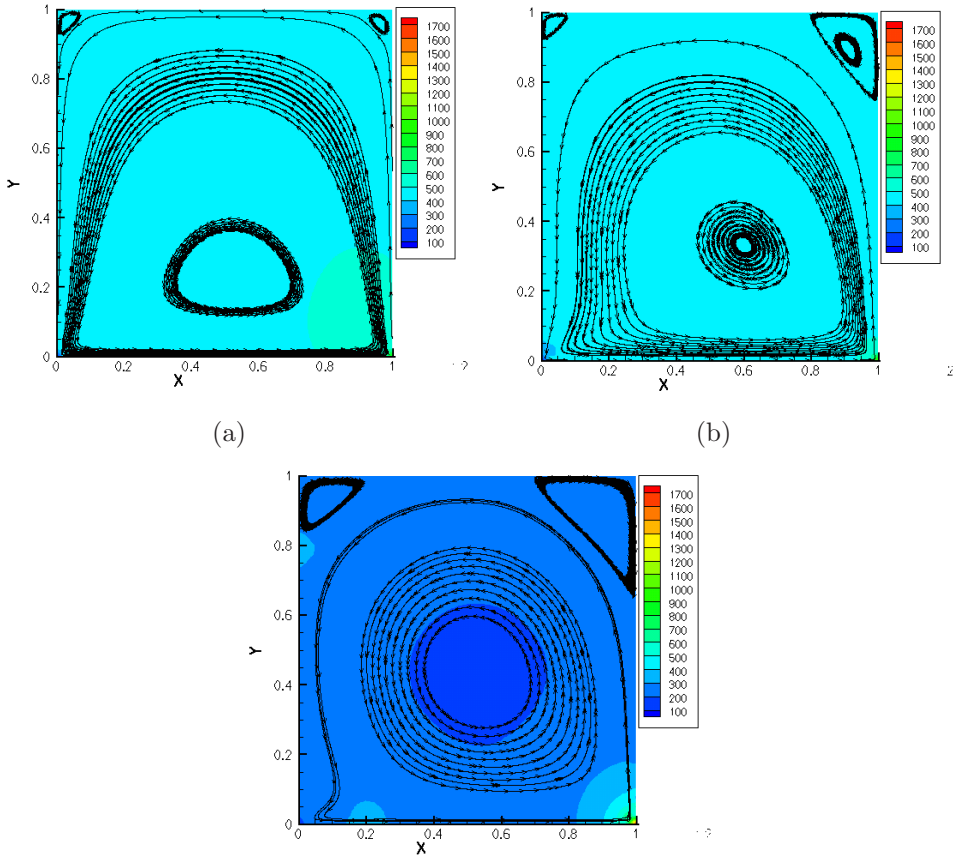


Figure 4.14: Floor-driven cavity problem: velocity streamlines and pressure field for (a) $Re = 20$, (b) $Re = 200$, (c) $Re = 1,000$. Discretisation of the problem is performed using Taylor-Hood elements on a uniformly refined mesh with $N = 14,162$ and a zero source term.

method, special care must be taken when selecting an initial solution guess. Newton's method is known to converge quadratically, providing that the current solution approximation is "sufficiently close" to the true solution [31, p.326]. The "ball of convergence" of Newton's method in the Navier-Stokes case has a radius typically inversely proportional to Re [31, p.326]. This implies that in the case of highly convective flows we need a more accurate initial guess to ensure convergence of Newton's method. This can be achieved in two ways. The first is to use a hybrid Picard/Newton method to solve a nonlinear system. Starting with an arbitrary initial guess (e.g. a zero vector) we apply a small fixed number of Picard's iterations to obtain an initial guess for Newton's method. The alternative is to apply a version of the continuation method where we increase gradually the value of Re and solve the problem for each particular Re, using the solution from the previous continuation step as the initial guess for the current solve. In our computations we apply pure Newton's method (as

no difficulties with convergence are observed in cases we tested). However, we found experimentally that to obtain a convergent solution with $Re > 500$, the continuation method (with increments $\Delta Re = 100$) is needed to obtain a sufficiently good initial guess of the solution.

We consider an iterative solution strategy for the linear system (2.40). We use GMRES as a solver, preconditioned by the inexact LSC preconditioner [31, Chapter 8]. In our application the entire momentum block F is approximately inverted by the "black-box" application of AMG. Although AMG is ideally suited only for discrete scalar elliptic problems, the assumption is that it will work reasonably well when applied to the momentum block F. The rationale behind this assumption is that the main diagonal blocks F_x and F_y in the momentum block are scalar convection-diffusion matrices, perturbed by the Newton blocks W_{xx} and W_{yy} (2.44). In the LSC approximation of the Schur complement $S = BF^{-1}B^T$, given by

$$\widetilde{S} = (BM_u^{-1}B^T)(BM_u^{-1}FM_u^{-1}B^T)^{-1}(BM_u^{-1}B^T),$$

with M_u as a velocity mass matrix [31, p.347]. The matrix block $BM_u^{-1}B^T$, is spectrally equivalent to the pressure Poisson operator D_p [31, p.348], and is inverted approximately by MG. Then the preconditioning step at each GMRES iteration requires the approximate solution of a linear system (4.14):

$$\begin{bmatrix} F_x + W_{xx} & W_{xy} & B_x^T \\ W_{yx} & F_y + W_{yy} & B_y^T \\ 0 & 0 & -\widetilde{S} \end{bmatrix} \overbrace{\begin{bmatrix} z_x \\ z_y \\ z_p \end{bmatrix}}^{\overline{z}} = \overbrace{\begin{bmatrix} r_x \\ r_y \\ r_p \end{bmatrix}}^{\overline{r}}. \tag{4.14}$$

The solution of the system (4.14) is achieved by a block back substitution. With each Newton step the only change to the momentum block is the Newton derivative contribution W. In (4.14) $[r_x, r_y, r_p]^T$ is the linear residual, at the current GMRES iteration ($\overline{r}^k = R(X) - J(X)\overline{\delta X}^k$).

We apply Gauss-Seidel smoother in MG approximation of the pressure Poisson operators $BM_u^{-1}B^T$ in $\widetilde{S}$ and test the efficiency of a variety of smoothers in MG approximation of the momentum block F. In the case of the tILU$_0$ smoother we test two levels of truncation, $\alpha = 0.25$ and $\alpha = 0.5$. Thus, to solve the linear system (4.14) at each Newton step we use the solver configuration set out in Solver Strategy 4.5.

In Table 4.31 we present the average values of the iteration counts of the preconditioned GMRES solver and the setup/total execution times over the entire course of Newton steps. In all tests, the average step counts do not entirely reflect the full picture, as the iteration counts at the first Newton step are always considerably smaller

Solver Strategy 4.5

Krylov Method: GMRES (tolerance 10^{-6})
Preconditioner: LSC
  Pressure Poisson Block D_p: Hypre BoomerAMG
    Coarsening:
      Classical Ruge-Stüben
      Strength of dependence: $\theta = 0.25$
    Cycle: $V(2,2)$
    Smoothers:
      $Gauss - Seidel$
  Momentum Block F: Hypre BoomerAMG
    Coarsening:
      Classical Ruge-Stüben
      Strength of dependence: $\theta = 0.25$
    Cycle: $2 * V(2,2)$
    Smoothers:
      $Gauss - Seidel$
      $ILU_0(\gamma)$
      $tILU_0(\gamma, \alpha)$
      $Jacobi(\gamma)$

Table 4.31: The iteration counts and the (setup,total) execution time (seconds) required for the convergence of the GMRES solver preconditioned by the LSC preconditioner applied to the discrete Navier-Stokes problem obtained by the Q2-Q1 discretisation of Case study 4.6. The convection-diffusion block within the momentum block are preconditioned by a V(2,2) cycle of classical AMG and a variety of smoothers (damping $\gamma = 0.5$ is used).

(a) Re=200

Smoother \ N(Grid)	3482(20×20)	14162(40×40)	32042(60×60)	57122(80×80)
$ILU_0(0.5)$	20(0.36,2.84)	30.5(1.78,29.7)	42.5(4.36,101.6)	56.2(8.25,255.9)
$tILU_0(0.5, 0.25)$	26.5(0.23,1.77)	42.2(1.14,21.5)	62.8(2.84,82.5)	87.6(5.27,228.4)
$tILU_0(0.5, 0.5)$	28.3(0.22,1.77)	44.2(1.17,22.0)	65.3(2.90,87.6)	90.6(5.19,227.7)
$Jacobi(0.5)$	27.3(0.16,1.77)	42.3(0.87,20.8)	62.7(2.18,90.2)	87.2(4.21,247.0)
$GaussSeidel$	22.2(0.15,1.42)	35(0.89,17.5)	51.7(2.21,75.5)	70.8(4.21,200.5)

than the average. The lowest iteration counts in Table 4.31 are seen when AMG preconditioner with the ILU_0 smoother is used for F, and the shortest execution times are seen in the case of Gauss-Seidel smoothing. Using $tILU_0$ as a smoother results in execution times that are 10% shorter than for the preconditioner using either ILU_0 or Jacobi smoothers.

In Table 4.32, we report the convergence characteristics of the preconditioned GMRES solver without damping ($\gamma = 1.0$) for the ILU_0/$tILU_0$/Jacobi smoothers. The results are presented for three different values of Re. For these cases Newton's method converges roughly in 3, 6 and 8 steps respectively. For completeness, we also present the convergence characteristics of the GMRES solver with the exact LSC preconditioner (i.e. where all the inverse blocks in the LSC preconditioner

Table 4.32: The iteration counts and the (setup,total) execution time (seconds) required for the convergence of the GMRES solver preconditioned by the LSC preconditioner applied to the discrete Navier-Stokes problem obtained by the Q2-Q1 discretisation of Case study 4.6. The convection-diffusion block within the momentum block are preconditioned by a V(2,2) cycle of classical AMG and a variety of smoothers (damping $\gamma = 1.0$ is used). Each sub-table represents a different value of the Reynolds number.

(a) Re=20

Smoother \ N Grid	3482 (20×20)	14162 (40×40)	32042 (60×60)	57122 (80×80)	89402 (100×100)
$ILU_0(1.0)$	10.7(0.35,1.62)	20(1.83,20.4)	30(4.68,80.3)	41(8.63,201.4)	53(12.6,337.6)
$tILU_0(1.0, 0.25)$	17.3(0.22,1.12)	33(1.21,17.0)	51.3(3.04,75.4)	69.7(5.71,194.2)	91.7(8.17,337.1)
$tILU_0(1.0, 0.5)$	20.7(0.22,1.23)	39(1.24,21.1)	59.3(3.05,83.0)	83.3(5.84,228.0)	107.7(8.15,388.5)
$Jacobi(1.0)$	16.3(0.16,1.05)	31.7(1.00,16.9)	49(2.48,76.8)	68(4.13,172.1)	88(6.64,358.8)
$GaussSeidel$	16(0.17,1.07)	31(0.97,16.0)	46.3(2.44,72.7)	65(4.13,162.6)	85.3(6.70,346.6)
$Direct$	9(0.39,0.55)	11(7.12,8.64)	12.7(23.9,28.5)	13.7(69.8,80.1)	15(151.8,171.4)

(b) Re=200

Smoother \ N Grid	3482 (20×20)	14162 (40×40)	32042 (60×60)	57122 (80×80)	89402 (100×100)
$ILU_0(1.0)$	19.8(0.35,2.85)	29.3(1.79,30.5)	39.5(4.74,108.)	50.8(8.06,217.6)	65.8(12.8,443.8)
$tILU_0(1.0, 0.25)$	23.3(0.24,1.67)	37.3(1.20,21.1)	54.5(3.00,83.7)	75.8(5.06,182.3)	93.8(8.91,396.8)
$tILU_0(1.0, 0.5)$	26.7(0.23,1.74)	42.7(1.19,22.6)	63.7(3.03,95.2)	89.2(4.99,210.5)	112.6(7.89,438.9)
$Jacobi(1.0)$	23(0.16,1.61)	36(0.89,18.4)	53(2.37,86.3)	72.8(3.98,192.9)	94.8(6.89,407.7)
$GaussSeidel$	22.2(0.16,1.44)	35(0.95,18.4)	51.7(2.33,82.1)	70.8(4.07,187.2)	92.6(6.51,393.7)
$Direct$	18.7(0.39,0.72)	22.2(7.12,10.2)	25.3(23.9,33.0)	26.2(69.8,89.2)	28.6(157.1,206.9)

(c) Re=400

Smoother \ N Grid	3482 (20×20)	14162 (40×40)	32042 (60×60)	57122 (80×80)	89402 (100×100)
$ILU_0(1.0)$	30.1(0.34,3.91)	35.6(1.74,32.4)	44.3(4.19,99.2)	56.9(7.84,240.1)	X
$tILU_0(1.0, 0.25)$	34.3(0.21,2.16)	44.3(1.11,21.7)	59.9(2.68,73.9)	79.1(5.01,191.6)	102.1(7.89,393.9)
$tILU_0(1.0, 0.5)$	53.9(0.21,3.18)	49.3(1.11,23.4)	67(2.68,84.1)	89.9(4.99,213.4)	117(8.06,468.2)
$Jacobi(1.0)$	60.3(0.15,3.70)	43.3(0.83,21.0)	58.3(2.08,80.4)	76.3(3.96,203.4)	98.1(6.41,442.2)
$GaussSeidel$	53.1(0.15,3.19)	42.1(0.84,20.0)	56.4(2.08,76.8)	74(3.95,194.1)	95(6.76,446.1)
$Direct$	28.4(0.39,0.90)	29.6(7.12,11.1)	32.6(23.9,35.6)	35.4(69.7,96.1)	38.1(151.8,200.6)

are performed exactly using a direct solver). In Table 4.32 we see that there is a considerable increase in iteration counts when the mesh is refined with constant Re. This trend is observed for all values of Re that are tested. However, it was observed that mesh dependence is reduced as the number of V-cycles, applied to the approximate inverse of the momentum block, increases. Taking more than two MG V-cycles, however, leads to an increase in the total execution time. For a fixed grid, the iteration counts grow as Re is increased. These trends are present in both the exact and inexact LSC preconditioners. For the inexact LSC preconditioner the best execution times are observed when $\text{tILU}_0(0.5,0.25)$ as a smoother is used, closely followed by Gauss-Seidel and Jacobi smoothers.

If we compare Tables 4.31 and 4.32(b) (the results for $Re = 200$ obtained for damped ($\gamma = 0.5$) and undamped ($\gamma = 1$) versions of ILU_0/tILU_0/Jacobi smoothers),

we see that slightly lower iteration counts are observed when $\gamma = 1$. This is contrary to the case of scalar discrete convection-diffusion problems, where much better convergence results are obtained when the smoothers are damped. A potential explanation for this is the fact that in the Navier-Stokes case MG is applied to the entire momentum block, which represents a vector-valued problem. Also, in addition to the convection-diffusion operators, the momentum block contains the Newton derivative blocks W, which may have a negative impact on the performance of MG (see [31, p.357, p.361]). We also remark that the convection-diffusion operators in the momentum block are discretised using standard Galerkin FEM, which can also have a detrimental effect on the MG performance if a sufficiently fine grid is not used.

In conclusion, the new smoother shows potential in the context of block preconditioning of the Navier-Stokes equations. However, further tests are needed on various model and realistic problems to obtain a more complete picture of the effectiveness and competitiveness of the new tILU_0 smoother in the context of fluid mechanics problems.

Chapter 5

Three-Dimensional Case Studies

A natural extension to the numerical study from the previous chapter is to test the new smoothing technique, tILU_0, on a range of three-dimensional convection-dominated problems. In this chapter we present the results of a comprehensive numerical evaluation of the tILU_0 smoothing technique for multigrid preconditioning performed on a number of discrete convection-diffusion problems in three spacial dimensions.

The main consequence of moving to a higher spacial dimension is that, for the same grid resolution, the resulting linear system and the matrix stencils are considerably larger. That is, a Q_1 discretisation on a uniform tensor product grid in 2D produces a coefficient matrix with a 9 point stencil. In the same context, a 3D problem will have a matrix stencil of size 27. This increase in both the problem size and the number of non-zero entries in the coefficient matrix may have a significant effect on the computational overhead and parallel performance of the algorithm used to solve the linear system. We introduce two case studies in three spacial dimensions that represent direct extensions of the Case studies 4.2.1 and 4.2.2. All the plots of the solutions in 3D are given as a sequence of two-dimensional surface plots at a distance of 0.05, 0.5 and 0.95 from the side walls.

In this chapter we test the Krylov solver initially with GMG as a preconditioner with either default or lexicographical node ordering. In Section 5.2 we test classical AMG as a preconditioner, and in Section 5.3 we evaluate the parallel performance of classical parallel AMG as a preconditioner. The primary objective in this chapter is to evaluate the effectiveness, the performance and the spacial robustness (robust with an additional requirement that performance is consistent over spacial dimensions) of the MG preconditioner with the novel tILU_0 smoother. The evaluation consists of comparing the tILU_0 smoother with the results obtained using the limiting cases of tILU_0 smoother (point Jacobi and ILU_0) as well as the standard point symmetric Gauss-Seidel smoother with reflection on the previous results in Chapter 4.

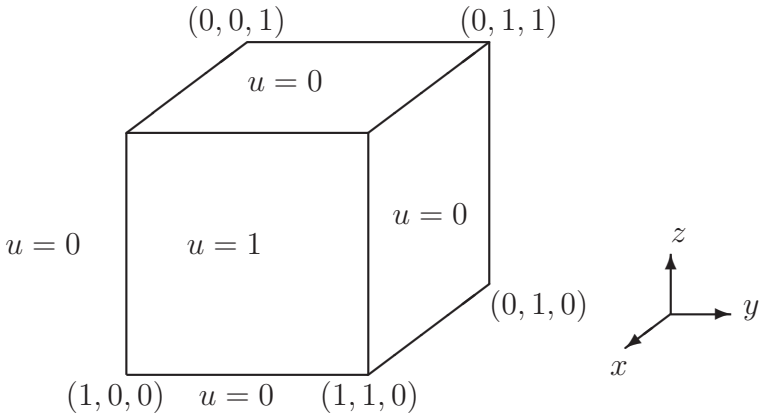
5.1 Geometric multigrid preconditioning of the convection-diffusion problem

In this section we present the convergence results, obtained from solving the systems of linear equations that arise in the SUPG FEM discretisation of two different convection-diffusion problems in three spacial dimensions. The linear solvers used in this section have the same configuration as in the two-dimensional cases. That is, GMRES method preconditioned with GMG which uses several different smoothers (Solver Strategy 4.2). This allows a direct comparison of the linear solvers' behaviour, when the spacial dimension is increased. The efficiency of the preconditioners are examined on a range of different problem parameters and different discretisations.

5.1.1 Constant uni-directional wind

This case represents a convection-diffusion model (2.7) for temperature distribution on a cube domain $\Omega = [0,1]^3$. We set one boundary ($x = 1$) to be hot ($u = 1$), and the remaining boundaries are assigned to homogeneous Dirichlet BCs as follows:

$$\begin{aligned} &u(x = 1, 0 \leq y \leq 1, 0 \leq z \leq 1) = 1 && \text{(hot wall)} \\ &u = 0\,, \text{ elsewhere on } \partial\Omega && \text{(cold wall).} \end{aligned} \tag{5.1}$$



The convection field is uniform and uni-directional in the negative x-direction ($\vec{w} = (-1, 0, 0)$) and is presented as an arrow plot in Figure 5.1(a).

