Masteroppgave

NTNU
Norges teknisk-naturvitenskapelige universitet
Fakultet for informasjonsteknologi og elektroteknikk
Institutt for matematiske fag

Mona-Lena Dordi Norheim

Investigating iterative solvers of Poisson-type equations discretized by the Two-Point Flux-Approximation scheme

Masteroppgave i MSMNFMA
Veileder: Knut-Andreas Lie, Olav Møyner
Juli 2019

NTNU
Kunnskap for en bedre verden

Mona-Lena Dordi Norheim

Investigating iterative solvers of Poisson-type equations discretized by the Two-Point Flux-Approximation scheme

Masteroppgave i MSMNFMA
Veileder: Knut-Andreas Lie, Olav Møyner
Juli 2019

Norges teknisk-naturvitenskapelige universitet
Fakultet for informasjonsteknologi og elektroteknikk
Institutt for matematiske fag

Summary

In this thesis we investigate alternative iterative solvers to MATLAB's in-built direct solver `mldivide` for Poisson-type problems. The solvers are tested on two models used for the purpose of benchmark studies for field development optimization: Olympus and SPE10 [5] [6]. It is found that the iterative solvers, in general, perform better than `mldivide` for large, computationaly heavy systems. It is evident that the use of algebraic multigrid (AMG) as a preconditioner improves convergence dramatically. Among the tested iterative solvers it is Krylov solvers BiCGstab and BiCGstab(l) combined with the smoother ILU(0) and the coarsening strategy smoothed aggregation that performed overall best.

Acknowledgements

I would like to thank my supervisors, Knut-Andreas Lie and Olav Møyner, for presenting me with this assignment. You have guided me through all the stages of my research and been patient with me along the way.

I would also like to thank Lars Sivertsen, who has been of invaluable help and stayed by my side even when the times were tough. Without you I am not sure I would have made it this far.

Abbreviations

0.1 Shorthand notations

MRST	:	MATLAB Reservoir Simulation Toolbox
TPFA	:	Two-Point Flux-Approximation (scheme)
PDE	:	partial differential equation
LHS	:	left-hand side
RHS	:	right-hand side
FDM	:	finite difference methods
FVM	:	finite volume methods
FEM	:	finite element methods
SPD	:	symmetric and positive definite
BHP	:	bottom-hole pressure
STB	:	stock tank barrel
AMG	:	algebraic multigrid
NaN	:	computer shorthand for ”not a number”, f.ex. ”infinite” or imaginary numbers

0.2 Units

m	:	meter
kg	:	kilogram
s	:	second
D (or d)	:	darcy (1 D $\approx 9.869233 \cdot 10^{-13}$m^2)
Pa	:	Pascal ($\sim$ kg/ms^2)
stb/d	:	stock tank barrels a day (measure of oil volumes) (1 stb/d $\approx 1.84 \cdot 10^{-6}$m^3/s)
psi	:	pound-force per square inch (1 psi $\approx$ 6.895 kPa)

0.3 Variables and equations

Ω	:	the domain
$\partial\Omega$	:	the boundary of the domain
$\vec{v} = \frac{Q}{A}$	:	flux density / specific discharge / Darcy flux / Darcy velocity
$\mathbf{v}$	:	velocity field
U	:	characteristic velocity of a fluid flow $[m/s^2]$
p	:	pressure
ρ	:	density
μ	:	dynamic viscosity (fluid viscosity)
φ	:	porosity of the medium
$\kappa = \frac{\rho g K}{\mu}$	:	hydraulic conductivity
$\mathbf{K}$	:	intrinsic permeability (either a constant or a tensor)
$h = \frac{E}{mg} = -z + \frac{p}{\rho g}$	:	hydraulic head, relative to a fixed datum, z axis pointing downwards
Q	:	flow rate
q	:	source/sink term
$Re = \frac{\rho U L}{\mu}$	:	Reynold's number
$\mathbf{I}$	:	the identity matrix (might include a subscript indicating the dimensions of it)

Contents

Chapter 1

Introduction

Most equations that model our physical world are not analytically solvable. The development of numerical methods has enabled us to explore solutions (if only approximate) that before have been unobtainable. Suddenly, solving complex equations is no longer a question of how, but a matter of having enough available computational power to satisfy reasonable tolerances.

The human mind is in reality very good at carrying out complex tasks, such as recognizing faces, playing sports, reading handwriting, and so on. However, our performance is restricted by several factors – limited memory, inter-human communication issues, the risk of making mistakes/miscalculations, the need to rest, lack of motivation or concentration difficulties, illness, etc. – but most of all by the time we require to accomplish these tasks. One of the great advantages of computers is their ability to execute a multitude of simple mindless tasks in the blink of an eye.

Some would say that modern numerical analysis began in 1947 [14] [2], with the work of John von Neumann and Herman Goldstine in their paper ”Numerical Inverting of Matrices of Higher Order”. Among other things this paper discussed what we today refer to as *scientific computing*, which is the science of using computers to solve problems. That is, instead of solving problems by hand we develop algorithms and methods that can efficiently and/or accurately deal with such problems for us. In order to keep up with today's demands efficiency is key/essential.

Even though the efficiency of modern computers still increase exponentially fast, the biggest bottleneck for solving complex systems is computational power – or, rather, the lack of it. To increase the efficiency of existing algorithms, as well as find the domains on which they work best, it is crucial to increase accuracy and save resources.

Linear solvers represent a cornerstone for most numerical simulations – from simple systems, such as the motion of mass on a spring, to incredibly complex systems, such as those used in weather forecasting or reservoir simulation. Due to this variety of applications linear solvers come in a variety of forms, but even though they are all developed with special areas of application in mind the domain on which these solvers are most effective is not universally known (still unclear).

This thesis explores and analyzes the performance of a wide range of different linear iterative solvers when applied to a Poisson-type equation discretized using the two-point flux-approximation (TPFA) scheme. In addition to looking for iterative methods that work better than others on a particular problem, we also want them to be better than direct methods, as this was the purpose for which iterative methods were introduced. For reference we are going to use the in-built MATLAB function `mldivide`, which analyzes the properties of a given linear system and chooses an appropriate direct solver thereafter – an approach that aims at minimizing computation time. On smaller (linear) problems direct solvers tend to outperform iterative ones. For non-linear system of equations iterative solvers are the common (and often the only) choice, but they can also be of great use on larger linear systems, where direct methods would be expensive and even impossible in some cases. The linear systems obtained by TPFA in the upcoming experiments will be solved using the algebraic multigrid library AMGCL [11, p.461]. Among other things, AMGCL can take in as input and combine Krylov subspace solvers, coarsening strategies and smoothers, and it offers preconditioning with or without algebraic multigrid methods. The objective of the thesis will be directed towards the modelling of fluid flow in reservoirs, as the experiments are conducted on different reservoir models. Nevertheless, it should be made clear that any results will be useful in other scientific fields as well.

1.1 Outline of thesis

In Chapter 2 we will begin by reviewing some basic theory on linear algebra and geometry, as well as introducing central physical laws and concepts. In Chapter 3 we look at how the simple Poisson equation can be discretized using finite element methods and finite element methods, before going through the steps of discretizing the stationary pressure equation using TPFA. Chapter 4 gives a brief review of multigrid methods, with particular regard to algebraic multigrid. Chapter 5 introduces the experimental set-up, and takes a peek at some of the general results. Chapters 6 explores how time may behave as a function of the number of grid cells, based on the SPE10 model. Chapter 7 follows with another experiment, this time involving mean values and corresponding standard deviations. The thesis ends with a final discussion and conclusion in Chapter 8, followed by a quick note on future work.

Chapter 2

Important prerequisites

Linear algebra is one of the fundamental building blocks needed when it comes to solving differential equations. We will in this section go through the basic properties of matrices, and how to define grids that allows us to discretize physical systems using basic geometry. Then, we will look at some properties of reservoirs and the differential equations that governs the realm of fluid dynamics. These are the equations that we later will use when testing different types of iterative solvers.

2.1 Concepts from linear algebra

From linear programming (optimization) to quantum computing to error correcting codes (coding theory) to facial recognition, one does not get far without linear algebra. A linear system can arise from a number of places, often from the discretization of PDEs. The TPFA method in Section 3.3 is an example of this. Typically, the more interesting a case is, the more difficult and demanding it is to solve. Sometimes the system coefficient matrix satisfies properties that lessen the workload significantly, by simplifying implementations and reducing both time and memory requirements. This is especially helpful when the system in question is large. Some properties are even necessary for the system to be solvable.

Unless otherwise stated, linear systems of equations will be presented as $\mathbf{A}\mathbf{x} = \mathbf{b}$, where $\mathbf{A} \in \mathbb{R}^{N \times N}$ is the coefficient matrix, $\mathbf{b} \in \mathbb{R}^N$ is the RHS vector, and $\mathbf{x} \in \mathbb{R}^N$ is the vector of unknowns. The exact solution will be denoted by $\mathbf{x}^*$. In this section we will go through some basic matrix properties that are useful to know when handling and analyzing linear systems, but first a brief review of the relationship between matrices and graphs.

2.1.1 Connection to graph theory

An $N \times N$ matrix, such as $\mathbf{A}$, can be seen as a representation of the couplings between N cells/nodes/objects (let us use the term ”cells”). The entry a_{ij} represents how cell i

is connected to cell j, and a_{ji} represents how cell j is connected to cell i. The value of an entry might signify some kind of ”power” or flow from the first cell to the other. For instance, imagine the cells as pools in a water park: then the entries of $\mathbf{A}$ could represent water flow between different pools (how much or how fast water flows, and from where to where) and whether they are connected or not. The *graph* associated to a matrix is an illustration of the connections represented by this matrix. A graph consists of a set of nodes (alternatively called vertices) and a set of edges that connect the nodes. The nodes can, for instance, represent the cells mentioned above. The edges might be *directed*, meaning that each edge is accompanied by an indication of which direction the connection it represents goes, typically illustrated as arrows instead of plain lines. Sometimes there are contributions in both direction between two nodes. A graph with directed edges is called a *directed graph* or a *digraph*. A *weighted* digraph has additional numbers, called *weights*, associated with its edges that represent the strengths of the connections.

2.1.2 Matrix properties

Invertible
First of all, we want $\mathbf{A}$ to be invertible (non-singular), ensuring that the solution is unique for every RHS vector $\mathbf{b}$. For this, it is necessary and sufficient that the determinant of $\mathbf{A}$ is non-zero, i.e., $\det(\mathbf{A}) \neq 0$.

Symmetric
A real matrix $\mathbf{A}$ is *symmetric* if it equals its own transpose ($\mathbf{A}^T = \mathbf{A}$). This is often seen in coefficient matrices that are derived from discretization of PDEs. Symmetric (real) matrices form a subset of the set of Hermitian (complex) matrices, but as our scope is limited to real matrices this will not be further explored.

Positive definite
Another useful property that must be mentioned is positive definiteness. For a symmetric[1] matrix $\mathbf{A} \in \mathbb{R}^{N\times N}$, the following statements are equivalent:

- $\mathbf{A}$ is *positive definite*
- All eigenvalues of $\mathbf{A}$ are strictly positive: $\lambda_i > 0$
- $\mathbf{y}^T\mathbf{A}\mathbf{y} > 0$ for all non-zero vectors $\mathbf{y} \in \mathbb{R}^N$
- All pivots[2] of $\mathbf{A}$ are strictly positive (> 0)

[1] A matrix can be positive definite while not being symmetric, but SPD matrices are preferred because of the nice properties they hold.

[2] The *pivots* of a matrix are central when doing elimination/reduction and obtained by scalar multiplication strictly performed from top to bottom. For matrix $\mathbf{A}$ the first pivot is always the entry $a_{1,1}$. The second pivot is the value of entry $a_{2,2}$ that remains after the entry $a_{2,1}$ has been reduced to zero by adding a multiplum of the first row. The third pivot is the value of entry $a_{3,3}$ after entries $a_{3,1}$ and $a_{3,2}$ are reduced to zero by adding multiplums of the first and second row. And so on.

- All ”upper left determinants” of $\mathbf{A}$, also called leading principal minors[3], are strictly positive (> 0)
- $\mathbf{A} = \mathbf{S}\mathbf{S}^T$, for some real non-singular matrix $\mathbf{S}$ (i.e., with independent columns)

One way to determine the eigenvalues is with the Gershgorin circle theorem, which can be found in the appendix, Section A.1. Since $\mathbf{A}$ is real-valued, the Gershgorin discs will simply be intervals. By replacing $>$ with $\geq$ we get positive *semi*-definiteness.