In Figure 5.1(b)(c)(d) the solution u is depicted as contour plot slices for $Pe_* = 0$, 500 and 10,000 respectively, assuming a zero source term ($f = 0$). From Figure 5.1 we can see that the increase in Pe_* leads to the creation of steep solution layers near all "cold" walls.

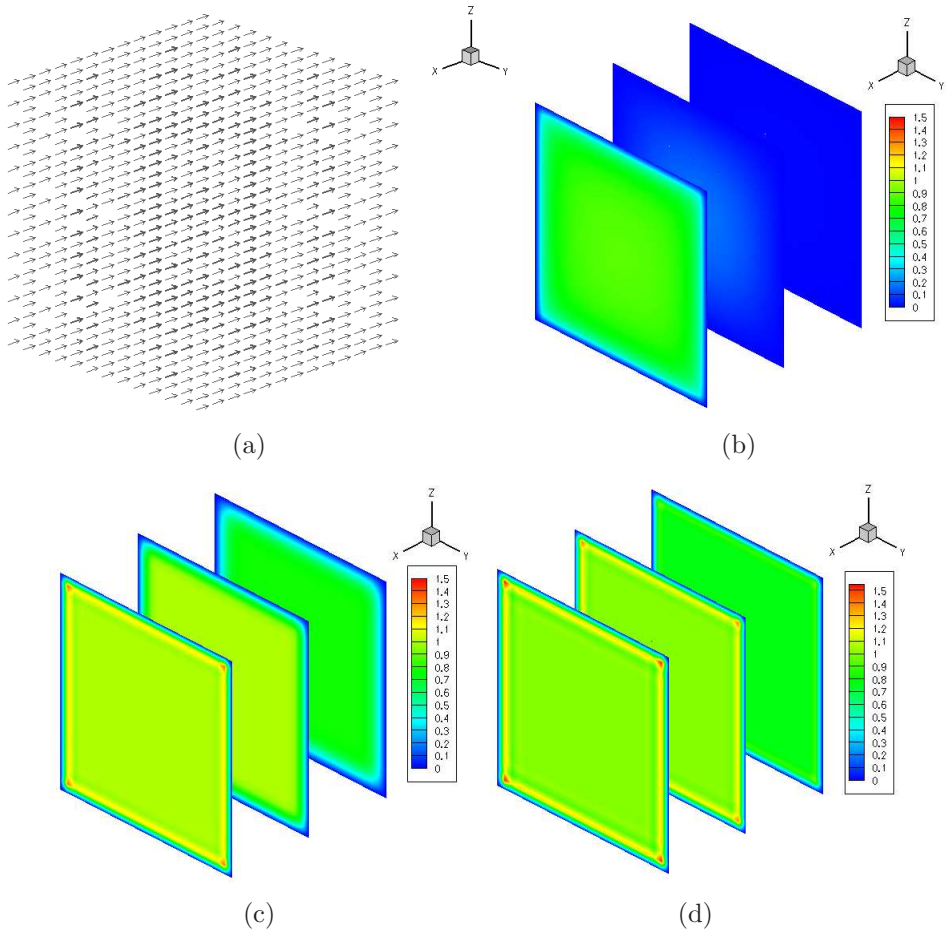


Figure 5.1: The convection-diffusion problem (Case study 5.1.1): constant, unidirectional wind $\vec{w} = (-1, 0, 0)$, no source term ($f = 0$); (a) Arrow plot of the wind. (b) the solution for $Pe_* = 0$, (c) the solution for $Pe_* = 500$, (d) the solution for $Pe_* = 10,000$.

Uniform grid refinement

This is the simplest case of discretisation, where a tensor product grid of brick elements is constructed and the convection-diffusion problem (2.7) with BCs (5.1) is discretised using SUPG FEM with trilinear approximation Q_1 [31, p.26]. Furthermore, a default (tree-based) nodal ordering is used [45].

In Table 5.1 we summarise the performance of Solver Strategy 4.2 for Case study 5.1.1 with default ordering. When $\text{ILU}_0(0.5)$ smoother is used, the results from the table show that the iterative solver performs Pe, h-robust. This was also the case in 2D (see Table 4.2). However, large computational cost associated with the assembly of the smoother makes this approach not competitive in 3D. When tILU_0 is used as a smoother, for this example, it appears that there is little difference in the performance of the iterative solver for the two different levels of truncation that are tested

Table 5.1: The iteration counts and the (setup,total) execution time (in seconds) required for the convergence of the GMRES solver right-preconditioned by geometric multigrid V(2,2) cycle with several smoothers, when applied to the solution of the discrete convection-diffusion problem (Case study 5.1.1) obtained from $Q1$ SUPG FEM on uniform grids. Natural (tree-based) ordering of the unknowns, with each sub-table representing a different diffusion parameter.

(a) Pe_*=500

Smoother \ N	1331	12167	103823	857375
$ILU_0(0.5)$	4(0.89,1.00)	5(7.14,8.82)	6(58.5,79.8)	6(524.1,695.6)
$tILU_0(0.5, 0.25)$	8(0.89,0.93)	7(7.14,7.58)	8(58.7,64.5)	8(513.1,562.9)
$tILU_0(0.5, 0.5)$	8(0.89,0.94)	7(7.14,7.58)	8(58.8,64.6)	11(512.9,578.2)
$Jacobi(0.5)$	16(0.89,0.99)	17(7.13,8.50)	18(58.6,77.8)	17(511.8,656.1)
$GaussSeidel$	8(0.89,0.94)	6(7.12,7.56)	5(58.5,63.7)	5(510.8,552.0)

(b) Pe_*=1,000

Smoother \ N	1331	12167	103823	857375
$ILU_0(0.5)$	4(0.89,0.99	5(7.13,8.81)	6(58.8,80.2)	6(511.9,683.5)
$tILU_0(0.5, 0.25)$	9(0.90,0.94)	8(7.15,7.64)	9(59.0,65.3)	8(511.4,561.3)
$tILU_0(0.5, 0.5)$	9(0.89,0.94)	8(7.13,7.64)	9(59.1,65.5)	8(524.1,576.8)
$Jacobi(0.5)$	18(0.89,1.00)	21(7.13,8.80)	24(58.8,84.1)	21(512.5,689.4)
$GaussSeidel$	9(0.89,0.93)	9(7.12,7.76)	7(58.8,65.9)	6(513.3,561.5)

(c) Pe_*=2,000

Smoother \ N	1331	12167	103823	857375
$ILU_0(0.5)$	4(0.89,1.00)	5(7.13,8.80)	5(58.5,76.8)	6(520.0,695.6)
$tILU_0(0.5, 0.25)$	10(0.89,0.94)	10(7.13,7.74)	10(58.5,65.6)	10(518.5,581.8)
$tILU_0(0.5, 0.5)$	10(0.89,0.94)	10(7.14,7.75)	10(58.7,65.5)	10(522.1,586.6)
$Jacobi(0.5)$	20(0.90,1.01)	29(7.13,9.47)	49(58.8,112.1)	43(518.6,908.4)
$GaussSeidel$	10(0.89,0.93)	12(7.12,7.94)	10(58.4,68.1)	7(517.2,574.9)

($\alpha = 0.25$ and 0.5). Comparable execution times in both cases suggest approximately the same level of truncation of non-zero entries in the MG operators' hierarchy. The iterative solver with tILU_0 as a smoother shows h-robust and a minor deterioration in performance with respect to Pe_*. Also, the execution times for tILU_0 are approximately 15% shorter than when ILU_0 is used. When Jacobi smoothing is used, the resulting iterative solver exhibits dependence in iteration counts both with respect to the problem size and Pe_*. This makes the solve strategy with the Jacobi smoother less competitive in terms of the execution time than when either ILU_0 or tILU_0 are used as smoothers for the convection-dominated case. This characteristic of the Jacobi smoother for uni-directional wind was also seen in 2D (see Table 4.2). When Gauss-Seidel smoother is used, the resulting iterative solver is h-robust and exhibits mild dependence on Pe_*. In terms of the execution times, the best performance is seen from the solvers with GMG preconditioned Gauss-Seidel or $\text{tILU}_0(0.5,0.25)$ smoothers.

Lexicographical nodal ordering in a negative x-direction

In Case study 4.2.1 different nodal orderings can have a considerable impact to the performance of some smoothers, and subsequently, the resulting preconditioned iterative solvers (for downwind results see Table 4.4, other nodal ordering results are not included in the thesis).

In Table 5.2 we summarise the iteration counts of Solver Strategy 4.2 with lexicographical ordering of the nodes in a negative x-direction. For Case study 5.1.1 this

Table 5.2: The iteration counts and the (setup,total) execution time (in seconds) required for the convergence of the GMRES solver right-preconditioned by geometric multigrid V(2,2) cycle with several smoothers, when applied to the solution of the discrete convection-diffusion problem (Case study 5.1.1) obtained from $Q1$ SUPG FEM on uniform grids. The lexicographical ordering of the unknowns in the negative x-direction, with each sub-table representing a different diffusion parameter.

(a) Pe_*=500

Smoother \ N	1331	12167	103823	857375
$ILU_0(0.5)$	7(0.89,1.22)	8(7.28,12.0)	8(59.3,108.0)	7(554.5,946.0)
$tILU_0(0.5, 0.25)$	10(0.89,0.96)	11(7.28,8.05)	10(59.4,67.5)	10(545.7,623.4)
$tILU_0(0.5, 0.5)$	10(0.90,0.96)	11(7.27,8.04)	10(59.5,67.7)	11(546.1,622.0)
$Jacobi(0.5)$	16(0.89,0.99)	17(7.13,8.50)	18(58.6,77.8)	17(511.8,656.1)
$GaussSeidel$	8(0.89,0.94)	9(7.24,7.83)	7(59.1,64.8)	6(541.2,589.4)

(b) Pe_*=1,000

Smoother \ N	1331	12167	103823	857375
$ILU_0(0.5)$	7(0.89,1.22)	9(7.25,12.5)	11(59.4,124.2)	9(544.4,1034)
$tILU_0(0.5, 0.25)$	12(0.90,0.98)	13(7.28,8.20)	12(59.5,69.1)	11(548.9,633.9)
$tILU_0(0.5, 0.5)$	12(0.90,0.97)	13(7.27,8.17)	12(59.5,69.2)	11(542.7,627.9)
$Jacobi(0.5)$	18(0.89,1.00)	21(7.13,8.80)	24(58.8,84.1)	21(512.5,689.4)
$GaussSeidel$	10(0.90,0.94)	12(7.25,8.03)	10(59.3,67.2)	8(543.4,605.2)

(c) Pe_*=2,000

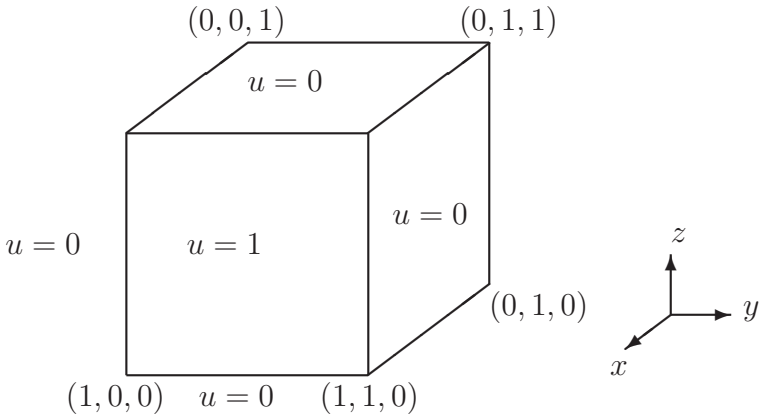
Smoother \ N	1331	12167	103823	857375
$ILU_0(0.5)$	7(0.89,1.22)	10(7.24,13.0)	13(59.5,135.4)	12(548.8,1197)
$tILU_0(0.5, 0.25)$	13(0.90,0.98)	16(7.25,8.36)	15(59.5,71.5)	14(546.4,654.5)
$tILU_0(0.5, 0.5)$	13(0.89,0.97)	16(7.28,8.40)	15(59.5,71.4)	14(551.0,661.1)
$Jacobi(0.5)$	20(0.90,1.01)	29(7.13,9.47)	49(58.8,112.1)	43(518.6,908.4)
$GaussSeidel$	10(0.89,0.94)	15(7.24,8.25)	15(59.5,71.1)	11(564.8,652.8)

corresponds to a downwind nodal ordering. Comparing the iteration counts between Table 5.1 and Table 5.2, we observe that the iteration counts in the latter case are larger (except for the Jacobi smoother that is insensitive to nodal ordering). Moreover, in the latter case we observe that the resulting solvers exhibit more dependence on Pe_*. The largest dependence on Pe_* is observed in the case of the ILU_0 smoother. In the context of execution time, the Gauss-Seidel smoother leads to an iterative solver with the best performance, however for higher values of Pe_*, tILU_0(0.5,0.25) smoother becomes competitive.

5.1.2 Double glazing problem - recirculating wind

We consider a generalisation of the double-glazing problem from Case study 4.2.2, with the convection-diffusion equation (2.7) posed over the unit-cube domain $\Omega = [0,1]^3$ and the convection field consisting of a clockwise circulation along the z-axis. The boundary plane $x = 1$ is regarded as "hot" ($u = 1$), while all other boundaries have zero Dirichlet BCs:

$$\begin{array}{ll} u(x = 1, 0 \le y \le 1, 0 \le z \le 1) = 1 & \text{(hot wall)} \\ u = 0\,, \text{ elsewhere on } \partial\Omega & \text{(cold wall).} \end{array} \tag{5.2}$$



The convection flow used in this case is given by

$$\vec{w} = (2x(1-x)(2y-1)z, -(2x-1)y(1-y), -(2x-1)(2y-1)z(1-z)). \tag{5.3}$$

This is represented as an arrow plot in Figure 5.2(a). In Figure 5.2 we also present the solution u for three different values $Pe_* = 0$, 500 and 10000, with zero source term ($f = 0$). For this case we examine the performance of the MG preconditioner in the context of the multi-directional lexicographical ordering of the nodes as well as the default (tree-based) ordering. In addition, the increase in the number of spacial dimensions inevitably leads to the increase in the computational cost of smoothing with multi-directional lexicographical ordering.

In Table 5.3 we summarise the convergence characteristics of Solver Strategy 4.2 for $Pe_* = 500$ using different orderings for the unknowns. In part(a) of the table we consider default (tree-based) ordering and use MG V(2,2) cycle. We see that all smoothers lead to a h-robust iterative solver, where the smallest iteration counts are observed for ILU_0 and Gauss-Seidel smoothers, and the shortest execution times are obtained when Gauss-Seidel and tILU_0(0.5,0.25) smoothers are used. In part(b) we consider x-directional lexicographical ordering and MG V(6,6) cycle. The reason for this choice of sweeps is that we want to have comparable computational cost of

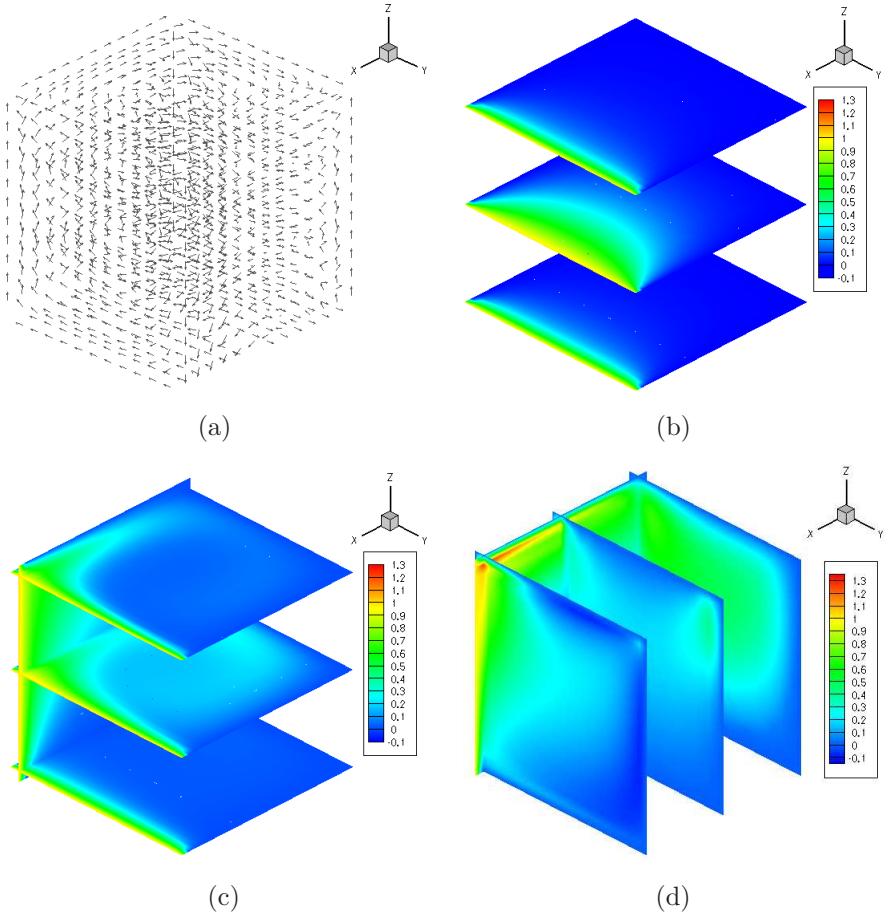


Figure 5.2: The convection-diffusion problem (Case study 5.1.2): recirculating wind (5.3), no source term ($f = 0$); (a) Arrow plot of the wind. (b) the solution for $Pe_* = 0$, (c) the solution for $Pe_* = 500$, (d) the solution for $Pe_* = 10,000$.

the preconditioner with the subsequent cases when multi-directional ordering is used (parts (c) and (d) of the table). We see that this choice of ordering and preconditioner leads to a h-robust solver, irrespectively of the choice of smoother. The shortest execution time is observed for the case when the Gauss-Seidel smoother is used. The tILU$_0$ smoothers lead to roughly 50% shorter execution time than when ILU$_0$ smoother is used. Also, we see, that the GMG V(6,6) cycle as a preconditioner with x-directional lexicographical ordering yields longer execution times than those obtained for GMG V(2,2) cycle as a preconditioner with default ordering.

In part(c) of Table 5.3 we report multi-directional forward lexicographical ordering (xyz-xyz) and MG V(6,6) cycle. We see that only ILU$_0$ smoother has slightly lower iteration counts, compared to the previous case, while all other smoothers show the same efficiency in both cases, with the relative relations between the execution times remaining roughly the same. Part(d) of the table presents the results for 6-directional lexicographical ordering made up of forward and backward sweeps in all

Table 5.3: The iteration counts and the (setup,total) execution time (in seconds) required for the convergence of the GMRES solver right-preconditioned by geometric multigrid with several smoothers, when applied to the solution of the discrete convection-diffusion problem (Case study 5.1.2) obtained from $Q1$ SUPG FEM on uniform grids. A constant diffusion parameter ($Pe_* = 500$), with each sub-table representing a different ordering of the unknowns.