On the other hand, $\mathbf{A}$ is *negative* (semi-)definite if the inequalities above are flipped. The following statements are equivalent:

- $\mathbf{A}$ is *negative definite*
- All eigenvalues of $\mathbf{A}$ are strictly negative: $\lambda_i < 0$.
- $\mathbf{y}^T\mathbf{A}\mathbf{y} < 0$ for all non-zero vectors $\mathbf{y} \in \mathbb{R}^N$
- All odd principal minors of $\mathbf{A}$ are strictly negative (< 0) and all even principal minors are strictly positive (> 0)

Symmetry and positive definiteness are useful properties for a linear system to possess. Among other things, these properties also ensure uniqueness of the solution. To illustrate this, turn the linear system $\mathbf{A}\mathbf{x} = \mathbf{b}$, with solution $\mathbf{x}^*$, into a minimization problem

$$\min_{\mathbf{x}\in\mathbb{R}^N} f(\mathbf{x}) = \frac{1}{2}\mathbf{x}^T\mathbf{A}\mathbf{x} - \mathbf{x}^T\mathbf{b}. \tag{2.1}$$

Here, $f(\mathbf{x})$ is called the *quadratic test function*, and $\mathbf{x}^*$ is the minimum of $f(\mathbf{x})$, i.e., the point where $\nabla f = 0$. Assume that f has more than one minimum. Since $\mathbf{x}^*$ is a minimum of f, we know that f is locally convex around $\mathbf{x}^*$, and it follows that the double derivative $\nabla^2 f(\mathbf{x}^*)$[4] is positive definite. Since $\nabla^2 f(\mathbf{x}) = \mathbf{A}$, it is independent of $\mathbf{x}$ and f must be (globally) convex. So, if $\mathbf{A}$ is positive definite, then $\mathbf{A}\mathbf{x} = \mathbf{b}$ has the unique solution $\mathbf{x}^*$.

Irreducible

The graph consists of a set of nodes (also called vertices) and a set of edges that connect the nodes. The nodes can represent the cells mentioned above. If the edges are directed, the graph is called a *directed graph*, often abbreviated *digraph*. The directed edges are equipped with arrow heads, pointing in the direction(s) that the connections go.

$\mathbf{A}$ is *reducible* if the indices $1, ..., N$ can be divided into two disjoint sets $i_1, ..., i_\alpha$ and $j_1, ..., j_\beta$, where $N = \alpha + \beta$, such that $a_{i_\mu j_\nu} = 0$ for $\mu \in [1, \alpha]$ and $\nu \in [1, \beta]$. From this definition it follows that $\mathbf{A}$ is reducible

- if and only if its associated digraph is not strongly connected[5].

[3] A *principal submatrix* is obtained from a matrix by removing j of its rows and the corresponding columns. The determinant of this submatrix is called a *principal minor*. By removing the last $N - j$ rows and columns of an $N \times N$ matrix we obtain the *leading principal submatrix* of order j, and the determinant of such a submatrix is called a *leading principal minor*

[4] The double-derivative operator ∇^2 is called the Hessian.

[5] A graph is strongly connected if there exists at least one path from every one node to any other node.

- if and only if it can be transformed into upper triangular form by simultaneous permutations of its rows/columns.

We call $\mathbf{A}$ *irreducible* if it is not reducible.

Diagonally dominant
$\mathbf{A}$ is said to be *diagonally dominant* if the absolute value of each diagonal entry outweighs the absolute values of the other entries in that row combined:

$$|a_{ii}| \geq \sum_{\substack{j \in [1,N] \\ j \neq i}} |a_{ij}| \quad \forall\, i \in [1, N] \tag{2.2}$$

A symmetric diagonally dominant matrix with non-negative diagonal entries is positive semi-definite. If the inequality in Equation (2.2) is strict, then $\mathbf{A}$ is *strictly* diagonally dominant. By the Levy–Desplanques theorem (see appendix, Section A.3), a strictly diagonally dominant matrix is non-singular. This can be proven by using the Gershgorin circle theorem, which is found in Appendix A. Moreover, a symmetric and strictly or irreducibly[6] diagonally dominant matrix with non-negative diagonal entries, is positive definite. If instead the matrix is weakly diagonally dominant, or some of its diagonal entries are zero[7], it will be positive semi-definite.

Sparse
The *sparsity* of a matrix is connected to the number of non-zero entries. A matrix is called *sparse* if most of its entries are zero. On the other hand, if most of its entries are non-zero, the matrix is called *dense*. The *sparsity pattern* of $N \times N$ matrix $\mathbf{A}$ is a set $S \subset \{1, ..., N\}^2$ that satisfies the following condition:

$$\{(i, j) \in \mathbb{N}^2 | a_{i,j} \neq 0\} \subset S \tag{2.3}$$

In words, the sparsity pattern consists of all index pairs (i, j) that correspond to non-zero entries. As an example, take the 5×5 matrix

$$\mathbf{B} = \begin{bmatrix} 1 & 2 & 0 & 3 & 0 \\ 0 & 5 & 0 & 0 & 8 \\ 4 & 4 & 3 & 0 & 1 \\ 0 & 0 & 0 & 2 & 6 \\ 0 & 0 & 1 & 0 & 5 \end{bmatrix}$$

whose sparsity pattern can be seen in Figure 2.1.

2.2 Grids

A grid is a tessellation of a planar or volumetric object (or, objects), dividing it into contiguous non-overlapping simple shapes called *cells*. The geometry of these cells is defined

[6]Irreducibly diagonally dominant simply means irreducible and diagonally dominant.

[7]Zero is defined as being both positive and negative. When speaking of non-negative numbers we therefore exclude zero.

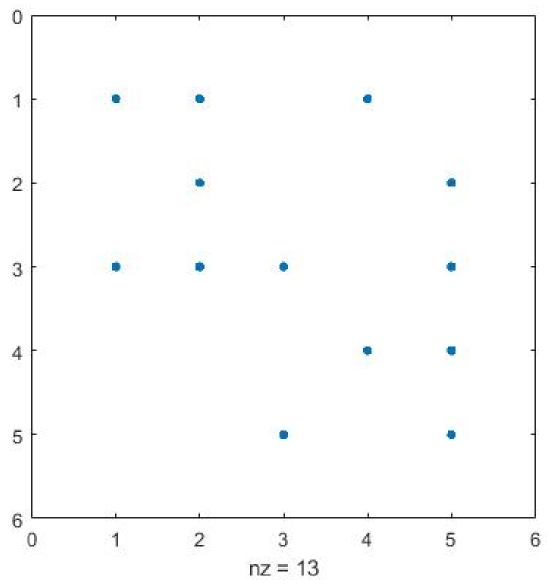


Figure 2.1: Plot showing the sparsity pattern of matrix $\mathbf{B}$, where `nz` denotes the number of non-zero entries.

by vertices, edges and faces. A *vertex* is simply a point. An *edge* is made up of two vertices. A *face* is made up of a set of vertices and edges, the number of which depends on the shape of the face (for example, a rectangular face is made up of four vertices and four edges). Two vertices are neighbors if they are connected by an edge, two faces are neighbors if they have an edge in common, and two cells are neighbors if they share a common face. The *connectivity* of a grid is the total set of connections, and defines the topology of that grid.

Before continuing further with the outline of grid theory, we will briefly review some useful terminology from geometry.

One of the most basic 2D shapes in geometry is the *triangle* (the polygon with three vertices). The *circumcircle* of a triangle is the unique circle that intersects the triangle in exactly its three vertices. In other words, it is the smallest circle containing the triangle. An illustration of this can be found in Figure 2.2 later in this section. The center of the circumcircle is called the *circumcenter*. The *incircle* of a triangle is the largest circle contained inside that triangle. Its center is called the *incenter* and is equidistant from the sides of the triangle. The generalization of triangles in higher dimensions is called *simplices*. A *simplex* in 3D is a tetrahedron. Triangles and tetrahedra are the only simplices of current interest.

Quadrilaterals are polygons with four vertices. Squares and rectangles fall under this category. *Cuboids* are convex[8] polyhedrons with eight vertices and bounded by six quadrilateral faces. Some define cuboids as having rectangular faces, so that all angles are 90 degrees.

The generic term for polygons (2D) and polyhedrons (3D) is *polytopes*.

In the field of grid implementation there is one central dilemma: generality or efficiency. We want to be able to cover all sorts of possible cases, different shapes and structures, and with varying complexity. At the same time we want a fast-working and accurate code. One could draw parallels to Heisenberg's uncertainty principle, in the way that the more precisely we wish to measure the position of a particle, the more inaccurate the measure of its momentum becomes, and vice versa. Exploiting geometrical regularity and topological structures enables huge simplifications when it comes to efficiency. We classify grids by two main types: structured and unstructured.

Structured grids have a regular repeating pattern consisting of only one basic cell shape, where among other things the number of cells meeting at each vertex is constant. Such grids can be represented by multi-index, (i, j, k), which makes it easier to keep track of connectivities. Some examples are Cartesian, rectilinear and curvilinear grids, all with quadrilateral cells (2D) or cuboid cells (3D) [9]. These grids will not be explored any further.

Unstructured grids consist of simple shapes, often several different types, laid out in irregular patterns. Unstructured grids are often used in the modelling of reservoirs. However, in this field it is even more common to begin with a structured grid and then impose certain (unstructured) non-neighboring connections. This is an easier approach and, also, a way to preserve some of the convenient properties of structured grids.

One advantage to unstructured grids is that they are highly flexible and, among other things, can be adapted to complex domains and objects relatively easily. On the other hand, unstructured grids require that we keep a list of the connectivities, in order to "navigate" the grid, e.g., to know the structure of the cells and the connections between them. This requires a lot of memory, something that a multi-index (like in structured grids) does not. Delaunay triangulation and Voronoi diagrams are well-known and widely used examples of unstructured grids, and following is a quick overview of them.

Grids can also be hybrids, containing both structured and unstructured portions.

2.2.1 Delaunay triangulations

Delaunay triangulation is the most common of many methods that generate tessellations based on sets of *generating points*, $\mathcal{P} = \{\mathbf{x}_i \in \mathbb{R}^N\}_{i=1}^m$. Such a tessellation consists of a

[8]A geometrical shape is *concave* if there exists two points on/within it so that the straight line between them exits the shape somewhere, i.e., that at least one point on this line is not contained within the shape. If not, the object is called *convex*.

set of simplices that completely fills the convex hull of $\mathcal{P}$:

$$H(\mathcal{P}) = \left\{ \sum_{i=1}^{n} \lambda_i \mathbf{x}_i | \ \mathbf{x}_i \in \mathcal{P}, \ 0 < \lambda_i \in \mathbb{R}, \ \sum_{i=1}^{n} \lambda_i = 1, \ \text{for } 1 \leq n \leq m \right\}. \tag{2.4}$$

The Delaunay triangulation of $\mathcal{P}$, denoted $DT(\mathcal{P})$, contructs simplices (from $N+1$ points each) such that the hypersphere intersecting the $N+1$ vertices of a Delaunay simplex does not contain any other points. In 2D, simplices and *hyperspheres* simplify to triangles and circumcircles, respectively. In this case, $DT(\mathcal{P})$ is constructed such that no points but the three points defining a triangle in $DT(\mathcal{P})$ can be contained on (or inside) the circumcircle of this triangle. An example with four generating points is shown in Figure 2.2.

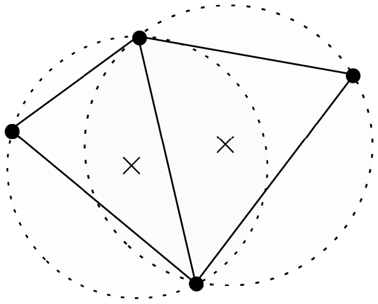


Figure 2.2: Delaunay triangulation of four two-dimensional generating points, consisting of two triangles. The circumcircles are shown in dashed lines, with circumcenters represented by crosses.

2.2.2 Voronoi diagram

The Voronoi diagram of a set of points, $\mathbb{P} = \{\mathbf{x}_i\}_{i=1}^{m}$, is the partitioning of Euclidean space into convex polytopes, one for each generating point $\mathbf{x}_i$ [11, p.71]. The polytopes are called *Voronoi regions* or *Voronoi cells* – we will use the latter one. Each Voronoi cell contains exactly the generating point $\mathbf{x}_j$ that it corresponds to, and every point inside this cell is strictly closer to $\mathbf{x}_j$ than to any other $\mathbf{x}_i$. The Voronoi cell of $\mathbf{x}_j$ can be defined as

$$\mathcal{V}(\mathbf{x}_j) = \{\mathbf{x} \text{ such that } \|\mathbf{x} - \mathbf{x}_j\| < \|\mathbf{x} - \mathbf{x}_i\| \, \forall i \neq j\}. \tag{2.5}$$

The Voronoi cells are not necessarily closed sets, since there might exist points that are equally close to two (or more) generating points. These points will not belong to any Voronoi cells, but they are said to lie on the *Voronoi segments*, which are the lines that enfold the cells (see Figure 2.3).