(a) OOMPHLIB natural ordering; MG V-cycle V(2,2)

Smoother \ N	1331	12167	103823	857375
$ILU_0(0.5)$	6(0.89,1.04)	6(7.15,9.10)	6(58.9,80.4)	6(526.3,698.6)
$tILU_0(0.5, 0.25)$	9(0.89,0.94)	10(7.16,7.77)	10(58.9,65.8)	9(515.0,568.4)
$tILU_0(0.5, 0.5)$	12(0.89,0.96)	12(7.16,7.85)	12(59.0,67.0)	11(516.6,581.1)
$Jacobi(0.5)$	15(0.89,0.98)	15(7.14,8.33)	14(58.9,73.9)	13(515.9,627.7)
$GaussSeidel$	6(0.89,0.93)	6(7.14,7.58)	6(58.6,64.8)	5(514.6,556.1)

(b) x-directional ordering ; MG V-cycle V(6,6)

Smoother \ N	1331	12167	103823	857375
$ILU_0(0.5)$	4(0.91,1.46)	4(7.56,14.8)	4(64.5,139.9)	4(596.4,1247.)
$tILU_0(0.5, 0.25)$	5(0.91,0.98)	6(7.57,8.42)	6(64.7,74.0)	6(598.5,683.3)
$tILU_0(0.5, 0.5)$	6(0.91,0.99)	7(7.58,8.48)	6(64.7,73.7)	6(594.2,678.9)
$Jacobi(0.5)$	8(0.90,1.02)	7(7.14,8.54)	7(57.7,76.6)	7(532.8,701.8)
$GaussSeidel$	4(0.91,0.95)	4(7.56,8.01)	4(64.1,71.9)	4(588.1,659.4)

(c) xyz-forward-directional ordering; MG V-cycle V(6,6)

Smoother \ N	1331	12167	103823	857375
$ILU_0(0.5)$	3(0.92,1.36)	3(7.55,13.5)	3(64.4,124.2)	3(598.2,1124)
$tILU_0(0.5, 0.25)$	5(0.91,0.98)	6(7.59,8.46)	6(64.3,73.5)	6(599.9,685.6)
$tILU_0(0.5, 0.5)$	6(0.91,0.99)	7(7.58,8.48)	6(64.4,73.2)	6(598.9,684.6)
$Jacobi(0.5)$	8(0.90,1.02)	7(7.14,8.54)	7(57.7,76.6)	7(532.8,701.8)
$GaussSeidel$	4(0.91,0.96)	4(7.56,8.05)	4(63.9,71.7)	4(591.0,667.0)

(d) xyz-forward-xyz-backward-directional ordering; MG V-cycle V(6,6)

Smoother \ N	1331	12167	103823	857375
$ILU_0(0.5)$	3(0.91,1.36)	3(7.57,13.6)	3(64.6,125.8)	3(597.6,1132)
$tILU_0(0.5, 0.25)$	5(0.91,0.98)	6(7.60,8.47)	6(64.6,73.9)	6(599.8,685.9)
$tILU_0(0.5, 0.5)$	6(0.91,0.99)	7(7.62,8.53)	6(64.7,73.7)	6(595.5,681.2)
$Jacobi(0.5)$	8(0.90,1.02)	7(7.14,8.54)	7(57.7,76.6)	7(532.8,701.8)
$GaussSeidel$	3(0.92,0.95)	4(7.57,8.14)	4(64.1,71.9)	4(590.1,666.9)

(e) "Black-box" ordering (HSL_MC13 Tarjan's algorithm); MG V-cycle V(2,2)

Smoother \ N	1331	12167	103823	857375
$ILU_0(0.5)$	7(0.91,1.24)	7(7.71,12.2)	6(86.5,124.3)	6(2661.0,3015.1)
$tILU_0(0.5, 0.25)$	10(0.91,0.97)	10(7.72,8.41)	10(86.7,93.5)	10(2646.4,2715.0)
$tILU_0(0.5, 0.5)$	12(0.91,0.98)	12(7.71,8.47)	12(86.7,94.5)	11(2602.5,2677.2)
$Jacobi(0.5)$	15(0.89,0.98)	15(7.14,8.33)	14(58.9,73.9)	13(515.9,627.7)
$GaussSeidel$	7(0.90,0.94)	6(7.72,8.11)	6(86.5,91.3)	6(2670.1,2715.8)

three Cartesian directions. Both the iteration counts and the execution times look remarkably similar to the previous case. We see no particular difference in the results reported for three different nodal orderings (part(b), (c), and (d) of the table)). This was also the case for the 2D counterpart of this problem (see Table 4.5).

Finally, in part(e) of Table 5.3 we present "black-box" ordering (where the lookup

vector is constructed from the application of Tarjan's algorithm to the convective part C of the matrix A). As in the 2D case we use the HSL application of Tarjan's algorithm [82] (the routine `HSL_MC13` [1]). Furthermore, we use GMG V(2,2) cycle as a preconditioner. Comparing these results with part(a) of the table, we see a remarkable similarity in terms of the iteration counts. However, a considerable overhead in terms of the execution time, observed in the case of "black-box" ordering, can be attributed to the time taken by calculating a matrix permutation by the HSL routine. This considerable overhead in terms of the execution time is also seen in the 2D case (Table 4.5), however, the iteration counts for all types of smoothers is worse (in 2D) when "black-box" nodal ordering compared to when default ordering is used.

In Table 5.4 we present the convergence results for $Pe_* = 2000$ for the same set of solver parameters. In all cases we observe h-robust preconditioned iterative solvers. In the case of default nodal ordering the most competitive solvers are obtained when Gauss-Seidel and $\text{tILU}_0(0.5,0.25)$ smoothers are used (as was seen for $Pe_* = 500$). The solvers that are Pe-robust are found when ILU_0 and $\text{tILU}_0(0.5,0.25)$ are used as smoothers. When smoothers based on directional or multi-directional lexicographical ordering of the nodes are used (part(b), (c) and (d) of Table 5.4), the resulting iterative solvers with ILU_0, $\text{tILU}_0(0.5,0.25)$ and Gauss-Seidel smoothers perform Pe, h-robust. As in the case $Pe_* = 500$, the use of "black-box" ordering based on Tarjan's algorithm does not produce any improvement in terms of the iteration counts, when compared to default ordering.

Truncation statistics

When discretisation of the convection-diffusion problem (3.1) is performed on uniformly refined grids, the resulting coefficient matrices A have the same number of non-zero entries, irrespectively of the problem. Numerical values of these non-zero entries, however, depend on the convective field $\vec{w}$ used in a particular problem, as well as on Pe_*. These two parameters have a crucial effect on the non-zero pattern of truncated matrix $\widetilde{A}$, obtained using (3.43).

With each further level of uniform grid refinement in 3D the discrete problem size increases by a factor of 8 (asymptotically). Moreover, the connectivity pattern in Q_1 approximation gives a stencil size (number of non-zero per row) of 27. This implies that in the case of the Jacobi smoothing the truncation ratio should be approximately $\frac{NNZ(\widetilde{A})}{NNZ(A)} = \frac{1}{27} \approx 0.04$. A further difference between 2D and 3D matrices is the matrix stencil. In 2D the entries of the diffusion matrix D are constant, i.e. behave as $O(1)$ (see Section 2.4), while in 3D they behave as $O(h)$. Thus, we can expect that the application of the truncation (3.43) to result in removing a different amount of non-zero entries than in the 2D cases.

Table 5.4: The iteration counts and the (setup,total) execution time (in seconds) required for the convergence of the GMRES solver right-preconditioned by geometric multigrid with several smoothers, when applied to the solution of the discrete convection-diffusion problem (Case study 5.1.2) obtained from $Q1$ SUPG FEM on uniform grids. A constant diffusion parameter ($Pe_* = 2000$), with each sub-table representing a different ordering of the unknowns.

(a) OOMPHLIB natural ordering; MG V-cycle V(2,2)

Smoother \ N	1331	12167	103823	857375
$ILU_0(0.5)$	8(0.89,1.08)	9(7.14,9.93)	8(58.8,86.4)	8(555.5,799.3)
$tILU_0(0.5, 0.25)$	13(0.89,0.97)	14(7.16.8.04)	13(58.9,68.2)	13(554.5,644.1)
$tILU_0(0.5, 0.5)$	18(0.90,0.99)	20(7.15,8.32)	19(59.0,71.8)	18(548.5,670.4)
$Jacobi(0.5)$	31(0.89,1.08)	34(7.14,10.1)	31(58.9,92.2)	29(546.5,847.8)
$GaussSeidel$	10(0.89,0.94)	11(7.14,7.88)	10(58.9,68.7)	9(553.3,636.9)

(b) x-directional ordering ; MG V-cycle V(6,6)

Smoother \ N	1331	12167	103823	857375
$ILU_0(0.5)$	5(0.91,1.56)	6(7.61,17.9)	6(64.7,170.6)	5(673.5,1546.)
$tILU_0(0.5, 0.25)$	7(0.91,1.02)	8(7.59,8.82)	7(67.0,80.7)	7(665.3,792.6)
$tILU_0(0.5, 0.5)$	9(0.91,1.02)	10(7.59,8.89)	9(66.8,81.5)	9(660.8,803.2)
$Jacobi(0.5)$	15(0.90,1.11)	15(7.15,10.1)	13(57.9,90.6)	12(531.6,809.5)
$GaussSeidel$	6(0.91,0.96)	6(7.60,8.35)	6(64.4,75.5)	5(645.8,751.1)

(c) xyz-forward-directional ordering; MG V-cycle V(6,6)

Smoother \ N	1331	12167	103823	857375
$ILU_0(0.5)$	5(0.91,1.57)	5(7.55,16.3)	5(64.2,153.6)	4(653.3,1381)
$tILU_0(0.5, 0.25)$	7(0.92,1.02)	7(7.58,8.68)	7(64.3,76.7)	7(686.4,818.8)
$tILU_0(0.5, 0.5)$	9(0.91,1.03)	10(7.57,8.89)	9(64.3,77.6)	9(698.7,865.0)
$Jacobi(0.5)$	15(0.90,1.11)	15(7.15,10.1)	13(57.9,90.6)	12(531.6,809.5)
$GaussSeidel$	6(0.91,0.96)	6(7.57,8.37)	6(64.1,74.9)	5(638.3,748.7)

(d) xyz-forward-xyz-backward-directional ordering; MG V-cycle V(6,6)

Smoother \ N	1331	12167	103823	857375
$ILU_0(0.5)$	5(0.91,1.59)	5(7.60,16.7)	5(64.4,155.9)	4(625.4,1318)
$tILU_0(0.5, 0.25)$	7(0.92,1.03)	7(7.59,8.71)	7(64.6,77.3)	7(608.9,718.9)
$tILU_0(0.5, 0.5)$	9(0.91,1.02)	10(7.59,8.92)	9(64.6,78.3)	9(597.3,722.1)
$Jacobi(0.5)$	15(0.90,1.11)	15(7.15,10.1)	13(57.9,90.6)	12(531.6,809.5)
$GaussSeidel$	6(0.91,0.96)	6(7.60,8.39)	5(64.6,73.9)	5(628.2,728.1)

(e) "Black-box" ordering (HSL_MC13 Tarjan's algorithm); MG V-cycle V(2,2)

Smoother \ N	1331	12167	103823	857375
$ILU_0(0.5)$	10(0.91,1.36)	11(7.72,14.4)	10(86.6,146.0)	9(2650.7,3167.6)
$tILU_0(0.5, 0.25)$	14(0.91,1.01)	15(7.72,8.82)	15(86.7,98.0)	14(2633.5,2741.2)
$tILU_0(0.5, 0.5)$	19(0.91,1.02)	21(7.73,9.11)	21(86.7,100.7)	19(2664.9,2797.9)
$Jacobi(0.5)$	31(0.89,1.08)	34(7.14,10.1)	31(58.9,92.2)	29(546.5,847.8)
$GaussSeidel$	12(0.91,0.97)	13(7.74,8.53)	11(86.5,94.7)	10(2679.8,2765.9)

In Table 5.5 we report the truncation statistics obtained for the Case study 5.1.2 discretised on a sequence of uniformly refined grids. For the mildly-convective problem ($Pe_* = 20$), we see no difference in the truncation rates between $\alpha = 0.25$ and 0.5. In both cases tILU_0 smoothing is reduced to the Jacobi smoother. For $Pe_* = 500$, tILU_0 smoothing differs only marginally from Jacobi (mainly at the coarsest levels),

Table 5.5: Truncation statistics for the convection-diffusion operator obtained from Q1 SUPG FEM discretisation of Case study 5.1.2 as a function of Pe_* with uniform grid refinement. $\widetilde{A}_l$ denotes the truncated matrix and A_l is the original matrix at the refinement level l. NNZ$(\cdot)$ denotes the number of non-zero entries and $\frac{NNZ(\widetilde{A}_l)}{NNZ(A_l)}[\%]$ the percentage of the non-zero entries kept in the truncated matrices.

(a) Pe_*=20

Level l	6	5	4	3	2	1	$\sum_l$(NNZ($\widetilde{A}_l$))
N	857375	103823	12167	1331	125	8	974829
NNZ(A_l)	22665187	2685619	300763	29791	2197	64	25683621
$\alpha = 0$							
NNZ($\widetilde{A}_l$)	22665187	2685619	300763	29791	2197	64	25683621
$\frac{NNZ(\widetilde{A}_l)}{NNZ(A_l)}[\%]$	100	100	100	100	100	100	η =100
$\alpha = 0.25$							
NNZ($\widetilde{A}$)	857375	103823	12167	1331	125	8	974829
$\frac{NNZ(\widetilde{A}_l)}{NNZ(A_l)}[\%]$	4	4	4	4	6	13	η =4
$\alpha = 0.5$							
NNZ($\widetilde{A}_l$)	857375	103823	12167	1331	125	8	974829
$\frac{NNZ(\widetilde{A}_l)}{NNZ(A_l)}[\%]$	4	4	4	4	6	13	η =4
$\alpha = 1.0$							
NNZ($\widetilde{A}_l$)	857375	103823	12167	1331	125	8	974829
$\frac{NNZ(\widetilde{A}_l)}{NNZ(A_l)}[\%]$	4	4	4	4	6	13	η =4

(b) Pe_*=500

Level l	6	5	4	3	2	1	$\sum_l$(NNZ($\widetilde{A}_l$))
$\alpha = 0.25$							
NNZ($\widetilde{A}_l$)	860153	118167	18527	2577	305	16	999745
$\frac{NNZ(\widetilde{A}_l)}{NNZ(A_l)}[\%]$	4	4	6	9	14	25	η =4
$\alpha = 0.5$							
NNZ($\widetilde{A}_l$)	857375	103823	12525	1633	173	14	975543
$\frac{NNZ(\widetilde{A}_l)}{NNZ(A_l)}[\%]$	4	4	4	5	8	22	η =4

(c) Pe_*=2000

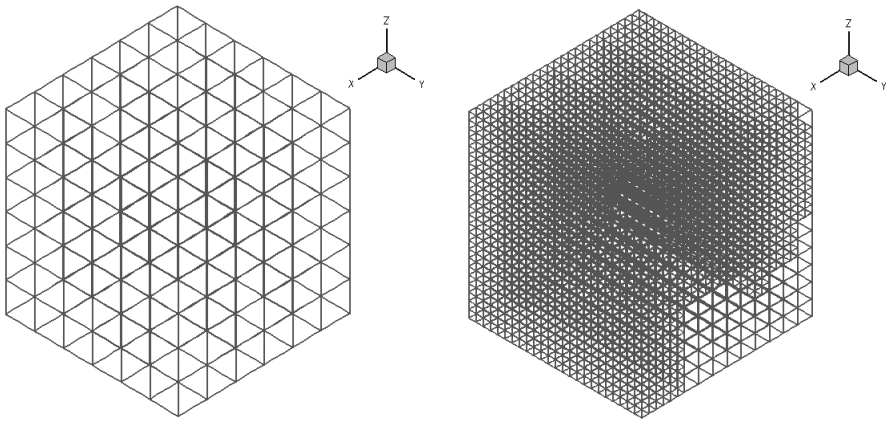
Level l	6	5	4	3	2	1	$\sum_l$(NNZ($\widetilde{A}_l$))
$\alpha = 0.25$							
NNZ($\widetilde{A}_l$)	1317847	207143	29569	3991	429	18	1558997
$\frac{NNZ(\widetilde{A}_l)}{NNZ(A_l)}[\%]$	6	8	10	13	20	28	η =6
$\alpha = 0.5$							
NNZ($\widetilde{A}_l$)	886947	130823	17947	2101	205	16	1038039
$\frac{NNZ(\widetilde{A}_l)}{NNZ(A_l)}[\%]$	4	5	6	7	9	25	η =4

but leads to a significant improvement in the iteration counts (by 31% - see Table 5.3(a)). For the highly convective case ($Pe_* = 2000$), a truncation of $\alpha = 0.25$ gives, on average, matrices with only 60% more entries than in the Jacobi case and the truncation with $\alpha = 0.5$ yields the matrices with only 6% more non-zero entries than the Jacobi case. This relatively small increase in smoother complexity leads to the reduction in iteration counts by 55% and 38%, respectively (see Table 5.4(a)).

The level of truncation, that still produces a satisfactory iteration count, at the finest mesh is staggering with over 21 million fewer non-zero entries. The overall total saving in storage costs is between $94 - 96\%$ when compared to using standard $\text{ILU}_0(0.5)$ as a smoother. At the few finest levels (where the discrete operators are the largest), the tILU_0 smoother resembles (or is equal to) the Jacobi smoother, but gives a much improved performance in comparison to the Jacobi smoother.

Adaptively refined grids

A more sophisticated form of grid refinement is to use adaptive mesh refinement (see Pages 123, 124 for further details). In Figure 5.3(a) a single uniform refinement has been taken from the original coarse grid structure. In Figure 5.3(b) adaptive mesh



(a) (b)

Figure 5.3: Mesh plot of Case study 5.1.2 using (a) uniform refinement with $N = 125$, (b) adaptive mesh refinement after 3 levels of refinement with $N = 8208$.

refinement is presented for Case study 5.1.2, with $Pe_* = 2000$. In this particular example, after three levels of refinement, a large proportion of the domain is refined uniformly. A consequence of the problem discretisation on adaptively refined grids is the introduction of an imbalance in the magnitude of the matrix coefficients. In searching for a general smoother it is important to investigate such a case and whether the smoother is robust.

In Table 5.6 we present the convergence characteristics of Solver Strategy 4.2 for Case study 5.1.2, discretised on a sequence of adaptively refined grids. In all cases we see that the GMG preconditioner performs h-robust and has only minor deterioration with respect to Pe_*. In terms of the execution time, the best results are observed when Gauss-Seidel or tILU_0 smoothers are used.

In Table 5.6(b) we find that the total solve times at the refinement steps (5) and (5b) are very similar. In Table 5.7(b) the number of refinement levels is identical to

Table 5.6: The iteration counts and the (setup,total) execution time (in seconds) required for the convergence of the GMRES solver right-preconditioned by geometric multigrid V(2,2) cycle with several smoothers, when applied to the solution of the discrete convection-diffusion problem (Case study 5.1.2) obtained by the Q1 SUPG FEM on an adaptive grid refinement. Natural (tree-based) ordering of the unknowns, with each sub-table representing a different diffusion parameter.

(a) Pe_*=500

Smoother \ Level	2	3	4	5
$ILU_0(0.5)$	5(0.90,1.04)	6(5.25,6.39)	6(15.9,19.9)	5(28.3,33.8)
$tILU_0(0.5, 0.25)$	9(0.92,0.97)	8(5.29,5.58)	8(15.8,16.8)	8(28.2,29.7)
$tILU_0(0.5, 0.5)$	11(0.92,0.98)	11(5.28,5.65)	10(15.9,17.1)	10(28.2,30.1)
$Jacobi(0.5)$	15(0.91,1.00)	13(5.27,5.83)	12(15.9,18.2)	11(28.2,31.6)
$GaussSeidel$	6(0.91,0.94)	5(5.26,5.45)	5(15.9,16.8)	5(28.3,29.7)

(b) Pe_*=2,000

Smoother \ Level	2	3	4	5	5b
$ILU_0(0.5)$	7(0.92,1.10)	8(5.87,7.57)	7(19.2,24.4)	7(46.6,59.1)	5(47.2,56.5)
$tILU_0(0.5, 0.25)$	12(0.94,1.01)	12(5.90,6.43)	11(19.2,20.7)	11(46.7,50.5)	9(47.3,50.5)
$tILU_0(0.5, 0.5)$	16(0.93,1.01)	17(5.86,6.54)	15(19.2,21.3)	15(46.6,51.6)	11(47.2,50.9)
$Jacobi(0.5)$	30(0.93,1.11)	31(5.87,7.33)	26(19.2,24.7)	24(46.5,58.7)	18(47.2,56.3)
$GaussSeidel$	10(0.89,0.95)	10(5.77,6.18)	9(19.1,20.8)	8(46.5,50.3)	6(47.1,50.0)

the number of refinement steps in Table 5.6(a). However in Table 5.6(b) we apply an extra refinement step, as the maximum number of refinement steps or the error tolerance has not been reached. A hierarchical refinement level is defined as an increase in the depth of a tree structure, where the original coarse mesh is the root of the tree. In Table 5.6(b) refinement (5b) is an example of when there is a decrease in the number of nodes in the mesh (i.e. some element patches are "un-refined"), however the depth of the tree structure remains the same. Therefore the number of hierarchical levels remains unchanged, but the number of adaptive refinement steps has increased. This means that the problem obtained in 5.6(b) has the same number of hierarchical levels as the one in 5.6(a).

Table 5.7 summaries the truncation statistics for a sequence of coarse grid levels generated by GMG for the case of five adaptive refinements. The amount of elements refined at each new level of the adaptive process is problem specific. That is, as Pe_* increases the size of the coefficient matrix may also increase. Furthermore, as Pe_* increases the magnitude of the off-diagonal entries becomes more dominant. This results in a smaller level of truncation. From part(a) of the table we see that in a mildly convective case ($Pe_* = 20$) the tILU$_0$ smoother reduces to a Jacobi smoother ($NNZ(\widetilde{A})$=N). When Pe_* is increased to 500, the cost of tILU$_0$(0.5,0.25) is only marginally higher than that of Jacobi, while for $\alpha = 0.5$ this difference is almost indistinguishable. For a highly convective case (part c), $Pe_* = 2000$ the total number of non-zero entries kept in all coarse matrices for $\alpha = 0.25$ is roughly twice that of the Jacobi smoother, while in the case $\alpha = 0.5$ the difference is again almost

Table 5.7: Truncation statistics for the convection-diffusion operator obtained from Q1 SUPG FEM discretisation of Case study 5.1.2 as a function of Pe_* with adaptive grid refinement. $\widetilde{A_l}$ denotes the truncated matrix and A_l is the original matrix at the refinement level l. NNZ$(\cdot)$ denotes the number of non-zero entries and $\frac{NNZ(\widetilde{A_l})}{NNZ(A_l)}[\%]$ the percentage of the non-zero entries kept in the truncation matrix.

(a) Pe_*=20

Level l	6	5	4	3	2	1	$\sum_l$(NNZ($\widetilde{A_l}$))
N	17,734	2,829	215	64	24	8	
NNZ(A_l)	434,534	60,187	2,459	480	176	64	497,900
$\alpha = 0$							
NNZ($\widetilde{A_l}$)	434,534	60,187	2,459	480	176	64	497,900
$\frac{NNZ(\widetilde{A_l})}{NNZ(A_l)}[\%]$	100	100	100	100	100	100	η =100
$\alpha = 0.25$							
NNZ($\widetilde{A_l}$)	17,734	2,829	215	64	24	8	20,874
$\frac{NNZ(\widetilde{A_l})}{NNZ(A_l)}[\%]$	4	5	9	13	14	13	η =4
$\alpha = 0.5$							
NNZ($\widetilde{A_l}$)	17,734	2,829	215	64	24	8	20,874
$\frac{NNZ(\widetilde{A_l})}{NNZ(A_l)}[\%]$	4	5	9	13	14	13	η =4
$\alpha = 1.0$							
NNZ($\widetilde{A_l}$)	17,734	2,829	215	64	24	8	20,874
$\frac{NNZ(\widetilde{A_l})}{NNZ(A_l)}[\%]$	4	5	9	13	14	13	η =4

(b) Pe_*=500

Level l	6	5	4	3	2	1	$\sum_l$(NNZ($\widetilde{A_l}$))
Size(A_l)	31,472	5,021	507	109	24	8	37,141
NNZ(A_l)	792,372	113,647	8,149	1,243	176	64	915,651
$\alpha = 0.25$							
NNZ($\widetilde{A_l}$)	39,271	7,659	1,018	234	44	16	48,242
$\frac{NNZ(\widetilde{A_l})}{NNZ(A_l)}[\%]$	5	7	12	19	25	25	η =5
$\alpha = 0.5$							
NNZ($\widetilde{A_l}$)	31,570	5,333	624	144	35	14	37,720
$\frac{NNZ(\widetilde{A_l})}{NNZ(A_l)}[\%]$	4	5	8	12	20	22	η =4

(c) Pe_*=2000

Level l	6	5	4	3	2	1	$\sum_l$(NNZ($\widetilde{A_l}$))
Size(A_l)	56,116	8,612	953	202	51	8	65,942
NNZ(A_l)	1,419,498	198,894	17,171	2,924	629	64	1,639,180
$\alpha = 0.25$							
NNZ($\widetilde{A_l}$)	115,672	21,623	2,586	530	143	18	140,572
$\frac{NNZ(\widetilde{A_l})}{NNZ(A_l)}[\%]$	8	11	15	18	23	28	η =9
$\alpha = 0.5$							
NNZ($\widetilde{A_l}$)	70234	13263	1576	331	82	16	85,502
$\frac{NNZ(\widetilde{A_l})}{NNZ(A_l)}[\%]$	4	4	6	7	8	13	η =5

indistinguishable. Nevertheless, the smoother tILU$_0$(0.5,0.5) leads to a considerable reduction in iteration counts, compared to the Jacobi smoother (see Table 5.6(b)).

Comparing the uniform mesh refinement truncation ratio η (in Table 5.5(c)) with

those obtained from adaptive mesh refinement (Table 5.7(c)) for Case study 5.1.2 we observe that adaptive mesh refinement leads to a larger truncation ratio η (fewer entries are truncated in adaptive mesh refinement). This is partially due to the need for smaller meshing found in adaptive mesh refinement allowing the coarse level refinement to have a larger contribution towards the overall total number of entries. That is, the coarse level refinement is proportionately closer to finer level in adaptive mesh refinement than in uniform refinement. The other reason for a large total truncation percentile is that adaptive refinement has been able to throw away less relevant areas of the domain, that our truncation method would also have disposed of if they were adequately small.

In Table 5.8 we present a comparative study of the total truncation ratios between Case studies 5.1.1 and 5.1.2 for the case of adaptively refined grids. The most

Table 5.8: Comparison of truncation statistics for Case studies 5.1.1 and 5.1.2 as a function of Pe_* and the truncation parameter α. The discrete convection-diffusion operators are obtained from Q1 SUPG FEM discretisation using adaptive meshing. The results present the total number of non-zero entries in the entire MG hierachy and, in brackets, the percentage of the total number of non-zero entries in the entire MG hierarchy after truncation.

Pe_*	Case Study	$\alpha = 0$	$\alpha = 0.25$	$\alpha = 0.5$	$\alpha = 1.0$
500	5.1.1	2,352,205(100)	213,997(9)	149,631(6)	96,501(4)
	5.1.2	915,651(100)	48,242(5)	37,720(4)	37,141(4)
2000	5.1.1	3,306,585(100)	303,521(9)	271,081(8)	135,649(4)
	5.1.2	1,639,180(100)	140,572(9)	85,502(5)	65,942(4)

important conclusion is that, comparing with the 2D results reported in Table 4.14, the amount of truncation in 3D is larger (smaller η) for the same value of α. In each case the discrete problem is adapted to either a maximum level of refinement or the default a posteriori error is reached.

5.1.3 Summary

In Section 4.2 we tested the performance of the GMRES solver preconditioned by GMG, with four different smoothing strategies. The truncation parameter that showed the best results in 2D for the tILU_0 smoother was $\alpha = 0.25$. In this section we tested the GMRES solver preconditioned by GMG with the same four different smoothers in the context discrete convection-diffusion problems in 3D. We observed that the iteration counts obtained with $\text{tILU}_0(0.5, 0.25)$ smoother are asymptotically optimal with respect to the discrete problem size and exhibited only a moderate dependence on Pe_*. This is consistent with our findings from Section 4.2. When truncation of the coefficient matrices in the MG hierarchy is concerned, a considerably large proportion of the non-zero off-diagonal entries are truncated in 3D discrete problems compared to the 2D case.

In addition, the study of different damping parameters for Jacobi/tILU$_0$/ILU$_0$ smoothing reveals that damped Jacobi and ILU$_0$ smoothing performs in a very similar fashion to Tables 4.15 and 4.16 when GMG preconditioner is used for 3D problems. When tILU$_0(\gamma, 0.25)$ smoothing is used, the best GMRES iteration counts are observed for $0.5 \leq \gamma \leq \frac{2}{3}$.

5.2 Algebraic multigrid preconditioning of the convection-diffusion problem

In this section we report the performance of the classical AMG preconditioner with four different smoothing strategies, when applied to Case studies 5.1.1 and 5.1.2.

AMG performs a variant of semi-coarsening in the characteristic directions, governed by the strength and directions of the wind [8]. This leads to an increase in the number of coarse levels (compared to GMG). Also, with the substantial increase in the stencil size of the original fine level matrix in 3D, and the fact that AMG is computationally more expensive per iteration than GMG, due to an increase in the coarse level matrices stencils in comparison to the original fine level matrix stencils, having a "cheap", robust smoother is essential (see Section 3.3.2 and Pages 132–134 for more details).

In Section 4.3 we demonstrated that the best results in 2D for the AMG preconditioner with tILU$_0$ smoothing are observed for the truncation parameter $\alpha = 0.5$. The aim here is to demonstrate that this combination works equally well for 3D problems.

In Table 5.9 we summarise the convergence results of Solver Strategy 4.3 for Case study 5.1.1. In the case of AMG, as was the case in 2D, we extend the range of Pe_* to include $Pe_* = 500$, 2000 and 10000. The discretisation of the problem is performed on a sequence of uniform grids. For a moderate value of Pe_* ($Pe_* = 500$) we see that an AMG preconditioner is h-robust with all smoothers considered. As Pe_* is increased Gauss-Seidel as a smoother exhibits a considerable deterioration in its performance, (the iteration counts behave with $O(\sqrt{Pe_*})$). In all cases, the ILU$_0$ and tILU$_0(0.5, 0.5)$ smoothers are Pe, h-robust and ILU$_0$ is observed to have the smallest number of iterations. However, the execution time of ILU$_0$ is not competitive, due to the excessive computational cost associated with ILU$_0$ factorisation in 3D. In Table 5.9(c), the shortest execution time is obtained when tILU$_0(0.5, 0.5)$ is used as a smoother (a large truncation resulting in only a small increase in the iteration count in comparison to ILU$_0$ smoother).

We report in Table 5.10 the convergence results of Solver Strategy 4.3 for Case study 5.1.2 where a more complex convection field is used, recirculating wind. It

Table 5.9: The iteration counts and the (setup,total) execution time (in seconds) required for the convergence of the GMRES solver right-preconditioned by classical algebraic multigrid V(2,2) cycle with several smoothers, when applied to the solution of the discrete convection-diffusion problem (Case study 5.1.1) obtained from Q1 SUPG FEM on uniform grids. Each sub-table represents a different diffusion parameter.

(a) Pe_*=500

Smoother \ N	1331	12167	103823	857375
$ILU_0(0.5)$	4(0.08,0.12)	5(1.99,4.32)	5(33.6,63.6)	6(1112.9,1660.2)
$tILU_0(0.5, 0.25)$	7(0.05,0.09)	7(0.98,2.58)	7(14.1,36.6)	9(413.4,849.0)
$tILU_0(0.5, 0.5)$	7(0.05,0.09)	7(0.98,2.51)	8(14.0,39.3)	10(413.5,891.8)
$Jacobi(0.5)$	11(0.03,0.09)	9(0.76,2.74)	7(11.9,36.7)	7(390.4,765.2)
$GaussSeidel$	7(0.03,0.07)	7(0.75,2.30)	7(11.9,36.3)	5(392.5,692.9)

(b) Pe_*=2,000

Smoother \ N	1331	12167	103823	857375
$ILU_0(0.5)$	4(0.08,0.12)	4(1.67,3.37)	5(30.8,60.0)	6(855.4,1337.8)
$tILU_0(0.5, 0.25)$	10(0.05,0.11)	9(0.84,2.66)	7(13.9,35.9)	7(310.8,609.2)
$tILU_0(0.5, 0.5)$	10(0.05,0.10)	8(0.84,2.36)	7(13.9,35.9)	8(311.0,646.5)
$Jacobi(0.5)$	16(0.03,0.12)	17(0.62,3.87)	12(11.8,51.2)	8(288.1,640.5)
$GaussSeidel$	10(0.03,0.09)	12(0.62,2.97)	12(11.8,50.6)	12(289.4,795.9)

(c) Pe_*=10,000

Smoother \ N	1331	12167	103823	857375
$ILU_0(0.5)$	4(0.08,0.12)	4(1.57,3.29)	4(31.1,54.0)	5(659.0,1026.6)
$tILU_0(0.5, 0.25)$	3(0.19,0.25)	14(0.78,3.57)	39(13.9,122.2)	17(254.0,889.1)
$tILU_0(0.5, 0.5)$	12(0.05,0.11)	13(0.77,3.24)	13(13.8,50.5)	10(253.9,640.1)
$Jacobi(0.5)$	17(0.03,0.13)	25(0.56,5.17)	30(11.7,102.3)	20(233.4,1032.7)
$GaussSeidel$	11(0.03,0.09)	17(0.56,3.78)	23(11.8,82.4)	24(231.4,1101.3)

can be seen that all smoothers are h-robust. Furthermore, the results show that $\text{tILU}_0(0.5, 0.5)$ performs consistently for both cases (Case studies 5.1.1 and 5.1.2). A more surprising result, is that Gauss-Seidel and Jacobi as smoothers perform better in the case of a more complex wind. A similar pattern was observed for the Gauss-Seidel smoother in 2D for Case study 4.2.4 (see Table 4.21). However, in the 2D counterpart of Case study 5.1.2 the Gauss-Seidel smoother performed poorly, which does raise some questions about using Gauss-Seidel as a default smoother for AMG preconditioning for convection-dominated problems.

An inherent property of the classical AMG coarsening algorithm is that it generates coarse grid operators that are much denser than the original coefficient matrix and this feature is particularly profound for 3D problems. To reduce this increase in the operator complexities one can increase the value of the strength of dependence threshold θ (3.34) from its default value 0.25 and/or applying a single pass of Ruge-Stüben coarsening [8]. This leads to a deterioration in the numerical efficiency of the preconditioner, however, much shorter execution times are observed (due to a significant reduction in computational cost of assembling and applying the preconditioner).

Table 5.10: The iteration counts and the (setup,total) execution time (in seconds) required for the convergence of the GMRES solver right-preconditioned by classical algebraic multigrid V(2,2) cycle with several smoothers, when applied to the solution of the discrete convection-diffusion problem (Case study 5.1.2) obtained from Q1 SUPG FEM on uniform grids. Each sub-table represents a different diffusion parameter.

(a) Pe_*=500

Smoother \ N	1331	12167	103823	857375
$ILU_0(0.5)$	4(0.27,0.46)	5(16.5,27.04)	5(190.5,322.0)	5(7137.9,8580.3)
$tILU_0(0.5, 0.25)$	7(0.10,0.23)	7(2.50,9.89)	8(43.88,177.6)	7(520.5,1702.5)
$tILU_0(0.5, 0.5)$	7(0.11,0.23)	7(2.49,9.88)	8(43.8,177.4)	7(508.8,1649.1)
$Jacobi(0.5)$	7(0.058,0.20)	7(1.92,9.64)	7(37.6,167.4)	6(442.2,1440.2)
$GaussSeidel$	4(0.06,0.14)	4(1.92,6.73)	5(37.6,132.9)	4(463.3,1253.8)

(b) Pe_*=2,000

Smoother \ N	1331	12167	103823	857375
$ILU_0(0.5)$	5(0.23,0.44)	5(7.48,15.0)	5(45.0,136.1)	5(18627.7,20353.)
$tILU_0(0.5, 0.25)$	7(0.09,0.20)	8(1.61,7.36)	8(30.9,141.8)	8(490.2,1860.3)
$tILU_0(0.5, 0.5)$	9(0.09,0.24)	9(1.60,7.93)	9(30.8,153.4)	9(494.6,2028.1)
$Jacobi(0.5)$	9(0.05,0.21)	10(1.10,8.35)	9(25.0,152.2)	8(429.3,1896.4)
$GaussSeidel$	6(0.05,0.16)	6(1.10,5.63)	7(25.1,129.6)	6(423.8,1511.6)

(c) Pe_*=10,000

Smoother \ N	1331	12167	103823	857375
$ILU_0(0.5)$	7(0.20,0.45)	7(5.39,13.8)	7(229.1,366.3)	7(6490.1,8015.6)
$tILU_0(0.5, 0.25)$	9(0.08,0.22)	11(1.33,7.67)	10(20.9,124.3)	10(255.3,1382.0)
$tILU_0(0.5, 0.5)$	10(0.08,0.23)	12(1.31,8.17)	11(20.9,133.2)	11(254.6,1473.6)
$Jacobi(0.5)$	11(0.04,0.21)	15(0.86,9.44)	16(16.0,194.0)	15(206.3,1945.9)
$GaussSeidel$	8(0.04,0.17)	10(0.85,6.59)	12(15.9,145.0)	12(204.8,1586.7)

For the purpose of the project we only use a two-pass Ruge-Stüben coarsening (classical AMG) with a default value for the parameter $\theta = 0.25$. In this context we present in Table 5.11 the complexity measures C_G, C_A and C_S obtained from coarsening of the largest discrete operator ($N = 857,375$) from Case study 5.1.2 as a function of Pe_*.

Table 5.11: Coarsening statistics for Case study 5.1.2 with $N = 857,375$ as a function of Pe_*.

Pe_*	20	500	1,000	5,000	10,000
L	17	20	20	22	25
C_G	2.23259	2.57160	2.82280	3.21759	3.25889
C_A	13.3185	37.5398	41.8640	32.3085	27.35226
C_S	778.997	1250.18	1320.47	935.313	742.447
$C_S^{(1)}$	26.4	26.4	26.4	26.4	26.4

L is the number of MG levels, C_G is the grid complexity, C_A is the operator complexity, C_S is the average matrix stencil over all MG levels and $C_S^{(1)}$ is the average matrix stencil of A at the finest level.