A Delaunay triangulation of $\mathcal{P}$ corresponds to the dual graph[9] of a Voronoi diagram of $\mathcal{P}$. Connecting the circumcenters in a Delaunay triangulation produces a Voronoi diagram. The edges in the Voronoi diagram are perpendicular to the edges of the Delaunay triangles.

[9]The *dual graph*, H, of a planar graph, G, has a vertex for each cell in G, and for every two cells in G separated by an edge, H has an edge connecting the two corresponding vertices.

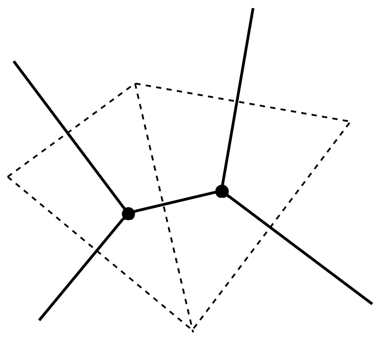


Figure 2.3: Voronoi diagram for the same four two-dimensional generating points as before. The Delaunay triangulation is shown in dashed lines.

2.3 Physical properties of reservoirs

Fluid properties
For basic incompressible single-face flow equations, viscosity and density are deemed the only necessary fluid properties for potential solvers. If the flow is compressible, fluid compressibility is added as a requirement.

Compressibility is the measure of the relative volume change in a solid or a fluid as a reaction to pressure change. A solid/fluid is *incompressible* if the effects of pressure on the material density are zero or negligibly small. If the material density flowing is constant within each infinitesimal fluid volume, the flow is called incompressible. Another indicator of incompressible flow is that the flow velocity field satisfies $\nabla \cdot \mathbf{v} = 0$ in the absence of source terms. The flow of a fluid might be incompressible, even if the fluid itself is not.

The *viscosity* of a fluid is a measure of its resistance to flow. High *viscosity* means that the fluid is ”thicker”. Honey and syrup both have higher viscosity than water, for example.

The *density* of a substance is its mass per unit volume.

Reservoir rock properties
The *porosity* of a porous medium is a measure of the void spaces in that medium. Denoted φ, the porosity is given as the fraction of the bulk volume that is occupied by void space. We assume that $\varphi \in [0,1]$ is a continuous variable such that for $\varphi = 1$ the bulk volume is void only, while for $\varphi = 0$ it is entirely occupied by solid material. In flow simulation, we are only interested in void (pore) space that is interconnected and available to fluid flow. The alternative is void space that is disconnected and cut off from flow, but this is in practice as useful as solid rock.

The *permeability* of a porous medium measures its ability to transmit fluids, and while it is usually equal in both lateral (x and y) directions, it will differ significantly from lateral to vertical (z) direction. If the permeability is high, less force is needed to move the fluid through the medium. Permeability is called *isotropic* if it can be represented by a scalar function $K(\vec{x})$, and *anisotropic* when it needs representation by a full tensor

$$\mathbf{K}(\vec{x}) = \begin{bmatrix} K_{xx} & K_{xy} \\ K_{yx} & K_{yy} \end{bmatrix}, \qquad \mathbf{K}(\vec{x}) = \begin{bmatrix} K_{xx} & K_{xy} & K_{xz} \\ K_{yx} & K_{yy} & K_{yz} \\ K_{zx} & K_{zy} & K_{zz}. \end{bmatrix} \tag{2.6}$$

Here, illustrated in 2D and 3D, respectively. The matrix element K_{ij} represents the dependence between flow rate in i direction and pressure drop in j direction. When modeling physical systems, we need $\mathbf{K}$ to be symmetric, i.e., $K_{ij} = K_{ji}$. The Two-Point Flux-Approximation scheme will only work for cases with a diagonal permeability tensor, where there are no mixed partial derivatives. The focus in this thesis will be on reservoirs where the permeability lies between 0.001 and 1000 mD (milli-Darcy) for liquid flow and down to 10 mD for gases. It should be mentioned that his is only an estimate, because permeability can vary a lot.

2.4 Differential equations

Differential equations that include only one independent variable are called *ordinary*, abbreviated ODEs. Those that include more than one are called *partial*, abbreviated PDEs. Independent variables are usually time (t) or spatial coordinates (x,y,z). Linear[10] second-order[11] PDEs are often classified as *hyperbolic*, *parabolic* or *elliptic*. In two independent variables (x and y), assuming $u_{xy} = u_{yx}$, such an equation will look like

$$au_{xx} + bu_{xy} + xu_{yy} + du_x + eu_y + fu + g = 0 \tag{2.7}$$

where $a, ..., g$ are the coefficients and functions of x and y. Equation (2.7) is

$$\begin{aligned} &\text{hyperbolic if} && b^2 - 4ac > 0 \\ &\text{parabolic if} && b^2 - 4ac = 0 \\ &\text{elliptic if} && b^2 - 4ac < 0 \end{aligned}$$

Naturally, there are some PDEs that do not fit either one of these labels, or that change from one type to another between different regions. An example of this is the Euler-Tricomi equation [4], but this is not of interest in this thesis.

Another way to characterize the three classifications, taken from a lecture on partial differential equations [17], is the following:

1. *Hyperbolic* PDEs describe time-dependent, conservative physical processes, such as convection, that <u>are not</u> evolving toward a steady state.

[10] A differential equation is *linear* if none of its terms are of higher degree than 1.

[11] A differential equation is of *second order* if all its derivative terms are of second order (that is, twice differentiated) or less.

2. *Parabolic* PDEs descibe time-dependent, dissapative physical processes, such as diffusion, that are evolving toward a steady state.

3. *Elliptic* PDEs describe systems that have already reached a steady state, or equilibrium, and hence are time-independent.

This is perhaps a more suitable explanation when the goal is to get a better understanding of the physical meaning behind the notions above.

By both definitions, the Poisson equation and variants of it are elliptic PDEs.

Many basic principles and laws of science are represented as partial differential equations. Below are some examples.

2.4.1 Advection equation

Before introducing the advection equation, it is natural to begin by introducing the concept of advection and, additionally, diffusion and convection.

Diffusion is molecular transport of mass, heat or momentum. It has to do with how a substance in a fluid will dissolve and, eventually, try to ”mix” with the fluid – that is, moving from regions of higher concentration to regions of lower concentration. Picture adding some powder into a glass of water. The powder particles may begin to move randomly around in the water, until they become distributed uniformly. Diffusion is the movement caused by this attempt to mix.

Advection is the bulk transport of mass, heat or momentum, like for instance in a pipe flow, atmospheric flow or bouyant flow. Instead of particles moving randomly around, they will move in bulk and follow the velocity field of the flow.

Convection is the flow that combines diffusion and advection.

Given a known velocity vector field $\mathbf{v}$ and an unknown scalar field ψ of some conserved quantity, the *advection equation* is a partial differential continuity equation that governs the motion of ψ due to advection by $\mathbf{v}$

$$\frac{\partial \psi}{\partial t} + \nabla \cdot (\psi \mathbf{v}) = 0. \tag{2.8}$$

Recall that

$$\nabla \cdot (\psi \mathbf{v}) = \mathbf{v} \cdot \nabla \psi + (\nabla \cdot \mathbf{v})\psi.$$

If the flow is incompressible, i.e., that $\nabla \cdot \mathbf{v} = 0$, then $\mathbf{v}$ is called solenoidal and (2.8) reduces to

$$\frac{\partial \psi}{\partial t} + \mathbf{v} \cdot \nabla \psi = 0. \tag{2.9}$$

If we instead consider a vector quantity $\boldsymbol{\psi}$, the advection equation becomes

$$\frac{\partial \boldsymbol{\psi}}{\partial t} + (\mathbf{v} \cdot \nabla)\boldsymbol{\psi} = 0. \tag{2.10}$$

In this thesis we will mainly consider instances where mass is conserved, in which case we can replace ψ by ρ.

2.4.2 Transport equation

A transport equation is of the form

$$\frac{\partial u}{\partial t} + \nabla \cdot j = S,$$

where u is the scalar field under evaluation, $j = j(u)$ is the flux of u through $\partial\Omega$ and S is the source/sink term inside Ω.

The transport equation is sometimes referred to as the convection-diffusion equation and can, by dividing the flux into its diffusive and convective parts, be rewritten as

$$\frac{\partial u}{\partial t} + \nabla \cdot (D\nabla u) + \nabla \cdot (u\mathbf{v}) = S, \tag{2.11}$$

where D is the diffusion coefficient and $\mathbf{v}$ is the velocity field.

2.4.3 Fick's laws of diffusion

Fick's two laws describe diffusion and can be used to find the diffusion coefficient D (also called diffusivity). Fick's first law states that

$$\vec{j} = -D\nabla u \tag{2.12}$$

where $\vec{j}$ is the diffusion flux and u is the concentration (for ideal mixtures). This law asserts that the flux moves from regions of high concentration to regions of low concentration.

Assuming that D does not depend on u, Fick's second law states that

$$\frac{\partial u}{\partial t} = D\Delta u. \tag{2.13}$$

The second law can be derived from the first law and predicts how diffusion causes the concentration to change with respect to time. It is also equivalent to the diffusion equation.

2.4.4 Darcy's law

Darcy's law describes the flow of fluid through a porous medium. It is, however, only valid for laminar (non-turbulent) flow, which we get for Reynold's number $Re < 1$. The original law [11, p.120] was derived by Henry Darcy in the mid-19th century and states that

$$\frac{Q}{A} = \kappa\frac{h_a - h_b}{L}. \tag{2.14}$$

Here, Q is the flow rate (when inflow equals outflow), $L = b - a$ is the flow length of the vertical experimental tank and A is the cross-sectional area. The quantities h_a and h_b is hydraulic head at the top (inlet) and bottom (outlet) of the tank, respectively, and κ is the hydraulic viscosity. We denote the *specific discharge*, also called *Darcy velocity* or *Darcy* *flux*, by $\vec{v} = \frac{Q}{A}$.

Using that $\kappa = \frac{\rho g K}{\mu}$, $h = -z + \frac{p}{\rho g}$ and $\nabla f = \frac{f(b)-f(a)}{b-a}$, Equation (2.14) can be expressed as

$$\vec{v} = \mathbf{K}\frac{(p_a - p_b) - \rho g(z_a - z_b)}{\mu L} = -\frac{\mathbf{K}}{\mu}\left(\nabla p - g\rho\nabla z\right), \tag{2.15}$$

which is Darcy's law for a single-phase fluid. Recall that z is the vertical axis and has, for convenience, been chosen to point downwards (indicating depth) instead of upwards.

Darcy's law is easily modified to fit horizontal flow only – picture the mentioned experimental tank laid down horizontally instead of standing up vertically. In the absence of gravitational forces the law takes the form

$$\vec{v} = -\frac{\mathbf{K}}{\mu}\nabla p, \tag{2.16}$$

where the depth term is cancelled out because $z_a = z_b$.

2.4.5 The Poisson equation and the pressure equation

In porous media, the general formula for a so-called *Poisson-type* equation is

$$-\nabla \cdot a\nabla u = f, \tag{2.17}$$

where $a(\mathbf{x})$ is a variable coefficient, $f(\mathbf{x})$ denotes the sink/source term and $u(\mathbf{x})$ is the quantity for which we want to solve the equation. Pressure, heat and magnetic field are examples of such quantities.

If the medium is homogeneous, i.e., the rock properties are equal everywhere, Equation (2.17) reduces to the standard *Poisson equation*:

$$-\Delta u = f. \tag{2.18}$$

Here, Δ, alternatively written as ∇^2, is the Laplace operator. In Euclidean space the Laplace operator is the sum of the unmixed partial double derivatives with respect to Cartesian coordinates:

$$\Delta = \nabla^2 = \frac{\partial^2}{\partial x^2} + \frac{\partial^2}{\partial y^2} + \frac{\partial^2}{\partial z^2} \qquad \text{(3D)}.$$

If $f \equiv 0$, the Poisson equation becomes the Laplace equation:

$$\Delta u = 0. \tag{2.19}$$

If there is no diffusion (i.e., $D = 0$) Equation (2.11), with $u = \rho$, reduces to the mass conservation law

$$\rho_t + \nabla \cdot (\rho \mathbf{v}) = S, \tag{2.20}$$

also known as the mass continuity equation. Poisson-type pressure equations are derived from the mass conservation law (2.20) on problems with stationary incompressible flow, i.e., where $u_t = 0$ and the mass density ρ is constant, so that all its derivatives are zero. Writing out the divergence term in Equation (2.20) yields

$$\rho_t + \mathbf{v} \cdot (\nabla \rho) + \rho(\nabla \cdot \mathbf{v}) = S \tag{2.21}$$

$$\rho(\nabla \cdot \mathbf{v}) = S \tag{2.22}$$

$$(\nabla \cdot \mathbf{v}) = \frac{S}{\rho} = q. \tag{2.23}$$

Substituting Darcy's law in the xy-plane, $\vec{v} = -\frac{\mathbf{K}}{\mu}\nabla p$, for the velocity field $\mathbf{v}$, we obtain the Poisson-type pressure equation

$$-\nabla \cdot \left(\frac{\mathbf{K}}{\mu} \nabla p \right) = q. \tag{2.24}$$

Equation (2.24) is sometimes referred to as the *stationary pressure equation*.

Chapter 3

Discretizations and numerical methods

Discretization of PDEs typically leads to large, sparse linear systems of equations, for instance when discretized using *finite difference methods* (FDM), *finite volume methods* (FVM) or *finite element methods* (FEM). Such systems can be solved by either sparse direct solvers or iterative solvers. In this thesis, the focus will be on the latter.

Sparsity of the coefficient matrix, $\mathbf{A}$, can be useful in many ways:

- One only needs to store the non-zero elements, thus using less memory.
- The coefficient matrix is easier to invert.
- It is less expensive, since fewer (binary) operations are required in total for the terms involving $\mathbf{A}$.
- The assembly of stiffness and mass matrices (obtained from the discretizations mentioned above) becomes much faster and cheaper when one does not have to calculate every single matrix entry.

Before we can apply direct or iterative methods in order to solve a linear system, this system must be obtained from discretization of a given partial differential equation. In this thesis, the PDE is the stationary pressure equation,

$$-\nabla \cdot \left(\frac{\mathbf{K}}{\mu} \nabla p \right) = q, \tag{3.1}$$

and we will discretize it using the two-point flux approximation (TPFA) scheme. To understand TPFA, we first need to understand finite volume methods. Finite element methods are closely related as they are both integral schemes, and thus we will explore these as well.

We will now go though some important discretization methods, ending with the two-point flux approximation scheme, which is a central part of the upcoming experiments. But first, a quick review of how the concepts of gradients and divergence might be modified to fit for grids instead of continuous domains.

3.1 Discrete gradient and divergence

Two basic properties of a vector field, which we may imagine model a fluid, is its curl and divergence. The curl gives us a measure of how much the fluid "swirls" around, while the divergence tell us the net flow in or out of a volume. When dealing with systems of fluids, it is thus of great importance to find a way to discretize these quantities on a grid.

Consider a grid consisting of a collection of cells $\mathscr{C} = \{1, ..., n_c\}$ and a collection of faces $\mathscr{F} = \{1, ..., n_f\}$, where n_c denotes the total number of cells, and n_f is the total number faces. Now, let the functions $C_1, C_2 : \mathscr{F} \to \mathscr{C}$ define a mapping from a given face f to its two corresponding neighbor cells, depending on the direction of the flux across f: from $C_1(f)$ to $C_2(f)$. Then, let $F : \mathscr{C} \to \{0,1\}^{n_f}$ be such that it maps a given cell, c, to all of the faces delimiting it. The function returns either a set containing the number of each of these faces or a vector of length n_f with 1's in the location of faces that delimit c and 0's on in the location of the faces that don't. Introducing the flux $\mathbf{v}[f]$ evaluated at a face f and the averaged cell pressure evaluated at c, $\mathbf{p}[c]$, we can express discrete versions of gradients and divergences:

$$\text{grad}(\mathbf{p})[f] = \mathbf{p}[C_2(f)] - \mathbf{p}[C_1(f)], \tag{3.2}$$

$$\text{div}(\mathbf{v})[c] = \sum_{f \in F(c)} \mathbf{v}[f]\mathbf{1}_{c=C_1(f)} - \sum_{f \in F(c)} \mathbf{v}[f]\mathbf{1}_{c=C_2(f)}. \tag{3.3}$$

Here, $\mathbf{1}_{c=C_i(f)}$ is a vector, where element c_j equals 1 if $c_j = C_i(f)$ for a given interface f, and 0 otherwise. It is usual convention to define the direction of $\mathbf{v} \in \mathbb{R}^{n_f}$ from $C_1(f)$ to $C_2(f)$ [11].

3.2 Introduction to the finite volume method and the finite element method

We begin by introducing two important approaches to finding numerical solutions of PDEs: the finite element method and the finite volume method. We will try to keep the reviews brief. For illustration, these will in turn be used to discretize the standard two-dimensional Poisson equation

$$-\Delta u = f, \quad \text{in } \Omega, \tag{3.4}$$

$$u = g, \quad \text{on } \Gamma_D, \tag{3.5}$$

$$-\nabla u \cdot \mathbf{n} = h, \quad \text{on } \Gamma_N, \tag{3.6}$$

where the domain is $\Omega \in \mathbb{R}^2$ and its boundary $\partial\Omega = \Gamma_D \cup \Gamma_N$ is divided into two disjoint parts: one with Dirichlet boundary conditions and the other with Neumann boundary conditions. The unit vector $\mathbf{n}$ is the outward normal to $\partial\Omega$. As we might remember, the Poisson equation is a simplified version of the pressure equation we are interested in.

3.2.1 Finite element method on the Poisson equation in 2D

First of all, note that the finite element method (FEM) is a very rigorous and extensive method. To not get lost in all the details, the review will be rather superficial. More detailed explanations can, for instance, be found in Yousef Saad's book *Iterative Methods for Sparse Linear Systems* [15].

Solving a PDE using finite element methods consists of four steps [12]:

1. Strong formulation
2. Weak (variational) formulation
3. Finite element method
4. Assembly and solution algorithm

Consider the Poisson problem (3.4)-(3.6). Equation (3.4) is called the *strong formulation* of the Poisson problem. Now we need to find the *weak formulation* of Problem (3.4). First, we multiply with an arbitrary test function $w(\mathbf{x}) \in \hat{V}$, where $\hat{V}$ is called the *test space*, and integrate over the domain Ω,

$$\int_\Omega -\Delta u w \,\mathrm{d}\mathbf{x} = \int_\Omega f w \,\mathrm{d}\mathbf{x}. \tag{3.7}$$

This has to hold for every $w \in \hat{V}$. We "get rid of" the double derivative by integrating by parts, using the divergence theorem (see appendix, Section A.2). The LHS of Equation (3.7) becomes

$$\begin{aligned} -\int_\Omega \Delta u \, w \,\mathrm{d}\mathbf{x} &= -\int_{\partial\Omega} \big[\nabla u \cdot \mathbf{n}\big] w \,\mathrm{d}\mathbf{s} + \int_\Omega \nabla u \cdot \nabla w \,\mathrm{d}\mathbf{x} \\ &= -\int_{\partial\Omega} \nabla u \cdot w\mathbf{n} \,\mathrm{d}\mathbf{s} + \int_\Omega \nabla u \cdot \nabla w \,\mathrm{d}\mathbf{x}. \end{aligned}$$

Inserting this into Equation (3.7) we get

$$\int_\Omega \nabla u \cdot \nabla w \,\mathrm{d}\mathbf{x} = \int_{\partial\Omega} (\nabla u \cdot \mathbf{n}) w \,\mathrm{d}\mathbf{s} + \int_\Omega f w \,\mathrm{d}\mathbf{x}. \tag{3.8}$$

The boundary integral can be divided into a Dirichlet part and a Neumann part. We can choose the test space to be such that all test functions are zero at the Dirichlet boundary.

$$\hat{V} = H^1_{0,\Gamma_D}(\Omega) = \left\{ w \in H^1 |\; w(\mathbf{x}) = 0 \;\forall \mathbf{x} \in \Gamma_D \right\}.$$

Because of this (the support of w) the Dirichlet boundary integral disappears and the total boundary integral simplifies to

$$\int_{\partial\Omega} (\nabla u \cdot \mathbf{n}) w \,\mathrm{d}\mathbf{s} = \int_{\Gamma_N} (\nabla u \cdot \mathbf{n}) w \,\mathrm{d}\mathbf{s} = \int_{\Gamma_N} -h w \,\mathrm{d}\mathbf{s}.$$

Inserting this into Equation (3.8), we obtain the weak formulation: *Find $u \in V$ such that*

$$\int_\Omega \nabla u \cdot \nabla w \, \mathbf{dx} = \int_\Omega f w \, \mathbf{dx} - \int_{\Gamma_N} h w \, \mathbf{ds}, \quad \forall w \in \hat{V}. \tag{3.9}$$

The space V is called the *solution space* and is chosen to be the same as $\hat{V}$, except for a shift in the Dirichlet condition:

$$V = H^1_{g,\Gamma_D}(\Omega) = \left\{ w \in H^1 | \; w(\mathbf{x}) = g(\mathbf{x}) \; \forall \mathbf{x} \in \Gamma_D \right\}.$$

Both spaces must be chosen carefully, in order to get a good solution. Variants of $H^1_s(\Omega)$ are common choices, but it depends on what we need of u and w. Typically, $f \in L^2(\Omega)$.

Define

$$\begin{aligned} a(u, w) &= \int_\Omega \nabla u \cdot \nabla w \, \mathbf{dx}, \\ L(w) &= \int_\Omega f w \, \mathbf{dx} - \int_{\Gamma_N} h w \, \mathbf{ds}. \end{aligned}$$

Here, $a(u, w)$ is a coercive continuous bilinear form and $L(w)$ is a continuous linear form.

Now we move on to the next step, which is the finite element method. The approach is to approximate the weak problem (3.9) by a finite-dimensional problem, which is obtained by replacing both solution and test space (V and $\hat{V}$) with discrete subspaces V_h and $\hat{V}_h$, respectively. These subspaces consist of low-degree polynomial functions defined on a version of Ω that is divided into small pieces, called "elements". A common choice for elements is triangles, obtained by triangulating the original domain. We introduce a basis $\phi_j : \Omega \to \mathbb{R}$, for $j = 1, ..., m$, for V_h and $\hat{V}_h$, and assume that all u_h and w_h can be written as

$$\begin{aligned} u_h(\mathbf{x}) &= \sum_{j=1}^{m} U_j \phi_j(\mathbf{x}), \\ w_h(\mathbf{x}) &= \sum_{i=1}^{m} W_i, \hat{\phi}_i(\mathbf{x}) \end{aligned}$$

where U_j and W_i are constants. The basis depends on the type of elements we choose. The functions ϕ_j are often called *shape functions*, and for a tessellation of Ω with nodes/vertices $\mathbf{x}_i$ we need that

$$\phi_j(\mathbf{x}_i) = \begin{cases} 1, \; j = i \\ 0, \; j \neq i. \end{cases} \tag{3.10}$$

However, in this review of FEM we will not go as far as to compute the solution, and will thus not go any further into the possible choices of basis nor tessellation. The approximate problem is: *Find $u_h \in V_h \subset V$ such that*

$$a(u_h, w_h) = L(w_h), \quad \forall w_h \in \hat{V}_h.$$

Inserting this into Equation (3.2.1) yields

$$\begin{aligned}
&\int_\Omega \nabla\left(\sum_{j=1}^m U_j\phi_j\right)\cdot\nabla\left(\sum_{i=1}^m W_i\hat{\phi}_i\right)\,\mathbf{dx} = \int_\Omega f\left(\sum_{i=1}^m W_i\hat{\phi}_i\right)\,\mathbf{dx} - \int_{\Gamma_N} h\left(\sum_{i=1}^m W_i\hat{\phi}_i\right)\,\mathbf{ds}\\
\iff\quad &\sum_{i=1}^m W_i\sum_{j=1}^m U_j\int_\Omega \nabla\phi_j\cdot\nabla\hat{\phi}_i\,\mathbf{dx} = \sum_{i=1}^m W_i\left(\int_\Omega f\hat{\phi}_i\,\mathbf{dx} - \int_{\Gamma_N} h\hat{\phi}_i\,\mathbf{ds}\right)\\
\iff\quad &\sum_{j=1}^m U_j\int_\Omega \nabla\phi_j\cdot\nabla\hat{\phi}_i\,\mathbf{dx} = \int_\Omega f\hat{\phi}_i\,\mathbf{dx} - \int_{\Gamma_N} h\hat{\phi}_i\,\mathbf{ds},\quad i=1,...,m. \qquad (3.11)
\end{aligned}$$

Since the equations must hold for all test functions, the W_i's could be moved outside the integrals and cancelled out. We see that Equation (3.11) can be simplified by substituting in the bilinear and linear forms from above:

$$\sum_{j=1}^m U_j a(\phi_j,\hat{\phi}_i) = L(\hat{\phi}_i),\quad i=1,...,n, \tag{3.12}$$

which is equivalent to expressing it as a linear system of equations, $\mathbf{Au}=\mathbf{b}$, where

$$\begin{aligned}
a_{i,j} &= \int_\Omega \nabla\phi_j\cdot\nabla\hat{\phi}_i\,\mathbf{dx},\\
b_i &= \int_\Omega f\hat{\phi}_i\,\mathbf{dx} - \int_{\Gamma_N} h\hat{\phi}_i\,\mathbf{ds},
\end{aligned}$$

and $\mathbf{A}=(a_{i,j})\in\mathbb{R}^{m,m}$, $\mathbf{u}=[U_1,...,U_m]^T\in\mathbb{R}^m$ and $\mathbf{b}=[b_1,...,b_m]^T\in\mathbb{R}^m$. By solving this system we can recover the coefficients U_j and use them to find the approximate solution u_h.