In the case of full coarsening of uniformly refined girds (characterised by a reduction in the number of grid points in each spacial dimension by a factor of 2), we expect $C_G = \frac{8}{7} \simeq 1.143$ in 3D (see Pages 77–79 for further details) [10, p.154] [8]. It is clear from Table 5.11 that AMG coarsening results in a larger grid complexity than GMG. In the 2D case (Table 4.22), grid complexity increases when Pe_* is increased. This can be explained by a more pronounced semi-coarsening that classical AMG performs in the direction of the convection. For the 3D case, there is an increase in grid complexity by 31%, between $Pe_* = 20$ and $Pe_* = 10,000$. By comparison, for a 2D problem, the results from Table 4.22 show a 29% increase. This implies that roughly the same increase in grid complexity with respect to the inverse diffusion parameter is observed in two and three spacial dimensions. However, for a fixed Pe_* there is an increase of between 17% and 28% in the grid complexity of AMG between the 2D and 3D cases.

In Table 5.11 we see that the operator complexity is significantly larger than in 2D (Table 4.22). Moreover, the operator complexity initially grows with Pe_* (up to 1000), and falls when Pe_* is increased further. This also implies that the interpolation matrices are fairly dense. Furthermore, in Table 5.11, there are large differences between the average stencil size over all levels C_S and the finest level $C_S^{(1)}$. This indicates a considerable increase in stencil sizes at coarse levels. A further observation is that C_S peaks at $Pe_* = 1000$, and is reduced significantly for larger values of Pe_*. This statistic may explain the high setup and total execution times of the iterative solver with ILU_0 smoother reported in Table 5.10. In cases when the physical memory is smaller than the memory required, data is moved to and from the hard disk with continuous swapping to perform a single task. This is known as memory page swapping. Page swapping usually leads to a poor execution performance. Furthermore, the large memory requirement of AMG with ILU_0 smoother may lead to poor caching, and further deterioration in execution.

In the case of 2D problems, Figure 4.10, the average amount of non-zero entries kept in the matrices MG hierarchy was below 20% for $\alpha = 0.25$ and around 11% for $\alpha = 0.5$. For three-dimensional problems, the grid complexity is larger than in two dimensions. Furthermore the stencil size also increases, increasing the density of the matrix. These two properties, together with a different magnitude to the matrix entries in 3D have the potential to have their storage costs reduced considerably. In Table 5.12 we calculate the total truncation ratio (3.40) for Case study 5.1.2. In the case of classical AMG coarsening, the amount of truncation is almost 99.5% of the total number of non-zeros, and this amount almost corresponds to the Jacobi case. Furthermore, the truncation ratio does not depend very strongly on Pe_*. Also, such a large percentage of truncated entries confirms that AMG's coarse level matrices

Table 5.12: Truncation ratio (3.40) for several different values of truncation parameter α as a function of Pe_*, for Case study 5.1.2 with $N = 857,375$.

Pe_*	20	500	1,000	5,000	10,000
$\alpha = 0.0$	1	1	1	1	1
$\alpha = 0.25$	0.00634147	0.002606405	0.00272057	0.006681074	0.00998486
$\alpha = 0.5$	0.006341109	0.002592616	0.002552394	0.004223302	0.00552579
$\alpha = 1.0$	0.006341106	0.002591332	0.002550646	0.003767262	0.00450701

have much larger matrix stencils than in the case of GMG full coarsening (in 3D the maximal truncation ratio in GMG corresponding to the Jacobi case, is $\sim 96\%$).

In Table 5.13 we expand the operator complexity results from Table 5.11 to allow further analysis of the operator complexity as a function of the truncation parameter α and Pe_*. When truncation (3.43) is applied to the coarse level operator, we

Table 5.13: Operator complexity, C_A, (3.38) for several different values of truncation parameter α as a function of Pe_*, for Case study 5.1.2 with $N = 857,375$.

Pe_*	20	500	1,000	5,000	10,000
$\alpha = 0.0$	13.31847829	37.53981368	41.86409775	32.30846862	27.35225595
$\alpha = 0.25$	2.232713807	2.578206435	2.631932896	2.652096866	2.84415412
$\alpha = 0.5$	2.232586674	2.572875346	2.824735967	2.672281753	2.59363991
$\alpha = 1.0$	2.232585508	2.571601691	2.822802158	3.217589736	3.25889372

have much lower operator complexities, which are virtually independent of both the truncation ratio and Pe_*. This implies that the cost of applying the smoother in cases $\alpha > 0$ will be much smaller than for $\alpha = 0$. Although, the complexity of the interpolation operators remains unchanged for all values of α. A further observation is that, given a level of truncation, the operator complexity is now closely related to the grid complexity $C_G \simeq C_A$. This suggests that during truncation there is no increase in matrix density. However, as the denominator in (3.38) varies with the truncation amount, we see that when $Pe_* > 5000$ the operator complexity of $\alpha = 1$ is greater than the operator complexity for $\alpha = 0.5$, even though in direct comparison at each hierarchical level the density of $\alpha = 1$ is smaller than the density of $\alpha = 0.5$. By using a different measure (the average stencil size (3.39)) this anomaly is rectified.

The results in Table 5.14 present the average stencil size C_S (3.39), in terms of the truncation parameter α and Pe_*. For completeness, we also report in brackets

Table 5.14: Average stencil size, C_S, (3.39) and in brackets the (stencil size at finest level), for several different values of truncation parameter α as a function of Pe_*, for Case study 5.1.2 with $N = 857,375$

Pe_*	20	500	1,000	5,000	10,000
$\alpha = 0.0$	778.997(26.4)	1250.18(26.4)	1320.47(26.4)	935.313(26.4)	742.447(26.4)
$\alpha = 0.25$	1.03664(1.00)	1.18548(1.00)	1.31730(1.14)	1.46262(2.15)	1.60849(2.54)
$\alpha = 0.5$	1.00014(1.00)	1.04421(1.00)	1.06464(1.00)	1.04781(1.35)	1.06614(1.54)
$\alpha = 1.0$	1	1	1	1	1

the stencil size for the matrix on the finest level ($C_S^{(1)}$). By comparing the stencil size C_S when $\alpha = 0$ with C_S when $\alpha > 0$ we see that the truncated stencil sizes are considerably smaller (almost equal to the Jacobi case). For $\alpha = 0.25$, we observe that the average stencil size C_S has a small dependence on Pe_*, which is not present for $\alpha = 0.5$. By comparing C_S with $C_S^{(1)}$ for $\alpha = 0$ we observe that there is an increase in matrix density for coarser levels. By introducing truncation, this difference has been reduced significantly. In the case when $Pe_* > 5000$ we observe that C_S is now smaller than $C_S^{(1)}$ for $\alpha = 0.25$ and $\alpha = 0.5$. This observation illustrates that the coarse matrix stencils are now not denser than the fine level stencils. Furthermore, this also explains why the operator complexity is larger for $\alpha = 1$ than $\alpha = 0.25$ and $\alpha = 0.5$ when $Pe_* > 5000$. That is, for $\alpha = 1$ there is no increase or decrease in the matrix density at coarser levels, however for $\alpha = 0.25$ and $\alpha = 0.5$ the matrix density is reduced at coarse levels. This results in C_A for $\alpha = 0.25$ and $\alpha = 0.5$ being proportionately smaller than $\alpha = 1$ (for C_A is smaller for $\alpha = 0.25$ and $\alpha = 0.5$ than for $\alpha = 1$).

5.3 Parallelisation

The effect of parallelisation of the AMG preconditioner should become more pronounced in 3D, where the size of the discrete problems increases much faster with grid refinement. We have seen in Table 5.9 and Table 5.10 a rapid increase in the solution times, which are, in part the consequence of excessive storage requirements (which may cause memory page swapping and thus poor performance). In order to resolve this difficulty, parallelisation should be introduced. In this way both the computational and the storage cost is divided among different processors. In this section we look into the parallel scaling of the AMG preconditioner, using a variety of different smoothers.

The iteration counts for a single processor, shown in Table 5.9, for Case study 5.1.1 scale very well with respect to an increase in Pe_*, when $\text{ILU}_0(0.5)$ and $\text{tILU}_0(0.5, 0.5)$ smoothers are used. Also, we see that $\text{ILU}_0(0.5)$ and $\text{tILU}_0(0.5, 0.5)$ perform comparably. This is considered to be the best we can achieve for tILU_0, in terms of iteration counts. While the iteration counts of an iterative solver with ILU_0 and tILU_0 smoothers change very little as the Pe_* increases, this is far from the case for the Jacobi and Gauss-Seidel smoothers. These iteration count patterns seem to correspond to those observed in two dimensions, see Table 4.19.

In 3D, only the classical Ruge-Stüben coarsening, with no communication between the subdomains, is used. The parallel performance results of Solver Strategy 4.4 for Case study 5.1.1 are summarised in Table 5.15 for two different values of Pe_*. We

Table 5.15: The iteration counts and the (setup,total) execution time (in seconds) required for the convergence of the parallel GMRES solver preconditioned by classical algebraic multigrid V(2,2) cycle with several smoothers, when applied to the solution of the discrete convection-diffusion problem (Case study 5.1.1) obtained from $Q1$ SUPG FEM on a uniform grid, with $N = 857,375$. Each sub-table represents a different diffusion parameter.

(a) Pe_*=500

Smoother \ P	1	2	4	8	16
$ILU_0(0.5)$	6(770.3,917.8)	6(558.9,665.1)	6(263.8,342.2)	6(188.6,234.3)	6(89.8,115.7)
$tILU_0(0.5, 0.25)$	9(398.8,481.0)	9(243.6,296.8)	9(150.3,183.0)	9(131.1,148.3)	9(68.9,78.1)
$tILU_0(0.5, 0.5)$	10(397.1,475.7)	10(231.3,275.3)	10(149.9,177.1)	10(93.6,112.7)	10(69.0,78.9)
$Jacobi(0.5)$	7(386.3,468.3)	7(230.1,277.0)	7(146.9,176.5)	7(92.1,107.1)	7(67.8,76.3)
$GaussSeidel$	5(389.3,445.4)	5(237.8,282.8)	5(146.5,185.4)	5(92.1,130.5)	5(67.7,107.1)

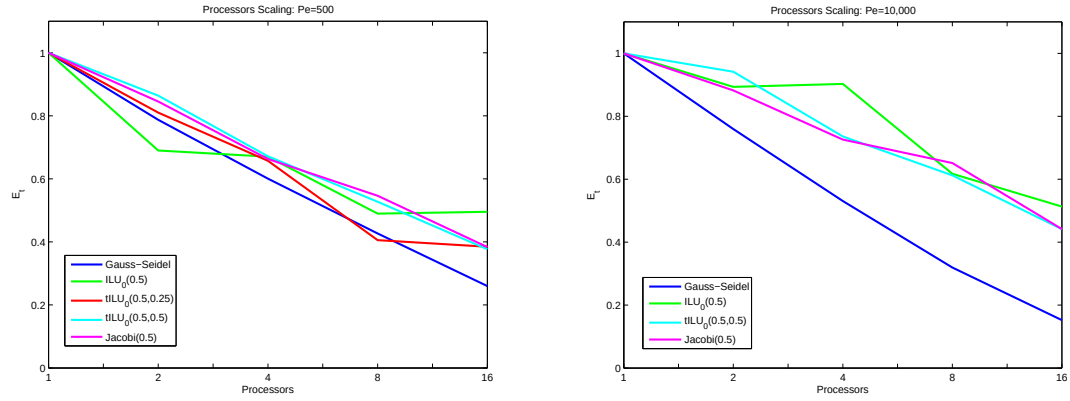
(b) Pe_*=10,000

Smoother \ P	1	2	4	8	16
$ILU_0(0.5)$	5(505.8,613.0)	6(263.0,343.0)	6(126.7,169.7)	6(100.5,124.1)	7(50.2,74.7)
$tILU_0(0.5, 0.5)$	10(241.2,304.7)	10(127.5,161.8)	10(83.2,103.5)	11(45.8,62.2)	11(33.5,43.2)
$Jacobi(0.5)$	20(230.2,391.9)	20(123.3,222.1)	20(73.9,135.0)	20(44.5,75.2)	23(32.4,55.5)
$GaussSeidel$	24(230.2,426.4)	26(123.2,280.7)	26(74.0,200.9)	26(44.6,166.8)	27(32.4,175.0)

use the largest discrete problem size achieved with current computational resources ($N = 857,375$), thus the reported results refer to the strong parallel scaling of the preconditioner (i.e. keeping the problem size fixed while increasing the number of processors). We see that the solver has independent iteration counts with respect to the number of processors, irrespective of the smoother. Also, the best execution times for relatively strong convection are observed when tILU$_0$(0.5,0.5) smoother is used. By taking a two-dimensional surface plot of Case study 5.1.1 we achieve a two-dimensional representation of the problem (see Case study 4.2.1). In Table 4.27 it was observed that all smoothers resulted in an independent iteration count with respect to P. In three dimensions this is also the case. However, with respect to spacial dimensions we find that the Gauss-Seidel and Jacobi smoothers are no longer independent, increasing the solver iteration count by a factor of 2 in the case of a convection-dominated problem. For the case of ILU$_0$ and tILU$_0(0.5, 0.5)$ the smoothers are observed to be close to iteration count independent over spacial dimensions.

In Figure 5.4 we present parallel efficiency (3.42) of the preconditioned solver, as a function of the number of processors P for Case study 5.1.1. From these graphs we see that the Jacobi and tILU$_0$ smoothers give a solver with a consistent efficiency behaviour when Pe_* is increased (this was also the case in 2D, Figure 4.11). By contrast, parallel efficiency of the solver with Gauss-Seidel smoother deteriorates when Pe_* is increased.

In Table 5.16 we present the convergence results of Solver Strategy 4.4 for Case study 5.1.2. For the convection-dominated set of results 5.16(b), we observe a con-



(a) (b)

Figure 5.4: Parallel efficiency for Case study 5.1.1 with $N = 857,375$; (a) $Pe_* = 500$, (b) $Pe_* = 10,000$.

Table 5.16: The iteration counts and the (setup,total) execution time (in seconds) required for the convergence of the parallel GMRES solver preconditioned by classical algebraic multigrid V(2,2) cycle with several smoothers, when applied to the solution of the discrete convection-diffusion problem (Case study 5.1.2) obtained from $Q1$ SUPG FEM on a uniform grid, with $N = 857,375$. Each sub-table represents a different diffusion parameter.

(a) Pe_*=500

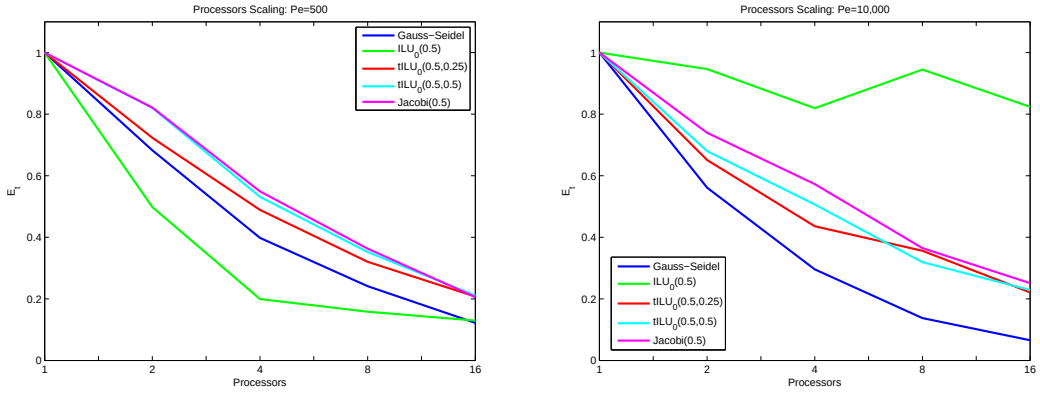
Smoother \ P	1	2	4	8	16
$ILU_0(0.5)$	6(513.4,782.6)	5(577.5,785.3)	5(759.9,980.5)	5(462.6,617.6)	5(279.2,377.4)
$tILU_0(0.5, 0.25)$	7(434.4,635.5)	7(319.6,439.3)	7(237.1,324.8)	7(204.7,248.0)	7(164.5,191.5)
$tILU_0(0.5, 0.5)$	7(433.6,633.7)	7(274.5,386.3)	7(235.4,298.3)	7(180.4,225.0)	7(163.0,187.7)
$Jacobi(0.5)$	6(407.7,621.0)	6(259.6,378.0)	6(216.5,282.8)	6(169.0,213.7)	7(160.1,188.3)
$GaussSeidel$	4(408.6,561.7)	4(275.4,411.4)	4(228.7,352.7)	4(173.5,291.1)	4(163.1,287.5)

(b) Pe_*=10,000

Smoother \ P	1	2	4	8	16
$ILU_0(0.5)$	7(2877,3274)	11(1176,1729)	15(512.6,998.7)	17(222.5,433.1)	19(129.6,248.4)
$tILU_0(0.5, 0.25)$	10(201.8,403.3)	14(140.4,309.7)	18(96.1,231.3)	18(76.9,141.5)	22(67.7,114.0)
$tILU_0(0.5, 0.5)$	11(201.4,418.2)	16(133.9,307.6)	20(90.4,206.3)	20(81.0,163.7)	24(64.9,114.3)
$Jacobi(0.5)$	15(175.4,533.6)	18(121.1,360.7)	21(83.6,232.9)	21(76.4,182.8)	25(66.0,132.9)
$GaussSeidel$	12(175.4,472.3)	15(126.7,420.8)	18(88.5,399.2)	20(76.4,428.7)	22(64.2,448.6)

siderable dependence in iteration counts with respect to the number of processors, when coarsening is performed on a parallel architecture. This is a consequence of the type of coarsening that is used. As in the 2D case, the problem can be fixed using more complex parallel coarsening (for example, the Falgout method, (see Tables 4.28 and 4.29)). The best parallel execution times, with the configuration we adopted, are observed when tILU$_0$ smoothing is used.

The parallel efficiency for Case study 5.1.2 seen in Figure 5.5 shows, as for Case study 5.1.1, that an increase in the relative convection strength does not have a distinct effect on the scaling of Jacobi and tILU$_0(0.5, 0.5)$.



(a) (b)

Figure 5.5: Parallel efficiency for Case study 5.1.2 with $N = 857,375$; (a) $Pe_* = 500$, (b) $Pe_* = 10,000$.

Again, this is not the case for Gauss-Seidel where parallel efficiency is reduced as Pe_* is increased.

5.4 Performance profile and summary

In summary of this chapter, we have analysed the iteration counts, execution times and parallel scaling of the MG preconditioned GMRES solver with different smoothers applied to three-dimensional case studies that are natural extension of two-dimensional Case studies 4.2.1 and 4.2.2 . For GMG as a preconditioner the truncation parameter $\alpha = 0.25$ was used as default for tILU$_0$ smoother, and for AMG preconditioner the truncation parameter $\alpha = 0.5$ was used as default for the tILU$_0$ smoother.

An obvious consequence of moving to a higher spacial dimension is the increase in both the size of coefficient matrix stencils and the rate of growth of discrete problems under mesh refinement (e.g Tables 4.9 and 5.5). For the cases of uniform and recirculating flow, for AMG preconditioned GMRES solver it has been observed that for increasing Pe_* there was a distinct lack of independence in the iteration count when using Gauss-Seidel as a smoother (e.g Tables 4.19 and 4.20) in two-dimensions. In three-dimensions Tables 5.9 and 5.10 show little robust characteristics in the solver when Gauss-Seidel is used as a smoother. However, ILU$_0$ and tILU$_0$(0.5,0.5) are observed to be robust, in comparison.