3.2.2 Finite volume method on the Poisson equation in 2D

The finite volume method (FVM) is particularly popular for discretizing PDEs that arise from physical conservation laws. These laws are of the form

$$\frac{\partial u}{\partial t} + \nabla\cdot\vec{F} = S, \tag{3.13}$$

where $S(\mathbf{x},t)$ is the source term and $\vec{F}(u,t)$ is called the *flux vector* [15, p.68]. Recall, for instance, the transport equation in Section 2.4.2. The standard Poisson equation is derived from Equation (3.13) by letting $\partial u/\partial t = 0$ and $\vec{F}=-\nabla u$. This review might leave out certain detail and, generally, be very simplified compared to what could have been. Again, more details can be found in Saad's book [15].

We begin with the same strategy as when we found the weak formulation in the previous subsection: by multiplying the Poisson equation (3.4) with a test function w, taking the integral over Ω and using integration by parts. As we have already done this for the finite element method, we skip to the resulting weak formulation

$$\int_{\Omega} \nabla u \cdot \nabla w \, \mathbf{dx} - \int_{\partial\Omega} (\nabla u \cdot \mathbf{n}) w \, \mathbf{ds} = \int_{\Omega} f w \, \mathbf{dx}, \tag{3.14}$$

but without dividing the boundary integral and enforcing any boundary conditions.

Now, we want to discretize the different components of Equation (3.14), just like with the FEM. Consider some polygonal control volume Ω_i (for example a triangle) and replace w with a function w_i that is equal to 1 on Ω_i and 0 elsewhere (thus $\nabla w_i = 0$). One could also refer to Ω_i as a (finite volume) cell, and we denote its area (or volume, in higher dimensions) by $|\Omega_i|$ and its boundary by $\Gamma_i = \partial\Omega_i$. Equation (3.14) becomes

$$\int_{\Gamma_i} (\nabla u \cdot \mathbf{n}) \, \mathbf{ds} = \int_{\Omega_i} f \, \mathbf{dx}. \tag{3.15}$$

Equation (3.15) is the basis of the finite volume approximation. Note that we can consider $\vec{F} = \nabla u$ as linear with respect to u, since $\nabla = [\frac{\partial}{\partial x}, \frac{\partial}{\partial y}]^T$. Furthermore, the right-hand side of Equation (3.15) can be approximated as

$$\int_{\Omega_i} f w_i \, \mathbf{dx} \approx f_i |\Omega_i|, \tag{3.16}$$

and the finite volume equation (3.15) itself becomes

$$\nabla \cdot \int_{\Gamma} u \mathbf{n} \, \mathbf{ds} = f_i |\Omega_i|, \tag{3.17}$$

where f_i is some average value of f in Ω_i. The integral in Equation (3.17) can be expressed as the sum of all the integrals over all the m edges of Ω_i. If Ω_i is a triangle, the boundary can be written as $\Gamma_i = \Gamma_1 \cup \Gamma_2 \cup \Gamma_3$ and Ω_i will have (up to) three neighboring cells, Ω_1, Ω_2 and Ω_3 (see Figure 3.1). Now, denote by $\bar{u}_{ij}$ some average approximating u at edge Γ_j (between Ω_i and some adjacent cell Ω_j), and let $\vec{l_j} = l_j \vec{n}_j$ where l_j is the length of edge Γ_j. A simple approximation of the left-hand side in Equation (3.17) could be

$$\nabla \cdot \int_{\Gamma} u \mathbf{n} \, \mathbf{ds} \approx \sum_{j=1}^{m} \bar{u}_{ij} \nabla \cdot \mathbf{n}_j l_j = \sum_{j=1}^{m} \bar{u}_{ij} \nabla \cdot \vec{l_j}. \tag{3.18}$$

The unknown u_j's, for which we try to solve the system of equations, are the approximations of function u associated with each cell Ω_j. They can be defined as the approximation of u at the center of gravity of each Ω_j (*cell-centered* approximation) or at the vertices of each Ω_j (*cell-vertex* approximation). With cell Ω_i as reference cell, we let

$$\bar{u}_{ij} = \frac{1}{2}(u_j + u_i). \tag{3.19}$$

This approximation of $\bar{u}_{ij}$ is very simple and will, among other thing, struggle in cases with large gradients of u. An alternative here could be to exploit upwind schemes, but such

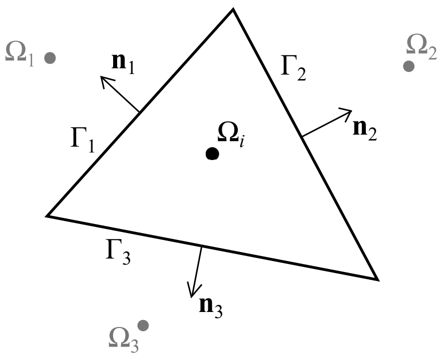


Figure 3.1: Triangular finite volume cell Ω_i, with three neighboring cells Ω_1-Ω_3 and corresponding normal vectors.

approximations would also have limitations and we will not explore this direction further. Inserting (3.19) into Equation (3.17) yields

$$\frac{1}{2}\sum_{j=1}^{m}(u_j + u_i)\nabla \cdot \vec{l}_j = f_i|\Omega_i|. \tag{3.20}$$

Now, take into account that $\sum_{j=1}^{m} \vec{l}_j = 0$, then

$$\begin{aligned}\sum_{j=1}^{m}(u_j + u_i)\nabla \cdot \vec{l}_j &= \sum_{j=1}^{m} u_j\nabla \cdot \vec{l}_j + \sum_{j=1}^{m} u_i\nabla \cdot \vec{l}_j \\ &= \sum_{j=1}^{m} u_j\nabla \cdot \vec{l}_j + u_i\nabla \cdot \underbrace{\sum_{j=1}^{m}\vec{l}_j}_{=0} = \sum_{j=1}^{m} u_j\nabla \cdot \vec{l}_j.\end{aligned}$$

With this in mind, Equation (3.20) becomes finite volume approximation

$$\frac{1}{2}\sum_{j=1}^{m} u_j\nabla \cdot \vec{l}_j = f_i|\Omega_i|, \tag{3.21}$$

where f_i is the approximate value of the source function f in cell Ω_i for $i = 1, ..., N_c$ and N_c is the number of cells in the grid.

3.3 Two-point flux approximation

The two-point flux-approximation (TPFA) is a simple finite volume discretization

$$v_{i,j} = -T_{ij}(p_i - p_j) \tag{3.22}$$

linking the difference in cell averaged pressure between two neighbouring cells, Ω_i and Ω_j, to the flux across the interface between them, Γ_{ij}.

Consider the simple single-phase flow equation [11, p.138]

$$-\nabla \cdot \left(\frac{\mathbf{K}}{\mu} \nabla p \right) = q, \tag{3.23}$$

without loss of generality. Assume the domain Ω is partitioned into a finite set of cells. We now choose some cell $\Omega_i \subset \Omega$ in the discrete grid to be our control volume (just like in the FVM) and take the integral

$$\int_{\Omega_i} -\nabla \cdot \left(\frac{\mathbf{K}}{\mu} \nabla p \right) \, \mathrm{d}\mathbf{x} = \int_{\Omega_i} q \, \mathrm{d}\mathbf{x}. \tag{3.24}$$

By using the divergence theorem (Section A.2) on the left-hand side, Equation (3.24) becomes:

$$\int_{\partial\Omega_i} - \left(\frac{\mathbf{K}}{\mu} \nabla p \right) \cdot \mathbf{n} \, \mathrm{ds} = \int_{\Omega_i} q \, \mathrm{d}\mathbf{x}. \tag{3.25}$$

Equation (3.25) guarantees mass conservation for every grid cell. Let Ω_j be a cell adjacent to Ω_i and let $\Gamma_{i,j} = \partial\Omega_i \cap \partial\Omega_j$ be the interface (*half-face*) between them, with area $A_{i,j}$ and outward normal $\mathbf{n}_{i,j}$ to $\partial\Omega_i$. As the name suggests, the two-point flux-approximation

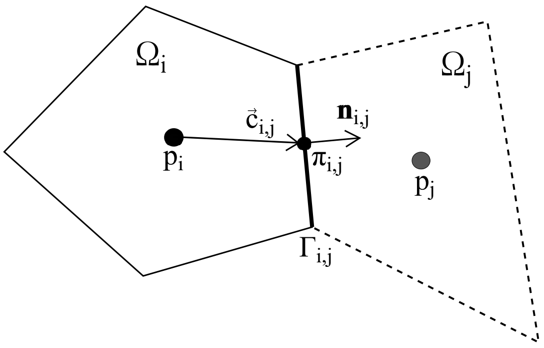


Figure 3.2: Cell Ω_i and one of its neighbors Ω_j, where p_i and p_j are the respective cell averaged pressures and $\pi_{i,j}$ is the pressure at the face centroid $\mathbf{x}_{i,j}$.

scheme uses two "points" (one in each of two adjacent cells) to compute the flux across an interface. We need to compute the flux across each face of cell Ω_i

$$v_{i,j} = \int_{\Gamma_{i,j}} - \left(\frac{\mathbf{K}}{\mu} \nabla p \right) \cdot \mathbf{n} \, \mathrm{ds}. \tag{3.26}$$

Because the grid is assumed to match, we know that each pair of twin half-faces have equal areas $A_{i,j} = A_{j,i}$ but opposite-directed normals $\mathbf{n}_{i,j} = \mathbf{n}_{j,i}$. Using the midpoint rule[1] with one (sub)interval, the integral (3.26) can be approximated as

$$v_{i,j} \approx -A_{i,j} \left(\frac{\mathbf{K}}{\mu} \nabla p \right) (\mathbf{x}_{i,j}) \cdot \mathbf{n}_{i,j}. \tag{3.27}$$

[1]The midpoint rule approximates an integral $\int_a^b f(x) \, \mathrm{d}x$ as $\frac{b-a}{n} \left(f(x_1^*) + ... + f(x_n^*) \right)$, where n is the number of subintervals that the interval $[a, b]$ is divided into and x_j^* are the midpoints of each subinterval.

We need to discretize the pressure gradient. A possible approach is to use a one-sided finite difference that expresses the gradient by the difference between the pressure at the face centroid and the cell averaged pressure inside Ω_i. However, there is no particular point value associated to the cell averaged pressure p_i, so we must reconstruct it. By assuming linear pressure inside each cell, the pressure calculated at the cell center must equal the average pressure p_i. Then

$$\nabla p(\mathbf{x}_{i,j}) \approx \frac{(\pi_{i,j} - p_i)}{|\vec{c}_{i,j}|}\frac{\vec{c}_{i,j}}{|\vec{c}_{i,j}|} = \frac{(\pi_{i,j} - p_i)}{|\vec{c}_{i,j}|^2}\vec{c}_{i,j},$$

where $\vec{c}_{i,j}$ is the vector pointing from the cell center of Ω_i to the face centroid of $\Gamma_{i,j}$. Thus, the flux can be approximated as follows

$$v_{i,j} \approx -A_{i,j}\frac{\mathbf{K}_i}{\mu}\frac{(\pi_{i,j} - p_i)}{|\vec{c}_{i,j}|^2}\vec{c}_{i,j} \cdot \mathbf{n}_{i,j} \tag{3.28}$$

$$= A_{i,j}\frac{\mathbf{K}_i}{\mu}\frac{(p_i - \pi_{i,j})}{|\vec{c}_{i,j}|^2}\vec{c}_{i,j} \cdot \mathbf{n}_{i,j}, \tag{3.29}$$

where $\mathbf{K}_i$ is the permeability (tensor) in Ω_i. We define the one-sided *transmissibilies*, or *half-transmissibilities*, as

$$T_{i,j} = A_{i,j}\frac{\mathbf{K}_i}{\mu}\frac{\vec{c}_{i,j} \cdot \mathbf{n}_{i,j}}{|\vec{c}_{i,j}|^2} \qquad \text{associated with half-face } \Gamma_{i,j} \tag{3.30}$$

The two half-transmissibilities, $T_{i,j}$ and $T_{j,i}$, each depend solely on the geometrical grid properties and the permeability tensor of a single cell (Ω_i and Ω_j, respectively). To find the two-sided transmissibility, we assume continuity of the pressures at all face centroids and of the fluxes across all faces, i.e., $\pi_{i,j} = \pi_{j,i} = \pi_{ij}$ and $-v_{j,i} = v_{i,j} = v_{ij}$. From Equation (3.29) we have that $v_{i,j} = T_{i,j}(p_i - \pi_{i,j})$ and $v_{j,i} = T_{j,i}(p_j - \pi_{j,i})$, which become

$$T_{i,j}^{-1}v_{ij} = p_i - \pi_{ij} \quad \text{and} \quad -T_{j,i}^{-1}v_{ij} = p_j - \pi_{ij}.$$

Subtracting one from the other yields

$$\begin{aligned}\left(T_{i,j}^{-1} + T_{j,i}^{-1}\right)v_{ij} &= T_{i,j}^{-1}v_{ij} + T_{j,i}^{-1}v_{ij}\\ &= (p_i - \pi_{ij}) - (p_j - \pi_{ij})\\ &= p_i - p_j.\end{aligned}$$

Finally, we have arrived at the TPFA scheme

$$v_{ij} = \left(T_{i,j}^{-1} + T_{j,i}^{-1}\right)^{-1}(p_i - p_j) = T_{ij}(p_i - p_j), \tag{3.31}$$

where T_{ij} is the (two-sided) transmissibility associated with the connection between cells Ω_i and Ω_j. Transmissibility is a measure of the connection strength between two neighboring grid cells and contains information about permeability and the grid geometry. By inserting this scheme (3.31) back into Equation (3.25) we get

$$\sum_j v_{ij} = \sum_j \int_{\Gamma_{ij}} -\left(\frac{\mathbf{K}}{\mu}\nabla p\right)\cdot \mathbf{n}\,\mathrm{ds} = \int_{\partial\Omega_i} -\left(\frac{\mathbf{K}}{\mu}\nabla p\right)\cdot \mathbf{n}\,\mathrm{ds} = \int_{\Omega_i} q\,\mathrm{d}\mathbf{x}$$

$$\Rightarrow \quad \sum_j T_{ij}(p_i - p_j) = q_i, \quad \forall\, \Omega_i \subset \Omega. \tag{3.32}$$

Let us write out Equation (3.32) for cell Ω_i:

$$\begin{aligned} q_i &= v_{ii_1} + v_{ii_2} + ... + v_{ii_m} \\ &= T_{ii_1}(p_i - p_{i_1}) + T_{ii_2}(p_i - p_{i_2}) + ... + T_{ii_m}(p_i - p_{i_m}) \\ &= \underbrace{\left(\sum_{j=1}^{m} T_{ii_j}\right) p_i}_{diagonal} \underbrace{- T_{ii_1}p_{i_1} - T_{ii_2}p_{i_2} - ... - T_{ii_m}p_{i_m}}_{off-diagonal}, \end{aligned}$$

where Ω_j for $j = i_1, ..., i_m$ are all the cells adjacent to Ω_i. We see that the TPFA discretization of the pressure equation (3.23) can be represented as a linear system of equations, $\mathbf{Ap} = \mathbf{q}$, where the matrix entries are of the form

$$a_{i,k} = \begin{cases} \sum_j T_{ij}, & \text{if } k = i \\ -T_{ik}, & \text{if } k \neq i \\ 0, & \text{otherwise.} \end{cases} \tag{3.33}$$

For logically Cartesian[2] grids in 1D, 2D and 3D the matrix $\mathbf{A}$ will be tridiagonal (three diagonals), pentadiagonal (five diagonals) and heptadiagonal (seven diagonals), respectively. We ensure positive definiteness of $\mathbf{A}$ by adding a positive constant to entry $a_{1,1}$ such that $p_1 = 0$

Advantages of the TPFA scheme:

- It is easy to implement
- It yields sparse linear systems that are relatively cheap to invert
- It produces monotone pressure approximations, thus robust

Disadvantages:

- If the grid does not satisfy K-orthogonality (i.e., conditionally consistent), TPFA will fail in providing convergent solutions for increased grid resolutions

Linear systems arising from TPFA discretization of pressure equations, such as Equations (2.24) and (3.23), are always symmetric positive definite (SPD). Many numerical methods benefit greatly from the coefficient $\mathbf{A}$, of linear system $\mathbf{Ax} = \mathbf{b}$, being SPD. Among other things, if $\mathbf{A}$ is SPD we know that any local solution of the system is also a global solution.

3.4 Iterative methods and preconditioning

As the reader may know, iterative methods aim to solve mathematical problems by generating a sequence of improving approximate solutions until certain termination criteria are

[2]Grids that can be viewed as embedded in a Cartesian-like box are called *logically cartesian*, e.g. a circular or L-shaped domain.

met and convergence is achieved. However, convergence is rarely guaranteed, and even if a method does converge, it can do so very slowly. This depends heavily on the nature of the coefficient matrix. For a standard linear system, $\mathbf{Ax} = \mathbf{b}$, the approximate solution at the n'th iteration is denoted $\mathbf{x}_n$. We want that $\lim_{n\to\infty} \mathbf{x}_n = \mathbf{x}^*$, but it is sufficient that the error between approximation and exact solution lies within some small tolerance, in order to achieve convergence. However, the norm of the error, $\|e_n\| = \|\mathbf{x}^* - \mathbf{x}_n\|$, is often difficult to measure. A common alternative is to consider the norm of the residual, $\|\mathbf{r}_n\| = \|\mathbf{b} - \mathbf{Ax}_n\|$. The error and the residual are related in the following way:

$$\mathbf{r}_n = \mathbf{b} - \mathbf{Ax}_n \tag{3.34}$$

$$\mathbf{r}_n = \mathbf{Ax}^* - \mathbf{Ax}_n \tag{3.35}$$

$$\mathbf{r}_n = \mathbf{A}(\mathbf{x}^* - \mathbf{x}_n) \tag{3.36}$$

$$\mathbf{r}_n = \mathbf{Ae}_n \tag{3.37}$$

3.5 Preconditioning

Preconditioning is a way of transforming a linear system into one that is likely easier to solve with an iterative method, without affecting the solution. A typical goal by preconditioning is to reduce the condition number of the problem at hand. That is, if a matrix $\mathbf{P}$ is a preconditioner of a matrix $\mathbf{A}$, then $\mathbf{P}^{-1}\mathbf{A}$ has a smaller condition number than $\mathbf{A}$. There are many kinds of condition number, but one definition is that the condition number is a measure of how much a slight change in the input argument of a function will affect the output value. For a linear system $\mathbf{Ax} = \mathbf{b}$, the question is how much small perturbations in the RHS vector $\mathbf{b}$ will affect the solution $\mathbf{x}$. If, for instance, a row/column of $\mathbf{A}$ is close to be a linear combination of the other rows/columns, then the condition number of $\mathbf{A}$ will be large, and even tend towards infinity, as the matrix is nearly singular. A singular matrix will have a condition number that tends to infinity. Convergence of the preconditioned system should be much faster than that of the original system. Other requirements for the preconditioning matrix $\mathbf{P}$ is that it itself is non-singular, that it is an easily invertible approximation to $\mathbf{A}$, and that operations with $\mathbf{P}^{-1}$ are inexpensive to perform.

A simple way of constructing a preconditioner is by performing an *incomplete LU factorization* (ILU) on $\mathbf{A}$, from which we obtain $\mathbf{A} = \mathbf{LU} - \mathbf{R}$. For preconditioning purposes we choose $\mathbf{P} = \mathbf{LU}$ and use it as preconditioner in another iterative solver (for instance, CG or GMRES). There are three different ways of applying a preconditioner to a system

From the left:	$\mathbf{P}^{-1}\mathbf{Ax} = \mathbf{P}^{-1}\mathbf{b}$
From the right:	$\mathbf{AP}^{-1}\mathbf{y} = \mathbf{b}; \quad \mathbf{x} = \mathbf{P}^{-1}\mathbf{y}$
Centrally/split:	$(\mathbf{P} = \mathbf{LU};) \quad \mathbf{L}^{-1}\mathbf{AU}^{-1}\mathbf{y} = \mathbf{L}^{-1}\mathbf{b}; \quad \mathbf{x} = \mathbf{U}^{-1}\mathbf{y}$

3.6 Smoothers

Relaxation processes are inexpensive, approximate iterative methods for solving systems of equations (both linear and non-linear). Such methods are typically used for preconditioning Krylov solvers, which we will get back to in Section 3.7. *Smoothers* is another name for relaxation processes, and is the convention we will stick to in this thesis. Smoothers are a central ingredient in multigrid theory. Smoothing n times means taking n iterative steps (i.e., finding $\mathbf{x}_n$) in order to ”smooth out” the error in the solution at each iteration. Typical examples of smoothers are the Jacobi method and the Gauss-Seidel method, as well as variants of Incomplete LU factorization. These will be presented briefly below.

One advantage to smoothers is that they are very efficient for high-oscillatory (and local) error. On the other hand, convergence is slow when the error is a trend (i.e., repeating).

3.6.1 Jacobi method

Split $\mathbf{A}$ into its diagonal $\mathbf{D}$ and the remainder $\mathbf{R}$, so that $\mathbf{A} = \mathbf{D} + \mathbf{R}$. Note that $\mathbf{D}$ is easy to invert and rewrite the linear system

$$\begin{aligned} \mathbf{Ax} = \mathbf{b} \quad &\Rightarrow \quad (\mathbf{D}+\mathbf{R})\mathbf{x} = \mathbf{b} \quad \Rightarrow \quad \mathbf{D}^{-1}(\mathbf{D}+\mathbf{R})\mathbf{x} = \mathbf{D}^{-1}\mathbf{b} \\ &\Rightarrow \quad \mathbf{x} + \mathbf{D}^{-1}\mathbf{R}\mathbf{x} = \mathbf{D}^{-1}\mathbf{b} \quad \Rightarrow \quad \mathbf{x} = \mathbf{D}^{-1}(\mathbf{b} - \mathbf{R}\mathbf{x}) \\ &\Rightarrow \quad \mathbf{x}_{n+1} = \mathbf{D}^{-1}(\mathbf{b} - \mathbf{R}\mathbf{x}_n). \end{aligned} \tag{3.38}$$

where $\mathbf{x}_n$ is the n'th iteration We denote by $\mathbf{x}^*$ the exact solution of this fixed point scheme. Equation (3.38) is the Jacobi method. From here one can obtain the *weighted Jacobi method*, see Equation (3.39), by introducing a relaxation parameter ω,

$$\mathbf{x}_{n+1} = \omega\mathbf{D}^{-1}(\mathbf{b} - \mathbf{R}\mathbf{x}_n) + (1-\omega)\mathbf{x}_n. \tag{3.39}$$

3.6.2 Gauss-Seidel method

Devide $\mathbf{A}$ into its upper triangular part $\mathbf{U}$ and its strictly lower triangular part $\mathbf{L}$, so that $\mathbf{A} = \mathbf{L} + \mathbf{U}$. Triangular matrices are easily inverted using backward substitution. Again, we rewrite the system to obtain the Gauss-Seidel method:

$$\begin{aligned} \mathbf{Ax} = \mathbf{b} \quad &\Rightarrow \quad (\mathbf{L}+\mathbf{U})\mathbf{x} = \mathbf{b} \quad \Rightarrow \quad \mathbf{L}^{-1}(\mathbf{L}+\mathbf{U})\mathbf{x} = \mathbf{L}^{-1}\mathbf{b} \\ &\Rightarrow \quad \mathbf{x} + \mathbf{L}^{-1}\mathbf{U}\mathbf{x} = \mathbf{L}^{-1}\mathbf{b} \quad \Rightarrow \quad \mathbf{x} = \mathbf{L}^{-1}(\mathbf{b} - \mathbf{U}\mathbf{x}) \\ &\Rightarrow \quad \mathbf{x}_{n+1} = \mathbf{L}^{-1}(\mathbf{b} - \mathbf{U}\mathbf{x}_n). \end{aligned} \tag{3.40}$$

Gauss-Seidel performs better than Jacobi, but is also more expensive. An advantage to Jacobi and Gauss-Seidel smoothers is that they are problem-independent and easy to implement.