A MG preconditioner with tILU$_0$ smoother exhibited considerably shorter execution times than when ILU$_0$ smoother is used for three-dimensional problems (Tables 5.1–5.16). Furthermore, the execution times of tILU$_0$ are competitive with Gauss-Seidel when GMG was used as a preconditioner (e.g Table 5.1). For large Pe_* the

execution times of AMG preconditioned tILU$_0$ are shorter and scale substantially better in parallel compared to Gauss-Seidel (Tables 5.9–5.15). The relative amount of entries truncated in the matrix hierarchy using (3.43) in the three-dimensional case is considerably larger than in the two-dimensional case (due to the different matrix stencils) (e.g GMG: Tables 4.14 and 5.8, AMG: Figure 4.10 and Table 5.12).

When different damping parameters are considered for AMG preconditioning with tILU$_0$ smoothing, we observe a negligible change in iteration count when damping parameter γ changes in the range $[0.5, \frac{2}{3}]$. Any further increase of γ leads to an increase in iteration counts.

We conclude our summary with a performance profile of the experiments reported in this chapter (based on total executions times). The performance profile results are presented for three different categories: GMG preconditioning Figure 5.6(a), AMG preconditioning Figure 5.6(b) and parallel AMG preconditioning Figure 5.6(c). In

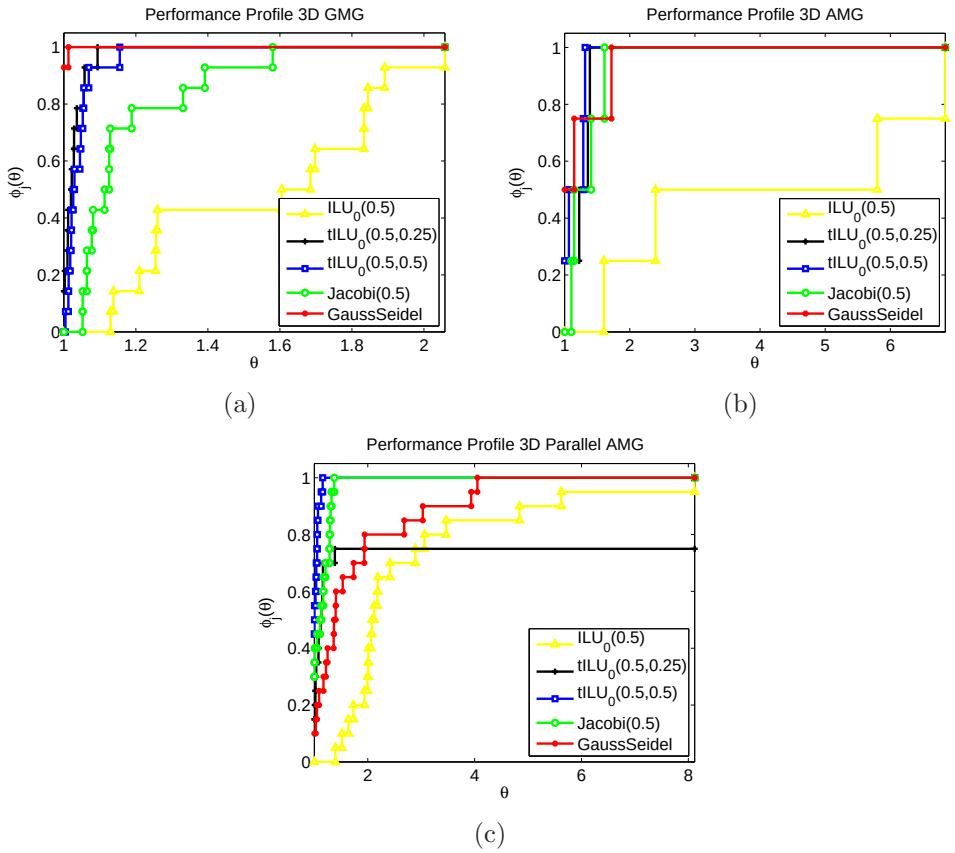


Figure 5.6: Performance profile for Case study 5.1.1 and 5.1.2, based on total execution time. GMRES solver with (a) GMG preconditioning (Pe_*=500, 2000), (b) AMG preconditioning (Pe_*=500, 10000)and (c) Parallel AMG preconditioning (Pe_*=500, 10000).

Figure 5.6(a) the performance profile is taken from Tables 5.1–5.6 using only the largest problem size N with $Pe_* = 500$ and 2000. It can be seen that Gauss-Seidel

is the most successful method according to the performance profile criterion followed closely by $\text{tILU}_0(0.5, 0.25)$.

In Figure 5.6(b) the performance profile is taken from Tables 5.9 and 5.10 using only the largest problem size N with $Pe_* = 500$ and 10000. Figure 5.6(b) shows that all smoothers, with the exception of ILU_0, are within a small factor of one another, with $\text{tILU}_0(0.5, 0.5)$ being the most reliable of smoothers given a small factor $\theta = 1.2$ of the fastest solver. As was the case in two-dimensions (Figure 4.13(b)) AMG preconditioned ILU_0 has a poor performance profile. This is now not only due to classical AMG having considerably denser coarser matrices than the coefficient matrix, but also due to the initial coefficient matrix being denser in three-dimensions (Tables 4.22 and 5.11).

In Figure 5.6(c) the performance profile is taken from Tables 5.15 and 5.16 using only the largest problem size N with $Pe_* = 500$ and 10000 and number of processors $P = 1, 2, 4, 8, 16$. In Figure 5.6(c), as was the case in Figure 4.13(c), we conclude that $\text{tILU}_0(0.5, 0.5)$ is the most robust smoother based on the smallest value of θ in the performance profile framework.

Chapter 6

Conclusion and Future Work

The main aim of this thesis is to evaluate numerically a new smoothing methodology for multigrid preconditioning of the discrete convection-diffusion problems. In the special case of uniform uni-directional wind these results are strengthened theoretically by Fourier smoothing analysis.

Robustness of ILU_0 smoothing in the context of MG preconditioning of the discrete convection-diffusion equation has been studied in the literature. Our numerical tests use a damped version of this smoother ($\gamma = 0.5$) and demonstrate its robustness with respect to the discrete problem size, the relative convection strength and the grid type, on a wide range of test problems in 2D and 3D. However, excessive computational and storage requirements associated with the ILU_0 smoother (especially in the AMG context, if coarsening is performed by the classical Ruge-Stüben method), makes this method not competitive in terms of total execution times. Thus, we are looking for a new smoothing methodology, which retains the asymptotic behaviour of the ILU_0 smoother, but at a considerably lower storage and computational cost (ideally, comparable to the standard point smoothers Jacobi or Gauss-Seidel). We propose a new incomplete factorisation type of smoother, based on the truncation of "insignificant" off-diagonal entries in matrices within the MG hierarchy, before performing incomplete factorisation on these matrices.

The truncation of the coefficient matrices reduces the numerical efficiency of the smoother, compared to a standard ILU_0 method. This will inevitably lead to an increase in the iteration counts, but most importantly, decrease the total execution time. A further characteristic of the new smoother, that we aim to satisfy, is that the asymptotic behaviour of the iteration counts with respect to the problem size, Pe and the type of grids used remains approximately the same as the ILU_0 smoother.

The truncation criterion is based on the strength of dependence principle used in classical AMG. Such a choice of truncation criterion, introduces an additional parameter into the algorithm. Finding an appropriate choice of truncation parameter, for

the tILU_0 smoother, that works efficiently for a wide range of problems (necessary if a "black-box" algorithm is required), may pose a considerable challenge. For geometric multigrid a truncation parameter of $\alpha = 0.25$ was found to perform competitively with respect to the other smoothers, frequently producing the shortest total solve times. For algebraic multigrid a truncation parameter of $\alpha = 0.5$ was found to be more appropriate for the same range of case studies. This increase in the value of the truncation parameter in the AMG case is associated with particular features of classical AMG coarsening.

Parallel computing is becoming ever more popular as the computational costs of solving very large problems increase. With the discrete problem size increasing in modern applications the algorithm designers need to produce new algorithms that are both numerically effective and have good parallel scalability. When smoothing algorithms are concerned, the Jacobi method has the best parallel scaling properties, but proves to be a fairly ineffective smoother in convection-dominated cases. The Gauss-Seidel smoother is a numerically effective smoother, however its parallelisation depends on the appropriate decoupling (red-black ordering) of the unknowns. Performance tests with BoomerAMG show poor scaling of the Gauss-Seidel smoother in both 2D and 3D. Parallelisation of the ILU_0 method (block Jacobi ILU_0) relies on a balanced subdivision of the coefficient matrix among processors, and the application of the ILU_0 algorithm concurrently to all diagonal blocks. In the context of the tILU_0 method, the coefficient matrix has already been truncated dynamically (with respect to the magnitude of the off-diagonal entries). Thus, static truncation based on load balancing, may lead to a much smaller deterioration in numerical efficiency of the tILU_0 algorithm than in the case of the parallel ILU_0 algorithm. These properties make the new tILU_0 smoother a competitive choice in the parallel setting.

Future work

In this project we studied briefly the effectiveness of the new smoothing strategy in the context of MG block preconditioning of the floor-driven cavity Navier-Stokes equations. We found that the beneficial effect of damping for tILU_0 smoother (and its limiting cases Jacobi and ILU_0) is not present when the MG preconditioner is used to invert, approximately, the entire momentum block. We outlined potential reasons for this finding (unsuitability of MG for vector-valued problems, presence of Newton derivative blocks and/or the absence of SUPG stabilisation). Applying Picard's linearisation [31, p.326] can change this situation, as the momentum block has a block diagonal structure. Also, the componentwise application of the MG preconditioner to scalar convection-diffusion subproblems within the momentum block is usually beneficial [31, p.361]. A natural extension is to test the new methodology

for time-dependent problems, where the momentum block is perturbed further by a (beneficial) mass matrix [57]. It is also necessary to test the new MG preconditioner with tILU_0 smoothing on a broader set of more realistic problems in fluid mechanics [28].

A further extension of this work can include testing of the new smoothing strategy in the context of different AMG coarsening. In this project we experimented with classical Ruge-Stüben and Falgout coarsening. By contrast, there exists an entire class of aggregation based MG methods [67]. These methods have naturally low operator complexities (i.e. the coefficient matrices at coarse levels remain relatively sparse). It would therefore be of interest to compare the effectiveness of the tILU_0 smoothing in this context (and to find an optimal value for the truncation parameter α).

In Section 2.1 we say that a scalar convection-diffusion equation represents a singular perturbation of a second-order elliptic problem. Another example of singular perturbation is the Helmholtz equation [7, p.275], this is relevant in modelling electromagnetic waves (acoustic scattering). A common feature of singular perturbations of the Poisson problem is that they introduce h-dependence in the FE discretisations of these problems. For the case of discrete convection-diffusion problems the tILU_0 smoother shows its effectiveness. Standard MG preconditioning of the Helmholtz problem is still an open problem [26] [29]. Thus, finding an effective MG scheme through using tILU_0 as a smoother for the discrete Helmholtz problem would be of interest.

Appendix A

Additional Results

Table A.1 shows the convergence characteristics of the classical AMG preconditioned Krylov solver for Case study 4.2.3. For $Pe_* = 500$ and 2000 the asymptotic performance of GMRES, for all smoothers, remains Pe, h-robust. For moderate values of Pe_* the shortest execution times are obtained when using the Gauss-Seidel smoother, however for highly convective flow ($Pe_* = 10,000$) using Gauss-Seidel as a smoother results in the solver not converging for large problem sizes. In the case of dominant convection ($Pe_* = 10,000$) tILU$_0$(0.5,0.5) smoother has asymptotically the smallest iteration count.

Table A.1: The iteration counts and the (setup,total) execution time (in seconds) required for the convergence of the GMRES solver right-preconditioned by classical algebraic multigrid V(2,2) cycle with several smoothers, when applied to the solution of the discrete convection-diffusion problem (Case study 4.2.3) obtained from $Q1$ SUPG FEM on uniform grids. Each subtable represents a different diffusion parameter. $\times$ denotes a lack of convergence within 100 GMRES iterations.

(a) Pe$_*$=500

Smoother \ N	3969	16129	65025	261121	1046529
$ILU_0(0.5)$	6(0.13,0.27)	6(0.56,1.60)	5(2.43,7.82)	5(11.0,38.9)	5(42.5,151.8)
$tILU_0(0.5, 0.25)$	8(0.11,0.22)	7(0.46,1.18)	7(1.92,6.50)	9(8.59,38.8)	9(33.9,152.2)
$tILU_0(0.5, 0.5)$	9(0.11,0.23)	8(0.45,1.20)	9(1.90,7.39)	10(8.42,39.6)	10(33.5,158.5)
$Jacobi(0.5)$	10(0.03,0.15)	9(0.13,0.92)	9(0.63,6.51)	9(3.32,35.6)	10(13.6,154.0)
$GaussSeidel$	7(0.03,0.11)	7(0.13,0.72)	6(0.63,4.77)	6(3.30,24.8)	6(13.6,98.3)

(b) Pe$_*$=2,000

Smoother \ N	3969	16129	65025	261121	1046529
$ILU_0(0.5)$	6(0.13,0.26)	7(0.56,1.75)	7(2.39,9.43)	7(10.7,47.4)	6(44.7,180.0)
$tILU_0(0.5, 0.25)$	10(0.11,0.25)	11(0.46,1.59)	9(1.93,7.74)	8(8.42,35.6)	8(35.1,152.0)
$tILU_0(0.5, 0.5)$	11(0.11,0.26)	12(0.45,1.60)	11(1.88,8.46)	10(8.22,39.7)	10(34.3,167.7)
$Jacobi(0.5)$	14(0.03,0.20)	13(0.13,1.30)	12(0.60,8.16)	10(3.05,37.9)	9(13.5,145.8)
$GaussSeidel$	9(0.02,0.13)	10(0.13,0.94)	13(0.61,8.66)	14(3.05,48.8)	7(13.5,113.5)

(c) Pe$_*$=10,000

Smoother \ N	3969	16129	65025	261121	1046529
$ILU_0(0.5)$	7(0.13,0.29)	8(0.56,1.91)	11(2.40,13.1)	14(10.5,79.3)	17(43.9,393.9)
$tILU_0(0.5, 0.25)$	11(0.11,0.26)	13(0.47,1.84)	15(1.96,11.6)	15(8.44,59.2)	17(35.0,275.4)
$tILU_0(0.5, 0.5)$	12(0.12,0.26)	15(0.45,1.87)	18(1.91,12.7)	18(8.20,63.6)	13(34.0,206.6)
$Jacobi(0.5)$	16(0.03,0.23)	19(0.12,1.78)	20(0.59,12.9)	18(2.95,63.5)	15(12.9,225.4)
$GaussSeidel$	10(0.03,0.14)	12(0.12,1.09)	19(0.59,12.0)	$\times$	$\times$

Appendix B

Iterative Linear Solvers

The purpose of this tutorial is to show the implementation details and how to specify different iterative linear solvers for `oomph-lib's` Newton solver.

B.1 Overview

For the case of linear problems, by default `oomph-lib's` Newton solver, `Problem::newton_solve(...)`, solves the linear systems that arise from the Newton iteration with its default (direct) linear solver, `SuperLUSolver`.

A direct solver may not always be the most appropriate method of solving a linear system of equations, especially in cases when they are very large. `oomph-lib` provides a number of alternative iterative linear solvers that can be used instead. All iterative linear solvers are derived from the class linear solver, which in turn is derived from a base class `LinearSolver`, which contains a single pure virtual function of the form:

```
virtual void LinearSolver::solve(Problem *const &problem_pt,
                                 Vector<double> &result)=0;
```

The task of this function is to compute the solution $\delta\mathbf{x}$ (returned in the vector `result`) of the linear system

$$\mathcal{J}\delta\mathbf{x} = -\mathbf{r_N},$$

where $\mathbf{r_N}$ and $\mathcal{J}$ are the nonlinear residual vector and the global Jacobian respectively, assembled by the `Problem` which is accessed through the pointer `problem_pt`.

B.2 List of available iterative linear solvers

The online users, can follow the links that will take you directly to the solvers' class references which explain any solver-specific member functions.

- **Point iterative solvers. These methods can be used as a `Solver`, `Preconditioner` or `Smoother`:**
  - `GaussSeidel`: Gauss-Seidel – A special instance of SOR solver, that approximates the coefficient matrix by its lower triangular part. As a solver

this method will need a significant amount of iterations to converge to a tolerance.

- `UpperGaussSeidel`: Upper Gauss-Seidel – Approximates the coefficient matrix by its upper triangular part. Can be used in conjunction with standard Gauss-Seidel to form a symmetric version of the Gauss-Seidel method. This is important when is used as a preconditioner/smoother within Conjugate Gradient method.
- `ILUzero(damping,truncation)`: ILUzero – A special case of LU decomposition, where the factorisation does not create "fill-in". This, in turn, preserves the original sparsity of the coefficient matrix in the incomplete factors. Allows truncation and damped version of the method.
- `Jacobi(damping)`: Jacobi – Computationally cheap method that approximates the coefficient matrix by its diagonal entries. Allows damped version of the method.

- **MultiGrid Solvers:**
  - `GMG`: Geometric Multigrid – Uses a hierarchical sequence of discrete representations of the underlying problem assembled on a nested sequence of uniformly or adaptively refined grids. An interpolation and restriction operator, based of finite element basis functions, are used communicate information between different levels in the hierarchy. At each multigrid level, except the coarsest, a residual (defect) equation is solved approximately by applying a small, fixed number of smoothing iterations.
  - `HypreSolver` : Algebraic Multigrid – Uses an algebraic multigrid method to produce discrete operator hierarchy based on automatic coarsening procedure. Hypre provides a choice of coarsening and smoothing strategies [2] [33]. Hypre also provides a parallel implementation of the AMG and smoothing strategy.

B.3 How to change the LinearSolver

Specifying Gauss-Seidel as a linear solver for a linear system with a sparse coefficient matrix represented in a compressed row format is done by:

```
// Gauss Seidel
//------
    linear_solver_pt()=new GaussSeidel<CRDoubleMatrix>;
```

Gauss-Seidel may not be the best choice for a solver/preconditioner/smoother when used with the default ordering of nodes (unknowns). Ordering the unknowns can be beneficial in some context. In cases of advection-diffusion problems ordering the in direction of the wind can reduce the number of iterations needed by the Gauss-Seidel solver to reach a prescribed tolerance.

```
// Gauss Seidel with lexicographical forward and back ordering
// in Cartesian directions (4 directional ordering)
//------
    linear_solver_pt()=new GaussSeidel<CRDoubleMatrix>(order_typeb,widthb);
```

There is a choice of ordering patterns within `oomph-lib` setup, that include forward and backward directional ordering in each of the Cartesian directions, as well as the black_box ordering, based on a variant of minimum degree ordering algorithm, referred to as Tarjan's algorithm (implemented in the routine HSL13 from the HSL library [1]). To set the vector *order_typeb* for a four-directional ordering in two spacial dimensions we need to proceed as follows:

```
//forward, back, black_box, nothing
// //ordering
// //0:x direction
// //1:y direction
     unsigned widthb=3;
//direction + (x and y order if needed) //dimensions for 2d
     unsigned order_type_nb=4;
     Vector<unsigned>order_typeb(widthb*order_type_nb);
     order_typeb[0]=direction::forward;
     order_typeb[1]=0;
     order_typeb[2]=1;

     order_typeb[3]=direction::forward;
     order_typeb[4]=1;
     order_typeb[5]=0;

     order_typeb[6]=direction::back;
     order_typeb[7]=0;
     order_typeb[8]=1;

     order_typeb[9]=direction::back;
     order_typeb[10]=1;
     order_typeb[11]=0;
```

The variable *widthb* controls the memory space needed for each ordering (the direction and the direction priority).