3.6.3 Incomplete LU factorization

Another popular choice for smoothing is *incomplete LU factorization* (ILU) [15, p.307-319]. From this process we obtain $\mathbf{A} = \mathbf{LU} - \mathbf{R}$, with matrices $\mathbf{L}$ and $\mathbf{U}$ being sparse lower and upper triangular, respectively, and $\mathbf{R}$ being the *residual* of the factorization. Standard LU factorization demands $\mathbf{LU} = \mathbf{A}$, but in incomplete LU factorization it is enough that $\mathbf{LU} \approx \mathbf{A}$. The system is then solved as follows

$$\begin{aligned} \mathbf{Ax} = \mathbf{b} \quad &\Rightarrow \quad (\mathbf{LU})\mathbf{x} = \mathbf{b} \quad \Rightarrow \quad \mathbf{L}(\mathbf{Ux}) = \mathbf{b} \\ &\Rightarrow \quad \text{Solve } \mathbf{Ly} = \mathbf{b} \qquad (3.41) \\ &\Rightarrow \quad \text{Solve } \mathbf{Ux} = \mathbf{y} \qquad (3.42) \end{aligned}$$

There are many ”types” of ILU factorization. The simplest case is called *zero fill-in* ILU factorization, abbreviated ILU(0), which requires $\mathbf{L}$ and $\mathbf{U}$ to have the same sparsity pattern as $\mathbf{A}$. If we choose the factorization such that $\mathbf{L}$ and $\mathbf{U}$ have the same non-zero structure as the lower and upper triangular parts of $\mathbf{A}$, respectively [15][3], we will find that the product $\mathbf{LU}$ typically ends up with more non-zero entries than $\mathbf{A}$ has. The phenomenon where LU factors of a sparse matrix are less sparse (i.e., more dense) than said matrix is referred to as *fill-in*. The extra entries come in the form of additional diagonals in the lower and upper parts of $\mathbf{LU}$ and are called *fill-in elements*. ILU factorizations allowing fill-in are called *level-based* and are abbreviated ILU(l), with l being the *level of fill*. That is, ILU(l) allows $\mathbf{L}$ and $\mathbf{U}$ to have the same sparsity pattern as $\mathbf{A}^{l+1}$, which is denser than $\mathbf{A}$ but still sparse. Consequently, level-based ILU factorizations are more accurate than ILU(0), as they among other things converge in fewer iterations and are generally more robust. However, they are also more expensive when it comes to computing the factors. Despite this fact, the higher accuracy is still favorable. Standard LU factorization can be thought of as ILU(∞).

3.7 Krylov spaces and Krylov solvers

The system in question may have been preconditioned beforehand, in which case we can view $\mathbf{A}$ and $\mathbf{b}$ as the ”updated” versions of their original selves ($\mathbf{A} := \mathbf{P}^{-1}\mathbf{A}$, $\mathbf{b} := \mathbf{P}^{-1}\mathbf{b}$). An iterative method is convergent if $\lim_{n\to\infty} \mathbf{x}_n = \mathbf{x}^*$. Most likely $\mathbf{x}_0 \neq \mathbf{x}^*$, so we look at the initial residual $\mathbf{r}_0 = \mathbf{b} - \mathbf{Ax}_0$, which is the error in $\mathbf{Ax}_0 = \mathbf{b}$. For each iterate, $\mathbf{x}_n$, the corresponding residual is given by $\mathbf{r}_n = \mathbf{b} - \mathbf{Ax}_n$. Convergence can therefore also be seen as $\lim_{n\to\infty} ||\mathbf{r}_n|| = 0$.

Recall from Section 3.6 the Jacobi iteration

$$\mathbf{x}_{n+1} = \mathbf{D}^{-1}(\mathbf{b} - \mathbf{Rx}_n) \qquad \text{or} \qquad \mathbf{Dx}_{n+1} = -\mathbf{Rx}_n + \mathbf{b}, \qquad (3.43)$$

[3] If $\mathbf{L}$ is unit lower triangular or $\mathbf{U}$ is unit upper triangular (that is, only 1’s on the diagonal), the decomposition is unique.

where $\mathbf{A} = \mathbf{D} + \mathbf{R}$. We may precondition our system by use of the Jacobi iteration, for which a natural choice of preconditioner is $\mathbf{P} = \mathbf{D}$. Noting that $-\mathbf{R} = \mathbf{D} - \mathbf{A}$, Equation (3.43) can be reformulated as follows

$$\begin{aligned}
&\mathbf{D}\mathbf{x}_{n+1} = (\mathbf{D} - \mathbf{A})\mathbf{x}_n + \mathbf{b} \\
\iff\ &\mathbf{P}\mathbf{x}_{n+1} = (\mathbf{P} - \mathbf{A})\mathbf{x}_n + \mathbf{b} \qquad (3.44)\\
\iff\ &\mathbf{P}\mathbf{x}_{n+1} = \mathbf{P}\mathbf{x}_n + \mathbf{b} - \mathbf{A}\mathbf{x}_n \\
\iff\ &\mathbf{P}\mathbf{x}_{n+1} = \mathbf{P}\mathbf{x}_n + \mathbf{r}_n \\
\iff\ &\ \ \mathbf{x}_{n+1} = \mathbf{x}_n + \mathbf{P}^{-1}\mathbf{r}_n \qquad (3.45)
\end{aligned}$$

The preconditioned iteration corrects the iterate $\mathbf{x}_n$ by the vector $\mathbf{P}^{-1}\mathbf{r}_n$. Equation (3.45) is a general ordinary preconditioned iteration and can also be obtained with different choices of matrix decomposition, for instance the Gauss-Seidel method.
Krylov solvers are often preconditioned, as it is a way of decreasing the number of iterations needed for convergence, but not always. For simplicity and without loss of generality, let us begin by assuming the linear system $\mathbf{A}\mathbf{x} = \mathbf{b}$ has been preconditioned beforehand with $\mathbf{P} = \mathbf{I}$ (the identity matrix) and setting the initial guess to be $\mathbf{x}_0 = \mathbf{b}$. Notice the pattern that unravels when we take a few steps of Equation (3.44)

$$\begin{aligned}
\mathbf{x}_1 &= (I - \mathbf{A})\mathbf{b} + \mathbf{b} &&= 2\mathbf{b} - \mathbf{A}\mathbf{b} \\
\mathbf{x}_2 &= (I - \mathbf{A})[2\mathbf{b} - \mathbf{A}\mathbf{b}] + \mathbf{b} &&= 3\mathbf{b} - 3\mathbf{A}\mathbf{b} + \mathbf{A}^2\mathbf{b} \\
\mathbf{x}_3 &= (I - \mathbf{A})[3\mathbf{b} - 3\mathbf{A}\mathbf{b} + \mathbf{A}^2\mathbf{b}] + \mathbf{b} &&= 4\mathbf{b} - 6\mathbf{A}\mathbf{b} + 4\mathbf{A}^2\mathbf{b} - \mathbf{A}^3\mathbf{b}
\end{aligned}$$

The approximate solutions $\mathbf{x}_n$ can be written as linear combination of the vectors $\mathbf{A}^i\mathbf{b}$, for $0 \le i \le n$. This leads to the definition of Krylov (sub)spaces.

Definition 3.7.1 (Krylov subspace)**.** Given a non-singular matrix $\mathbf{A} \in \mathbb{C}^{N\times N}$ and a vector $\mathbf{y} \neq \mathbf{0} \in \mathbb{C}^N$, the m'th Krylov (sub)space generated by $\mathbf{A}$ from $\mathbf{y}$ is

$$\mathcal{K}_m := \mathcal{K}_m(\mathbf{A}, \mathbf{y}) := \text{span}(\mathbf{y}, \mathbf{A}\mathbf{y}, ..., \mathbf{A}^{m-1}\mathbf{y}). \qquad (3.46)$$

Thus, in the case above, we could simply write $\mathbf{x}_n \in \mathcal{K}_{n+1}(\mathbf{A}, \mathbf{b})$. However, another way of writing this is $\mathbf{x}_n \in \mathbf{b} + \mathcal{K}_{n+1}(\mathbf{A}, \mathbf{b}(\mathbf{I} - \mathbf{A}))$, which will be explained shortly.

To illustrate the extension to the general case, consider again $\mathbf{A}\mathbf{x} = \mathbf{b}$ preconditioned by $\mathbf{P} = \mathbf{I}$, but this time let $\mathbf{x}_0$ be just some initial guess. Writing out the first three steps of Equation (3.44) gives:

$$\begin{aligned}
\mathbf{x}_1 &= \mathbf{x}_0 + \mathbf{b} - \mathbf{A}\mathbf{x}_0 &&= \mathbf{x}_0 + \mathbf{r}_0 \\
\mathbf{x}_2 &= \mathbf{x}_1 + \mathbf{b} - \mathbf{A}\mathbf{x}_1 = [\mathbf{x}_0 + \mathbf{r}_0] + \mathbf{b} - \mathbf{A}[\mathbf{x}_0 + \mathbf{r}_0] &&= \mathbf{x}_0 + 2\mathbf{r}_0 - \mathbf{A}\mathbf{r}_0 \\
\mathbf{x}_3 &= [\mathbf{x}_0 + 2\mathbf{r}_0 - \mathbf{A}\mathbf{r}_0] + \mathbf{b} - \mathbf{A}[\mathbf{x}_0 + 2\mathbf{r}_0 - \mathbf{A}\mathbf{r}_0] &&= \mathbf{x}_0 + 3\mathbf{r}_0 - 3\mathbf{A}\mathbf{r}_0 + \mathbf{A}^2\mathbf{r}_0
\end{aligned}$$

From this we see that $\mathbf{x}_n \in \mathbf{x}_0 + \mathcal{K}_n(\mathbf{A}, \mathbf{r}_0)$. If nothing is assumed about the preconditioner $\mathbf{P}$ then, strictly speaking, we get $\mathbf{x}_n \in \mathbf{x}_0 + \mathcal{K}_n(\mathbf{P}^{-1}\mathbf{A}, \mathbf{P}^{-1}\mathbf{r}_0)$, but one can also define $\mathbf{A} := \mathbf{P}^{-1}\mathbf{A}$ as previously mentioned.

Lemma 3.7.1 (Grade of $\mathbf{y}$)**.** *The grade of* $\mathbf{y}$ *is a non-negative integer* $\nu := \nu(\mathbf{y}, \mathbf{A})$ *such that*

1. *dim* $\mathcal{K}_m(\mathbf{A}, \mathbf{y}) = \min(m, \nu)$*,*
2. $\nu = \min\{m \mid \mathbf{A}^{-1}\mathbf{y} \in \mathcal{K}_m(\mathbf{A}, \mathbf{y})\}$*.*

Corollary 3.7.1.1. $\mathcal{K}_\nu(\mathbf{A}, \mathbf{y})$ *is the smallest* $\mathbf{A}$*-invariant subspace that contains* $\mathbf{y}$*.*

By $\mathbf{A}$-invariant we mean that $\forall\ \mathbf{z} \in \mathcal{K}_\nu(\mathbf{A}, \mathbf{y})$, the matrix-vector product $\mathbf{Az} \in \mathcal{K}_\nu(\mathbf{A}, \mathbf{y})$ too.

Corollary 3.7.1.2. *Let* $\mathbf{x}^*$ *be the solution of* $\mathbf{Ax} = \mathbf{b}$*,* $\mathbf{x}_0$ *the initial approximation of* $\mathbf{x}^*$*,* $\mathbf{r}_0 = \mathbf{b} - \mathbf{Ax}_0$ *the corresponding residual, and* $\nu = \nu(\mathbf{r}_0, \mathbf{A})$*. Then*

$$\mathbf{x}^* \in \mathbf{x}_0 + \mathcal{K}_\nu(\mathbf{A}, \mathbf{r}_0).$$

The following definition of Krylov space solvers is taken from one stated by Martin H. Gutknecht [8] in a seminar on applied mathematics:

Definition 3.7.2 (Krylov space solvers)**.** A standard Krylov space methods for solving a linear system $\mathbf{Ax} = \mathbf{b}$, i.e., a Krylov space solver, is an iterative method starting from some initial approximation, $\mathbf{x}_0$, and the corresponding residual, $\mathbf{r}_0 = \mathbf{b} - \mathbf{x}_0$, and generating iterates $\mathbf{x}_n$ for all, or at least most, n, until it possibly finds the exact solution, such that

$$\mathbf{x}_n - \mathbf{x}_0 = q_{n-1}(\mathbf{A})\mathbf{r}_0 \in \mathcal{K}_n(\mathbf{A}, \mathbf{r}_0).$$

Here, q_{n-1} is a polynomial in $\mathbf{A}$ of exact degree $n-1$. For some n, $\mathbf{x}_n$ may not exist or q_{n-1} may have lower order.