The following example demonstrates how a black-box ordering of the unknowns is defined:

```
 unsigned width=1;
 unsigned order_type_n=2;
 Vector<unsigned>order_type(width*order_type_n);
 order_type[0]=direction::black_box;
 order_type[1]=direction::black_box;
```

In this example a single iteration will consist of two sweeps using the same black-box ordering.

A Jacobi method can be setup as a linear solver in a similar fashion. The Jacobi method has an additional feature, damping. Damped Jacobi uses a simple variable, for damping, in its constructor.

```
 std::cout<<"Damped=0.5
Jacobi solver=================="<<std::endl;
     std::cout<<endl;
     double weight=0.5;
     linear_solver_pt()=new Jacobi<CRDoubleMatrix>(weight);
```

The ILU_0 method is also setup to work with a damping parameter and can be initialised in a similar fashion as the Jacobi method. Performing ILU_0 on a large sparse matrix, especially in cases of three-dimensional problems, can be time and

memory consuming. These can be reduced by using the new tILU$_0$ method, whereby, prior to applying ILU$_0$ factorisation, the coefficient matrix is analysed and in each row the entries that are small in magnitude (compared to a predefined threshold relative to the largest off-diagonal entry in a particular row) are neglected.

```
// ILUZero(trunc=0.25,damp=0.5) Solver
//-------
     double reduction_weight=0.25 ;//  1=Jacobi, 0=ILU
     double ilu_weight=0.5;//Damping
     bool reduction=true;

     linear_solver_pt()=new ILUZero<CRDoubleMatrix>(ilu_weight,
                                                    reduction,
                                                    reduction_weight,
                                                    order_type,width);
//-----------------------------------.
```

The damping variable in this example is `ilu_weight`, and the relative truncation threshold variable is `reduction_weight`. The two limiting cases of tILU$_0$ are `reduction_weight=0`, which gives the standard ILU$_0$ method and `reduction_weight =1.0` which gives the Jacobi method. The boolean variable `reduction` is used to turn the truncation process off or on, as there is a small additional computational cost associated with performing the reduction. In cases where `reduction_weight=0` it would be better to set the `reduction` boolean to false, as the analysis phase does not need to be performed for ILU$_0$. These methods can all be used as smoothers for geometric multigrid.

For scalar second-order elliptic operators, multigrid (MG) is an optimal solver. In setting geometric MG as a solver we need to set the template to represent the number of spacial dimensions of the problem <2>. The geometric MG method is implemented in `oomph-lib` as a `V-cycle`. A single `V-cycle` constitutes one MG iteration. When MG is used as a solver, `V-cycles` are repeated until the solution residual is reduced to satisfy a given tolerance. Different methods can be used in pre- and post-smoothing. This is achieved by associating two different methods with the functions `pre_smoother_pt()` and `post_smoother_pt()`. The default value settings for pre- and post-smoothers is Gauss-Seidel. The MG setup is performed by:

```
// Geometric Multigrid with pre
smoother=GaussSeidel and postsmoother=
// UpperGaussSeidel using an ordering
//-------

     cout << "Geometric MultiGrid Solver with pre=GS and post=UGS smoother" << std::endl;
     cout << "-----------------------" << std::endl;
     linear_solver_pt()=new MGSolver<2>;

     cout << "GS and UGS smoother using BB ordering" << std::endl;
     cout << "-----------------------" << std::endl;

     pre_smoother_pt()=new GaussSeidel<CRDoubleMatrix>(order_type,width);
     post_smoother_pt()=new UpperGaussSeidel<CRDoubleMatrix>(order_type,width);
```

All the methods mentioned until now can also be used as preconditioners. A default Krylov solver in `oomph-lib` is GMRES which can be used with the left of right preconditioner. Choosing the preconditioning side can be done by setting the

preconditioner_LHS() function to true for the left side or false for the right-side. In the example below a right hand side preconditioner is used. The preconditioner is declared in much the same way as a linear solver. In the example below we are using geometric MG preconditioner with Gauss-Seidel smoother using a four directional nodal ordering.

```
 cout << "GMRES Solver RHS GMG" <<
std::endl;
     cout << "-----------------------" << std::endl;

     bool preconditioner_LHS=false;//false=rhs preconditioner
     linear_solver_pt()=new GMRES<CRDoubleMatrix>;
     dynamic_cast<GMRES<CRDoubleMatrix>*>(linear_solver_pt())->
      preconditioner_LHS()= preconditioner_LHS;


     cout << "GMG preconditioner" << std::endl;
     cout << "-----------------------" << std::endl;
     dynamic_cast<IterativeLinearSolver*>(linear_solver_pt())->
      preconditioner_pt()=new MGSolver<2>;


     cout << "Gauss-Seidel smoother" << std::endl;
     cout << "-----------------------" << std::endl;
    pre_smoother_pt()=new GaussSeidel<CRDoubleMatrix>(order_typeb,
                                                      widthb);
    post_smoother_pt()=new GaussSeidel<CRDoubleMatrix>(order_typeb,
                                                       widthb);
    //-----------------------//
```

In oomph-lib a default algebraic MG solver is BoomerAMG [50] from the package hypre [2] [33]. The setup of an algebraic MG preconditioner is therefore done slightly differently from the geometric MG case:

```
IterativeLinearSolver* oomph_linear_solver_pt = new GMRES<CRDoubleMatrix>;
oomph_linear_solver_pt->tolerance()=1.0e-7;
problem.linear_solver_pt() = oomph_linear_solver_pt;

// Create a new Hypre preconditioner
HyprePreconditioner* hypre_preconditioner_pt = new HyprePreconditioner;
// set the preconditioning method within the hypre solver
hypre_preconditioner_pt->hypre_method() =  HyprePreconditioner::BoomerAMG;
// set the preconditioner in the iterative solver
oomph_linear_solver_pt->preconditioner_pt() = hypre_preconditioner_pt;
```

B.4 Point iterative solver

From a general matrix splitting $\mathcal{J} = M+N$ we obtain a general simple-point iteration:

$$\mathbf{x^{k+1}} = \mathbf{x^k} + M^{-1}\mathbf{r^k}, k = 0, 1, ...$$

where M is a non-singular matrix that is easier to invert than $\mathcal{J}$, and $\mathbf{r^k}$ is the residual at the k-th iteration.

If we replace the splitting $\mathcal{J} = M + N$ by the splitting $\mathcal{J} = D + L + U$, where D is the diagonal L strictly lower and U strictly upper parts of $\mathcal{J}$, we obtain several well-known splitting (simple-point) iterative methods:

- The Jacobi method takes $M = D$. To increase the efficiency of the Jacobi method a damping parameter $\gamma \in (0, 1]$ is added, giving:

$$\mathbf{x^{k+1}} = \mathbf{x^k} + \gamma D^{-1}\mathbf{r^k}, \qquad k = 0, 1, ...$$

- In the Gauss-Seidel method $M = D + L$.

$$\mathbf{x^{k+1}} = \mathbf{x^k} + (D + L)^{-1}\mathbf{r^k}, \qquad k = 0, 1, ...$$

- In the Upper Gauss-Seidel method $M = D + U$.

$$\mathbf{x^{k+1}} = \mathbf{x^k} + (D + U)^{-1}\mathbf{r^k}, \qquad k = 0, 1, ...$$

- The ILU_0 method takes M to be a product of two incomplete factors $\widetilde{L}$ and $\widetilde{U}$. The method can be damped for improved efficiency. The method then uses forward and backward substitution applied to the residual vector $\mathbf{r^k}$.

$$\mathbf{x^{k+1}} = \mathbf{x^k} + \gamma(\widetilde{L}\widetilde{U})^{-1}\mathbf{r^k}, \qquad k = 0, 1, ...$$

The disadvantages of using these methods as solvers is they require a large amount of iterations to converge to a given tolerance.

B.5 Geometric multigrid

Point iterative methods are very efficient at reducing high frequency (oscillatory) error components. However, they are very inefficient at resolving low frequency (smooth) error components. This inefficiency is the reason behind the large number of iterations needed for their convergence to a given tolerance. Thus, an effective way of using point iterative solvers is to apply only a few iterations, to reduce oscillatory error components, followed by a projection of the remaining (smooth) error components to a coarser grid. These simple-point methods are known as smoothers and are represented as $S_L^{\nu_1}(\mathcal{J}_L, \overline{f}_L)$ where ν is the number if iterations. Multigrid is a hierarchical method that attempts to take advantage of the potential of point iterative methods, by allowing the point iterative methods to reduce the high frequency error components at each hierarchical level.

The solve phase of the geometric MG method in `oomph-lib` consists of a standard `V-cycle`. The matrices $\mathcal{J}^{l+1}$, $l = 1, \ldots, L$ are constructed though a direct finite element discretisation of the problem at a different grid level in the MG hierarchy. To transfer the information between different grid levels the interpolation matrices

P_l^{l+1} are introduced. Finally, at the coarsest level the linear system is solved using a direct solver (SuperLU). The MG `V-cycle` can be schematically presented as:

$$
\begin{array}{ll}
\widetilde{\delta x}_L = S_L^{\nu_1}(\mathcal{J}_L, (\overline{r}_N)_L) & \widetilde{\delta x}_L = S_L^{\nu_2}(\mathcal{J}_L, (\overline{r}_N)_L) \\
\overline{r}_L = \overline{f}_L - \mathcal{J}_L \widetilde{\delta x}_L & \widetilde{\delta x}_L = \widetilde{\delta x}_L + \overline{e}_L \\
\quad \overline{r}_{L-1} = R_L^{L-1} \overline{r}_L & \overline{e}_L = P_{L-1}^L \overline{e}_{L-1} \\
\quad\quad \widetilde{e}_{L-1} = S_{L-1}^{\nu_1}(\mathcal{J}_{L-1}, \overline{r}_{L-1}) & \widetilde{e}_{L-1} = S_{L-1}^{\nu_2}(\mathcal{J}_{L-1}, \overline{r}_{L-1}) \\
\quad\quad \widetilde{r}_{L-1} = \overline{r}_{L-1} - \mathcal{J}_{L-1} \widetilde{e}_{L-1} & \widetilde{e}_{L-1} = \widetilde{e}_{L-1} + \overline{e}_{L-1} \\
\quad\quad\quad \ddots & \quad\quad\quad \ddots \\
\quad\quad\quad\quad \widetilde{e}_2 = S_2^{\nu_1}(\mathcal{J}_2, \overline{r}_2) & \widetilde{e}_2 = S_2^{\nu_2}(\mathcal{J}_2, \overline{r}_2) \\
\quad\quad\quad\quad \widetilde{r}_2 = \overline{r}_2 - \mathcal{J}_2 \widetilde{e}_2 & \widetilde{e}_2 = \widetilde{e}_2 + \overline{e}_2 \\
\quad\quad\quad\quad\quad \overline{r}_1 = R_2^1 \widetilde{r}_2 & \overline{e}_2 = P_1^2 \overline{e}_1 \\
\multicolumn{2}{c}{\mathcal{J}_1 \overline{e}_1 = \overline{r}_1.}
\end{array}
$$

The `V-cycle` presented above starts and finishes at the finest grid level `L`. At the finest level L a smoother $(S_L^{\nu}(\mathcal{J}_L, (\overline{r}_N)_L))$ is applied to the system $\mathcal{J}_{\mathcal{L}} \overline{\delta x}_L = (-\overline{r}_N)_L$ ($\overline{r}_N$ is the nonlinear residual). The linear residual $\overline{r}_L$ is then computed and projected (restricted) to a coarse level. At all coarse levels we solve the residual equation approximately, except at the coarsest level, where it is solved exactly. After the exact solution of the residual equation, at the coarsest level, the correction to the solution $\overline{e}_1$ is projected (interpolated) to the finest grid. At each finer grid the correction is further smoothed to remove the high-frequency error components introduced by interpolation. Finally, at the finest level, the correction $\overline{e}_L$ is added to the current approximate solution.

Having in mind that the interpolation matrix and the coarse-level operators in standard geometric MG method are fixed (determined by the finite element interpolation and direct finite element discretisation of the problem on coarse grids), the efficiency of the method is solely determined by the quality of the smoother. In this context, point smoothers work well if the coefficient matrix is diagonally dominant.

B.6 Interpolation matrix

Interpolation in the context of MG is the process of passing the information from a coarser to a finer grid. If the size of the coarser grid is n_c and the size of the finer grid is n_f, then the interpolation matrix P is an $n_f \times n_c$ sparse matrix, constructed from the finite element basis functions as in Algorithm B.1.

Note that this is an example of assembling the interpolation matrix in the case of two-dimensional problem discretised by a uniformly refined quadrilateral grid. A requirement of this setup phase is that the interpolation matrix assumes that the element 1, and the node 1 at level l, must be the same as the element 1 and the node 1 at level $l+1$. For adaptively refined grid refinement hanging nodes must be taken into account when assembling the interpolation matrix (Algorithm B.2).

The restriction matrix can be constructed in a number of ways. We have already introduced the fully weighted restriction matrix which is the transpose of the interpolation matrix. This is the default setting in `oomph-lib`. A simpler (and consequently less effective) restriction is injection (Algorithm B.3). In this case the nodes at a

Solver Strategy B.1 Assembly of the interpolation matrix P_l^{l+1} for uniformly refined grids

for $e = 1 : N_e^l$ %loop over all elements at level l
−if (e is refined)
−for $k = 1 : 4$ %loop over e_{l+1}^k
+for $i = 1 : 4$ %loop over nodes in e_{l+1}^k
$+ii =$ %global node number of the node i_{l+1}
∗for $j = 1 : 4$ %loop over nodes in e_l^k
$∗jj =$ %global node number of the node j_l
$∗P(ii, jj) = \psi_i^l(x_i, y_j)$
$-else$
−for $i = 1 : 4$ %loop over all nodes in $e_l = e_{l+1}$
$+jj =$ %global node number of the node i in e_l
$+ii =$ %global node number of the node i in e_{l+1}
$+P(ii, jj) = \psi_i^l(x_i, y_i)$
−end if

Solver Strategy B.2 Assembly of the interpolation matrix P_l^{l+1}, when hanging nodes are present

for $e = 1 : N_e^l$ %loop over all elements at level l
if (e is refined)
for $k = 1 : N_e$ %loop over nodes e_{l+1}^k
for $i = 1 : N_e$ %loop over nodes in e_{l+1}^k
$ii =$ %global node number of the node i_{l+1}
for $j = 1 :$ number of nodes %loop over nodes in e_l^k
$jj =$ %global node number of the node j_l
if(hanging)
% loop over all master nodes that are related to this hanging node
$P(ii, master_{jj}) = P(ii, master_{jj}) + \psi_i^l(x_i, y_j)$
$else$
$P(ii, jj) = \psi_i^l(x_i, y_j)$
$else$
for $i = 1 : 4$ %loop over all nodes in $e_l = e_{l+1}$
$jj =$ %global node number of the node i in e_l
$ii =$ %global node number of the node i in e_{l+1}
$P(ii, jj) = \psi_i^l(x_i, y_i)$
end if

coarser grid directly take the residual vector values from the nodes at the same geometric positions in the fine grid. This is equivalent to using only the integer entries in the fully weighted restriction matrix.

A modification to this method is the sum injection matrix (Algorithm B.4). Here the sum of each column of the interpolation matrix is multiplied by the residual vector.

Solver Strategy B.3 Assembly of the restriction matrix R_{l+1}^{l}, via injection

$$\begin{aligned} &\text{if}(P(i,j)=1)\\ &\quad rhs^{l+1}[j] = re^{l}[i] \end{aligned}$$

Solver Strategy B.4 Assembly of the restriction matrix R_{l+1}^{l}, via sum injection

$$\begin{aligned} &\text{if}(P(i,j)=1) \quad z=i\\ &\quad rhs^{l+1}[j] = \textstyle\sum_i (P(i,j)) * re^{l}[z] \end{aligned}$$

Appendix C

HPCx Notes

The aim of using HPCx was to further test the scalability of the new MG preconditioner with tILU smoother on a larger number of processors and for large discrete problem sizes. This chapter will cover the procedure and the difficulties encountered during the installation of OOMPHLIB and modified Hypre 2.2b software libraries on the HPCx architecture.

HPCx is the UK science community's capability computing service, run by EPCC and CCLRC on behalf of the EPSRC. The current specifications used are a 160 IBM eServer 575 LPARs for computations, using a IBM PowerPC architecture. Each eServer LPAR contains 16, 1.5Ghz processors, with a total shared main memory of 32GB. The total machine capacity is 2560 processors [4].

C.1 Download

The latest version of the OOMPHLIB (48 at the time) library was downloaded using SVN. To do this the `.profile` file must be modified to include the following path:

PATH=$PATH : /usr/local/packages/svn/subversion - 1.5.6/bin$

The svn scripts used to locate the OOMPHLIB library are:

```
#!/bin/bash if [ $# -ne 2 ]; then
    echo ‘‘Usage: check_me_out username path_to working_copy’’
    exit 1
fi

http-proxy-host = 148.79.162.144
%
http-proxy-port = 8080
%
svn checkout
svn+ssh://$1@orac.ma.man.ac.uk/home/svn-oomphlib/public/trunk $2 cd
$2 svn checkout
svn+ssh://$1@orac.ma.man.ac.uk/home/svn-oomphlib/private/trunk
private cd ..
```

We then run the script as follows: *./check_me_out.sh* <`oomphlib user name`> < `hpcx user directory`> or alternatively download the library from the home directory [45]. Hpcx does not have the capability to zip/unzip a file so we must first do

the following:
`gunzip oomphlib.tar.gz, scp oomphlib.tar username@login.hpcx.ac.uk:.`
We can then unpack the tar file into a directory:
`tar -xvf oomphlib.tar`

C.2 Building the library

Change the compiler options to match those of HPCx compilers.

Serial Compilers: *oomphlib/config/configure_options/hpcx_serial*:

```
--enable-suppress-doc CXX=‘‘-qrtti=all’’ LD=xlf_r CXX=xlC_r CC=xlc_r
F77=xlf_r
```

Parallel Compiler: *oomphlib/config/configure_options/hpcx_mpi*:

```
 --enable-MPI --enable-suppress-doc
CXX=‘‘-qrtti=all’’ LD=xlf_r CXX=mpCC_r CC=mpcc_r F77=mpxlf_r
```

To build the library run
`./autogen --rebuild`
and choose the appropriate compiler configuration.

C.3 Libtool

Libtool is not supported by HPCx but the staff allow a local directory download to be used. If Libtool is not present, the following error will arise:
`Libtool library used but ‘LIBTOOL’ is undefined:`
The usual way to define ‘LIBTOOL’ is to add ‘$AC\_PROG\_LIBTOOL$’ to `configure.in` and run `aclocal` and `autoconf` again.

Download and install Libtool to your local directory as instructed and compile
`./configure --prefix  /oomphlib/locallibtool make; make install`
we then need to add the path to the libtool `bin` and `lib` in `.profile`.
$PATH = \$PATH : \ /oomphlib/locallibtool/bin$
The library path is not set by using $LD\_LIBRARY\_path$ but LIBPATH in AIX.
$LIBPATH = \$LIBPATH : \ /oomphlib/locallibtool/lib$

C.4 OOMPHLIB

Within OOMPHLIB we need to modify `regenerate_cofig_files.sh` from
`aclocal`
to
`aclocal -I  /oomphlib/locallibtool/share/aclocal`
and modify `configure.ac_script/start` to
`AC_CONFIG_MARCO_DIR([m4])`
and `Makefile.am`
`ACLOCAL_AM_FLAGS=-I m4`

We will now find that an error occurs during the `autogen.sh` compilation. This is due to a comparability problem with HPCx not supporting `.sh` and so `echo -n` does not work.

To solve this problem temporarily we can change the top of each of the following files from `bin/sh` to `bin/bash`

```
oomphlib/bin/
    regenerate_config_file.sh
    build_mesh_makefile.sh
    change_headers_to_links.sh
oomphlib/
    autogen.sh
```

The next error will be found in `external_src/arpack` with `debug.h` and `stat.h` being missing. This is also temporarily resolved by coping the files from a GNU distribution.

An observation that was noticed was that a file would need two to five times more memory on the HPCx machine than on a the wulf machine (University of Manchester, Department of Mathematics, parallel machine).

Using `CXX=xlC_r` as a compiler we found that an `oomphlib_utitiy.cc line 349` error exists. This can be resolved by using a `CXX=g++`.

Recompiling OOMPHLIB via
`./autogen --rebuild`
will now compile successfully.

Having successfully compiled OOMPHLIB, we turn our attention to specific problems. A simple one-dimensional problem has now compiled however during runtime a segmentation fault will occur. To resolve the problem of differences between AIX and GNU the following flag was introduced to the `configure_options` `LDFLAGS=‘‘-brtl’’`. However due to the size of the library an `ld:0711-781 ERROR:TOC overflow` error occurs because the table of contents is over 64KB. Many solutions to this were used but with no success. It was concluded that `LDFLAGS=‘‘-brtl’’` could not be used. The segmentation fault was in relation to using a C language function `superlu`. If a C++ language solver was used such as `DenseLU` or `GMRES<CRDoubleMatrix>` we find that the one dimensional Poisson problem is solved accurately.

The problem may have been down to `CXX=g++` and `CC=xlc_r` begin incompatible. Therefore, to resolve this problem the serial compiler `oomphlib/config/configure_options/hpcx_serial`:

```
--enable-suppress-doc CXX=‘‘-qrtti=all’’ LD=xlf\_r CXX=xlC\_r
CC=xlc\_r F77=xlf\_r
```

must be used. The LD flag was another option that was changed, and was found to work best with `LD=xlC_r` or `xlf_r`.
Furthermore to get to the stage where the problem is in the “C” functions a number of files must be commented: `oomphlib_utilities.cc line 349`, `mesh.h line 1348`

and `fsi.cc` taken out of the build process completely. Also, `elements_with_external_element.cc L144` and `oomph_info L168` was also a new problem and needed to be modified to `std::cout`

To resolve the problem of not being able to use any functions from `superlu3.0` which is written in "C" we can add all the relevant files that are used into `src/generic/Makefile.am` for example `sources= /oomphlib/external_src/oomph_superlu3.0/util.c`. This is clearly a temporary solution as the `superlu3.0` library should link up to `src/user_driver/Makefile`.

We can conclude that building the OOMPHLIB library on an AIX machine is problematic.

Bibliography

[1] HSL. http://www.aspentech.com/hsl/, 01-02-10.

[2] Hypre. https://computation.llnl.gov/casc/hypre/download/hypre-2.2.0b_ref_manual.pdf/, 01-02-10.

[3] Trilinos. http://trilinos.sandia.gov/packages/, 01-02-10.

[4] HPCx. http://www.hpcx.ac.uk/, 20-9-2009.

[5] M. Benzi, D. Szyld, and A. van Duin. Orderings for incomplete factorization preconditioning of nonsymmetric problems. *SIAM J. Sci. Comput.*, 20:1652–1670, 1999.

[6] J. Bey and G. Wittum. Downwind numbering: robust multigrid for convection-diffusion problems. *Appl. Numer. Math.*, 23:177–192, 1997.

[7] G. Birkhoff and R. Lynch. *Numerical Solution of Elliptic Problems*. SIAM, Philadalphia, 1984.

[8] J. Boyle, M. Mihajlović, and J. Scott. HSL_MI20: An efficient AMG preconditioner for finite element problems in 3D. *Int. J. Numer. Meth. Engng*, 82(1):64–98, 2010.

[9] J. Boyle and D. Silvester. Algebraic multigrid and convection-diffusion problems. http://www.maths.manchester.ac.uk/~djs/amg06.pdf, 2008.

[10] W. Briggs, V. Henson, and S. McCormick. *A Multigrid Tutorial*. SIAM, Philadelphia, second edition, 2000.

[11] T. Chan. Fourier analysis of relaxed incomplete factorization preconditioners. *SIAM. J. Sci. Stat. Comp.*, 12(3):668–680, 1991.

[12] T. Chan and H. van der Vorst. Approximate and incomplete factorizations. *Parallel Numerical Algorithms*, 4:167–202, 1997.

[13] E. Chow and Y. Saad. Experimental study of ILU preconditioners for indefinite matrices. *J. Comput. Appl. Math.*, 86(2):387–414, 1997.

[14] A. Cleary, R. Falgout, V. Henson, and J. Jones. Coarse-grid selection for parallel algebraic multigrid. *Springer: Lecture Notes in Computer Science*, 1457:104–115, 1998.

[15] A. Cleary, R. Falgout, V. Henson, J. Jones, T. Manteuffel, S. McCormick, G. Miranda, and J. Ruge. Robustness and scalability of algebraic multigrid. *SIAM J. Sci. Comput*, 21:1886–1908, 1998.

[16] E. Cussler. *Diffusion: Mass Transfer in Fluid Systems*. Cambridge University Press, Cambridge, 3rd edition, 2009.

[17] T. Davis. *Direct Methods for Sparse Linear Systems*. SIAM, Philadelphia, 2006.

[18] P. de Zeeuw. Matrix-dependent prolongations and restrictions in a blackbox multigrid solver. *J. Comput. Appl. Math.*, 33(1):1–27, 1990.

[19] J. Donato and T. Chan. Fourier analysis of incomplete factorization preconditioners for three-dimensional anisotropic problems. *SIAM. J. Sci. Stat. Comp.*, 13(1):319–338, 1992.

[20] J. Dongarra, I. Duff, D. Sorensen, and H. van der Vorst. *Numerical Linear Algebra for High-Performance Computers*. SIAM, Philadelphia, 1998.

[21] C. Douglas, G. Haase, J. Hu, M. Kowarschik, U. Rüde, and C. Weiss. Portable memory hierarchy techniques for PDE solvers part II. *ETNA*, 10:21–40, 2000.

[22] P. Drazin. *Introduction to Hydrodynamic Stability*. Cambridge University Press, Cambridge, 2004.

[23] I. Duff, A. Erisman, and J. Reid. *Direct Methods for Sparse Matrices*. Oxford University Press, Oxford, 1986.

[24] H. Elman and M. Chernesky. Ordering effects on relaxation methods applied to the discrete one-dimensional convection-diffusion equation. *SIAM J. Numer. Anal.*, 30(5):1268–1290, 1993.

[25] H. Elman and M. Chernesky. Ordering effects on relaxation methods applied to the discrete convection-diffusion equation. *Institute for Mathematics and Its Applications*, 60:45–57, 1994.

[26] H. Elman, O. Ernst, and D. O'Leary. A multigrid method enhanced by Krylov subspace iteration for discrete Helmholtz equations. *SIAM J. Sci. Comput.*, 23(4):1291–1315, 2001.

[27] H. Elman and G. Golub. Line iterative methods for cyclically reduced discrete convection-diffusion problems. *SIAM J. Sci. Stat. Comput.*, 13(1):339–363, 1992.

[28] H. Elman, M. Mihajlović, and D. Silvester. Fast iterative solvers for buoyancy driven flow problems. *J. Comput. Phys.*, 230(10):3900–3914, 2011. MIMS EPrint: 2010.75.

[29] H. Elman and D. O'Leary. Efficient iterative solution of the three-dimensional Helmholtz equation. *J. Comput. Phys.*, 142(1):163–181, 1998.

[30] H. Elman and A. Ramage. A characterisation of oscillations in the discrete two-dimensional convection-diffusion equation. *Math. Comput.*, 72(241):263–288, 2001.

[31] H. Elman, D. Silvester, and A. Wathen. *Finite Elements and Fast Iterative Solvers*. Oxford University Press, Oxford, 2005.

[32] K. Eriksson, D. Estep, P. Hansbo, and C. Johnson. *Computational Differential Equations*. Cambridge University Press, Cambridge, 1996.

[33] R. Falgout and U. Yang. hypre: A library of high performance preconditioners. In *ICCS '02: Proceedings of the International Conference on Computational Science-Part III*, pages 632–641, London, 2002. Springer-Verlag.

[34] Y. Feng, G. Huang, D. Owen, and D. Perić. An evaluation of iterative methods in the solution of a convection-diffusion problem. *Int. J. Num. Meth. Heat Fluid Flow*, 5:213–223, 1995.

[35] B. Fischer, A. Ramage, D. Silvester, and A. Wathen. On parameter choice and iterative convergence for stabilised discretisations of advection-diffusion problems. *Comput. Methods Appl. Mech. Engrg.*, 179:179–195, 1999.

[36] C. Goldstein. Preconditioning convection dominated convection-diffusion problems. *Int. J. Num. Meth. Heat Fluid Flow*, 5:99–119, 1995.

[37] G. Golub and C. Van Loan. *Matrix Computations*. Johns Hopkins University Press, London, 3rd edition, 1996.

[38] A. Greenbaum. *Iterative Methods for Solving Linear Systems*. SIAM, Philadelphia, 1997.

[39] A. Greenbaum and Z. Strakos. Matrices that Generate the same Krylov Residual Spaces. Technical report, New York University, NY, USA, 1992.

[40] P. Gresho, D. Griffiths, and D. Silvester. Adaptive time-stepping for incompressible flow, part I: Scalar advection-diffusion. *SIAM J. Sci. Comput*, 30:2018–2054, 2008.

[41] P. Gresho and R. Lee. Don't suppress the wiggles–they're telling you something! *Computers and Fluids*, 9(2):223–253, 1981.

[42] P. Gresho and R. Sani. *Incompressible Flow and the Finite Element Method*. John Wiley, Chichester, 1998.

[43] W. Hackbusch. *Iterative Solution of Large Sparse Systems of Equations*. Springer, New York, 1994.

[44] W. Hackbusch and T. Probst. Downwind Gauß-Seidel smoothing for convection-dominated problems. *Numer. Linear. Algebr*, 4(2):85–102, 1997.

[45] A. Hazel and M. Heil. oomph-lib. `http://www.oomph-lib.org/`, 20-8-2008.

[46] A. Hazel and M. Heil. oomph-lib. `http://oomph-lib.maths.man.ac.uk/doc/advection_diffusion/two_d_adv_diff_adapt/html/index.html`, 20-8-2008.

[47] P. Hemker. The incomplete LU-decomposition as a relaxation method in multigrid algorithms. *Computational and Asymptotic Methods for Boundary And Interior Layers. (J.J.H. Miller ed.)*, pages 306–311, 1980.

[48] P. Hemker. On the comparison of line-Gauss Seidel and ILU relaxation in multigrid algorithms. *Computational and Asymptotic Methods for Boundary And Interior Layers (J.J.H.Miller ed.)*, pages 269–277, 1982.

[49] P. Hemker, R. Kettler, P. Wesseling, and P. de Zeeuw. Multigrid methods: Development of fast solvers. *Appl. Math. Comput.*, 13(3-4):311–326, 1983.

[50] V. Henson and U. Yang. BoomerAMG: a parallel algebraic multigrid solver and preconditioner. *Appl. Numer. Math.*, 41(1):155–177, 2002.

[51] D. Higham and N. Higham. *MATLAB Guide*. SIAM, Philadelphia, 2005.

[52] N. Higham. *Accuracy and Stability of Numerical Algorithms*. SIAM, Philadelphia, 2002.

[53] D. Hysom and A. Pothen. A scalable parallel algorithm for incomplete factor preconditioning. *SIAM J. Sci. Comput.*, 22(6):2194–2215, 2001.

[54] C. Kelley. *Solving Nonlinear Equations with Newton's Method*. SIAM, Philadelphia, 2003.

[55] R. Kettler. Analysis and comparison of relaxation schemes in robust multigrid and preconditioned conjugate gradient methods. *Springer: Multigrid Methods: Lecture Notes in Mathematics*, 960:502–534, 1982.

[56] R. Kettler and P. Wesseling. Aspects of multigrid methods for problems in three dimensions. *Appl. Math. Comput.*, 19:159–168, 1986.

[57] D. Key, P. Gresho, D. Griffiths, and D. Silvester. Adaptive time-stepping for incompressible flow, part II: Navier-stokes equations. *SIAM J. Sci. Comput*, 32:111–128, 2010.

[58] D. Kopriva. *Implementing Spectral Methods for Partial Differential Equations*. Springer, New York, 2009.

[59] S. Lipschutz and M. Lipson. *Linear Algebra*. McGraw-Hill, London, third edition, 2001.

[60] T. Manteuffel. An incomplete factorization technique for positive definite linear systems. *Math. Comput.*, 34(150):473–497, 1980.

[61] S. McCormick. *Multigrid Methods*. SIAM, Philadelphia, 1987.

[62] J. Meijerink and H. van der Vorst. An iterative solution method for linear systems of which the coefficient matrix is a symmetric M-matrix. *Math. Comput.*, 31:148–162, 1977.

[63] J. Meijerink and H. van der Vorst. Guidelines for the usage of incomplete decompositions in solving sets of linear equations as they occur in practical problems. *J. Comput. Phys.*, 44(1):134–155, 1981.

[64] G. Meurant. *Computer Solution of Large Linear Systems*. North-Holland, Amsterdam, 1999.

[65] S. Mijalković and M. Mihajlović. *Multigrid methods, In: Wiley Encyclopedia of Computer Science and Engineering*. John Wiley, 2008.

[66] K. Morton. *Numerical Solution of Convection-Diffusion Problems*. Capman & Hall, London, 1996.

[67] Y. Notay. An aggregation-based algebraic multigrid method. *ETNA*, 37:123–146, 2010.

[68] The University of Manchester. Research computing service - horace. `http://www.rcs.manchester.ac.uk/services/computational/Horace`, 20-4-2010.

[69] C. Oosterlee and T. Washio. An evaluation of parallel multigrid as a solver and a preconditioner for singular perturbed problems part I: The standard grid sequence. *SIAM J. Sci. Comput.*, 19(1):87–110, 1998.

[70] C. Oosterlee and T. Washio. Krylov subspace acceleration of nonlinear multigrid with application to recirculating flows. *SIAM J. Sci. Comput*, 21(5):1670–1690, 2000.

[71] A. Ramage. A note on parameter choice and iterative convergence for stabilised discretisations of advection-diffusion problems in three dimensions. *Strathclyde Mathematics Research Report No. 32*, 1998.

[72] A. Ramage. A multigrid preconditioner for stabilised discretisations of advection-diffusion problems. *J. Comput. Appl. Math*, 110:187–203, 1999.

[73] G. Rees. A truncated ILU smoother for multigrid preconditioning of the convection-diffusion equation. *MSc thesis, The University of Warwick*, 2008.

[74] G. Rees, D. Silvester, and M. Mihajlović. A truncated ILU smoother for multigrid preconditioning of convection dominated flow problems. *MIMS EPrint: 2011.34*, 2011. Submitted to J. Comput. Appl. Math.

[75] W. Rohsenow, J. Hartnett, and Y. Cho. *Handbook of Heat Transfer*. McGraw-Hill, New York, 3rd edition, 1998.

[76] H. Roos, M. Stynes, and L. Tobiska. *Numerical Methods for Singularly Perturbed Differential Equations: Convection-Diffusion and Flow Problems*. Springer, Berlin, 1996.

[77] Y. Saad. *Iterative Methods for Sparse Linear Systems*. SIAM, Philadelphia, second edition, 2003.

[78] Y. Saad and H. van der Vorst. Iterative solution of linear systems in the 20th century. *J. Comput. Appl. Math.*, 123(1-2):1–33, 2000.

[79] D. Silvester. A finite element primer. `http://www.maths.manchester.ac.uk/~djs/primer.pdf`, 20-4-2009.

[80] A. Stuart and J. Voss. Matrix analysis and algorithms: Ma398. Lecture Notes, Mathermatics Department, The University of Warwick, 2006.

[81] Syamsudhuha. A study of multigrid methodology for convection-diffusion equations. *PhD, The Universisty of Manchester*, 2002.

[82] R. Tarjan. Depth-first search and linear graph algorithms. *SIAM J. Comput.*, 1:146–160, 1972.

[83] C. Thole and U. Trottenberg. Basic smoothing procedures for the multigrid treatment of elliptic 3D operators. *Appl. Math. Comput.*, 19:333–345, 1986.

[84] L. Trefethen and M. Embree. *Spectra and Pseudospectra: the Behavior of Nonnormal Matrices and Operators*. Princeton University Press, Princeton and Oxford, 2005.

[85] U. Trottenberg, C. Oosterlee, and A. Schüller. *Multigrid*. Academic Press, London, 2001.

[86] S. Turek. *Efficient Solvers for Incompressible Flow Problems*. Springer, Berlin, 1999.

[87] H. van der Vorst. *Iterative Krylov Methods for Large Linear Systems*. Cambridge University Press, Cambridge, 2003.

[88] H. van der Vorst and T. Chan. Linear system solvers: Sparse iterative methods, manuscript. 1997.

[89] H. van der Vorst and G. Sleijpen. The effect of incomplete decomposition preconditioning on the convergence of conjugate gradients. *Notes on Numerical Fluid Mechanics*, 41:179–187, 1993.

[90] F. Wang and J. Xu. A Crosswind Block Iterative Method for Convection-Dominated Problems. *SIAM J. Sci. Comput.*, 21(2):620–645, 1999.

[91] P. Wesseling. A robust and efficient multigrid method. *Springer: Multigrid Methods: Lecture Notes in Mathematics*, 960:614–630, 1982.

[92] P. Wesseling. Theoretical and practical aspects of a multigrid method. *SIAM J. Sci. Stat. Comput*, 3(4):387–407, 1982.

[93] P. Wesseling. *An Introduction to Multigrid Methods*. John Wiley, Chichester, 1992.

[94] G. Wittum. Linear iterations as smoothers in multigrid methods: Theory with applications to incomplete decompositions. *Impact of Computing in Science and Engineering*, 1:180–215, 1989.

[95] G. Wittum. On the robustness of ILU smoothing. *SIAM J. Sci. Stat. Comput.*, 10(4):699–717, 1989.

[96] G. Wittum and F. Liebau. On truncated incomplete decompositions. *BIT*, (29):719–740, 1989.

[97] C. Wu and H. Elman. Analysis and comparison of geometric and algebraic multigrid for convection-diffusion equations. *SIAM J. Sci. Comput.*, 28(6):2208–2228, 2006.

[98] D. Young. *Iterative Solution of Large Linear Systems*. Academic Press, London, 1971.