Krylov space methods are among the most important classes of numerical methods to this day. One of many reasons for this is the central role they play in the solving of sparse linear systems. Some examples that we will present briefly below are the Conjugate Gradient method (CG) and the Generalized Minimum Residual Method (GMRES).

CG is a numerical method for solving symmetric positive definite (SPD) matrices.This method is a realization of an orthogonal projection technique onto Krylov subspace $\mathcal{K}_m(\mathbf{A}, \mathbf{r}_0)$, for initial residual $\mathbf{r}_0$. Further details can be found in Y. Saad's book "Iterative Methods for Sparse Linear Systems" [15, p.196]. The BiConjugate Gradient stabilized method (BiCGstab) is a variant of CG that can be seen as a combination of GMRES and the BiConjugate Gradient method (BiCG). BiCGstab(l) is a generalization of BiCGstab.

GMRES is an iterative method for the numerical solution of non-symmetric linear systems. It is a projection method based on taking $\mathcal{K} = \mathcal{K}_m$ and $\mathcal{L} = \mathbf{A}\mathcal{K}_m$, where $v_1 = \frac{\mathbf{r}_0}{\|\mathbf{r}_0\|_2}$. This technique minimizes the residual norm over all vectors in $\mathbf{x}_0 + \mathcal{K}_m$. Further details about GMRES can also be found in Y. Saad's book [15, p.171]. Two variants of GMRES are the flexible GMRES (FGMRES) and LGMRES(m, l), which uses the last l error approximations in addition to a Krylov subspace of dimension m.

Chapter 4

Multigrid methods

When applied to linear systems of increasing size arising from discretized PDEs, the convergence of preconditioned Krylov subspace methods will tend to slow down considerably [15]. As a result of this, as well as of the fact that the sheer problem size causes high increase in the number of operations necessary at each step, the efficiency becomes severely impaired. This is where multigrid (MG) methods come in. In theory, multigrid methods can achieve convergence rates that are independent of the mesh size and are, in general, superior performance-wise compared to standard preconditioned Krylov methods. However, where the latter type of methods attempts to be general-purpose, multigrid methods may require implementations that are problem-specific, in which case they can prove both complicated and time-consuming.

To understand some of the motivation for multigrid methods, we will briefly explain about error modes. *Error modes* are the eigenvectors of an iteration matrix which correspond to large eigenvalues. A (normal) mode is a standing wave state of excitation, in which all the components of a system are oscillating at one common frequency (though with different amplitude), giving the impression of a ”flip-flopping” sine wave. To illustrate, picture the string of a guitar being plucked and vibrating up and down – that is what we call a (fundamental) normal mode. When analyzing multigrid for some system of equations, $\mathbf{Ax} = \mathbf{b}$, it is common to begin by considering the case $\mathbf{Ax} = \mathbf{0}$, where we know that the (trivial) solution is $\mathbf{x} = \mathbf{0}$. Expressing the initial iterate, $\mathbf{x}_0$, using sine modes of the form $\mathrm{e}(x) = \sin(k\pi x)$, with different values for the wave number k, we will find that the solution is equal to the error. That way, we can actually observe the error, not just the residual error.

The different sine modes have the same norm but they become increasingly oscillatory as k increases. Smoothing is effective for high-frequency errors (that is, large k), but becomes less effective as the frequency decreases.

4.1 Brief outline of the idea

Using relaxation techniques and exploiting discretizations with different grid sizes, multigrid methods aim to obtain optimal convergence for a given problem. Despite converging slowly for certain problems, relaxation-type iterative processes can be quite effective at damping error or residual components associated with oscillatory/high-frequency modes. On the other hand, the same processes will struggle when attempting to dampen components associated with smooth/low-frequency modes. This is the cause of the slow-down that can be observed in basic iterative schemes such as relaxation methods. However, many of these low-frequent modes can be mapped naturally into high-frequent modes on a coarser grid, where relaxation once again can dampen them effectively. It is from here the idea of moving from fine (original) grid to coarser grids comes, creating a hierarchy of grids, in order to eliminate all error components. At a coarse enough grid, we are able to solve the problem exactly and, moving back up, use this solution to find the solutions on the finer grids. For more details, the reader is referred to Yousef Saad's book [15].

In short, the multigrid idea can be summarized as follows [13]:

1. Generate a set of successively coarsened levels
2. Generate prolongation (interpolation) and restriction operators to map between levels
3. Using some cycling strategy, move between levels while applying approximate solvers at each level
4. Solve the coarsest level exactly
5. Check convergence and if necessary go back to step 3

4.2 Ingredients in multigrid

We will now go through some of the more important ingredients and aspects of multigrid. That includes coarse grids, interpolation and restriction, multigrid cycles and their computational cost.

4.2.1 Coarser grids and interpolation

The idea behind multigrid is to deal with each component of error at a level where it appears high-frequent to the smoother. The approach is to recursively coarsen the grid until the coarsest grid has only one single cell. E.g., for a 32×32 grid, we coarsen to 16×16, then to 8×8, 4×4, 2×2 and finally 1×1 – a total of six grids (see Figure (4.1)). Often, we skip the coarsest of these levels, because we have reached a sufficiently coarse level (where the problem can be solved exactly). Typically we look at some multigrid level, $l = m$, or perhaps at a series of multigrid levels, $m, m-1, ...$, where coarser levels are associated with lower numbering. When considering only two consecutive levels at a time, it is customary to use indices h and H (instead of m and $m-1$) to distinguish fine and

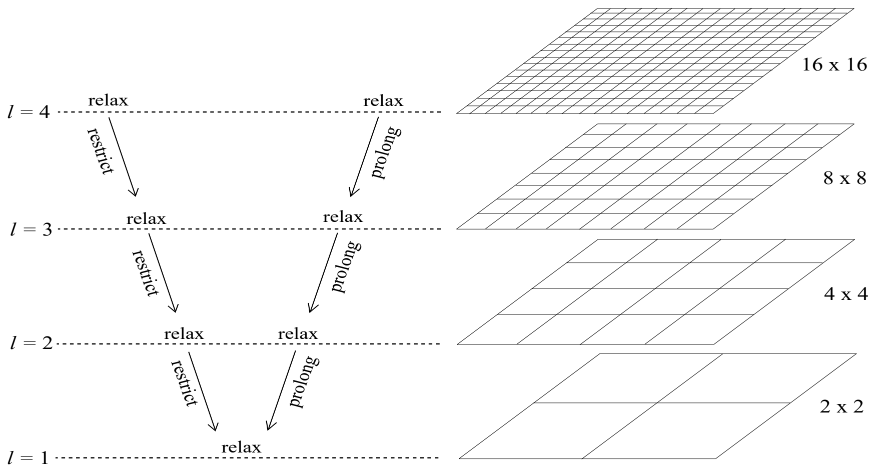


Figure 4.1: Grids of different levels of coarsening, excluding the finest (32×32) and the coarsest (1×1) level.

coarse level quantities, respectively.

Interpolation techniques

In order to move up and down between different levels we define a *prolongation operator* and a *restriction operator*. The prolongation operator I_H^h (also referred to as simply *interpolator*) moves from a coarse grid to a finer grid, while the restriction operator I_h^H moves from a fine grid to a coarser grid. Let $\mathbf{x}_h$ and $\mathbf{x}_H$ denote the fine solution and the coarse solution, respectively, with

$$\mathbf{x}_\mathrm{h} = I_\mathrm{H}^\mathrm{h} \mathbf{x}_\mathrm{H}, \qquad \mathbf{x}_H = I_h^H \mathbf{x}_h.$$

By default we assume that these operators are each other's transpose, i.e., $(I_H^h)^T = I_h^H$, unless otherwise noted. Furthermore, we require two things

1. The restriction operator from a fine grid with M cells to a coarse grid with N cells (if the grid data is on column vector form) will be an $M \times N$ matrix. The entry with coordinates (i, j) corresponds to the relative weight that fine cell i receives from coarse cell j.
2. $\sum_{j \in (1,N)} (I_H^h)_{ij} = 1, \; \forall \, i \in (1, M)$ *(partition of unity)*

The coarse scale system will be constructed using the *Galerkin principle*

$$\mathbf{A}_H = I_h^H \mathbf{A}_h I_H^h = (I_H^h)^T \mathbf{A}_h I_H^h,$$

where $\mathbf{A}_H$ and $\mathbf{A}_h$ are the coefficient matrices obtained from the coarse grid and the fine grid, respectively.

The term *coarsening* is used for this process of picking the coarse grid and defining interpolation [1].

4.2.2 Multigrid cycles and computational cost

Define:

- $S(\mathbf{A}_h, \mathbf{b}_h, \mathbf{x}_h)$: a single smoother cycle applied to initial guess $\mathbf{x}_h$
- $\mathbf{d}_h = \mathbf{b}_h - \mathbf{A}_h\mathbf{x}_h$: the defect

Multigrid cycle algorithm

Denote the multigrid cycle function by "mgc". The multigrid cycle algorithm is as follows:

```
function mgc($\mathbf{A}_h, \mathbf{b}_h, \mathbf{x}_h, l, \gamma, \nu_{pre}, \nu_{post}$)
  for $i \leftarrow 1, ..., , \nu_{pre}$
    $\mathbf{x}_h \leftarrow S(\mathbf{A}_h, \mathbf{b}_h, \mathbf{x}_h)$
  end
  $\mathbf{d}_h \leftarrow \mathbf{b}_h - \mathbf{A}_h\mathbf{x}_h$
  $\mathbf{d}_H \leftarrow I_h^H\mathbf{d}_h$
  $\mathbf{A}_H \leftarrow I_h^H\mathbf{A}_h I_H^h$
  $\mathbf{c}_H \leftarrow 0$
  if $l = 0$
    $\mathbf{c}_H \leftarrow \mathbf{A}_H^{-1}\mathbf{d}_H$
  else
    for $i \leftarrow 1, ..., \gamma$
      $\mathbf{c}_H \leftarrow$ mgc($\mathbf{A}_H, \mathbf{b}_H, \mathbf{x}_H, l-1, \gamma, \nu_{pre}, \nu_{post}$)
    end
  end
  $\mathbf{x}_h \leftarrow \mathbf{x}_h + I_H^h\mathbf{c}_H$
  for $i \leftarrow 1, ..., \nu_{post}$
    $\mathbf{x}_h \leftarrow S(\mathbf{A}_h, \mathbf{b}_h, \mathbf{x}_h)$
  end
end
```

The multigrid cycle algorithm begins by pre-smoothing the approximate solution. Then the defect is calculated and restricted to the coarser scale. Now we can obtain the coarse system. At the coarsest level the system is solved exactly. For the other levels we solve the defect approximately and precede to the next level. Then the solution is updated using correction, before it is post-smoothed as the last step of the algorithm.

The first and last `for`-loops are for pre- and post-smoothing, respectively. The cycle index is denoted by γ and indicates the type of cycle. We will limit ourselves to V cycles ($\gamma = 1$), W cycles ($\gamma = 2$) and F cycles (a hybrid between W and V cycles). The basic structure of these cycles can be found in [1, p.21]. The choice of cycle can drastically alter the convergence. A high cycle index tends to give faster convergence, but is more expensive because of the cost of extra computational effort.

We assume that multigrid cycles are best treated recursively, and set up the cost for each level:

$$W_{l+1} = W_{l+1}^{l} + \gamma_l W_l, \qquad \text{with} \quad W_1 = W_1^0 + W_0 \tag{4.1}$$

where

- γ_l: cycle index at level l (if the cycling strategy is constant, then $\gamma_l = \gamma$
- W_l: solution of the defect equation a level l
- W_{l+1}^{k}: work done going from level $l+1$ to level l (including smoothers at level $l+1$)

Recursively, we can rewrite this as

$$W_{l+1} = W_{l+1}^{l} + \gamma_l (W_l^{l-1} + \gamma_{l-1} W_{l-1}) = W_{l+1}^{l} + \gamma_l W_l^{l-1} + \gamma_l \gamma_{l-1} W_{l-1}$$

Assuming constant cycling strategy, this becomes

$$\begin{aligned} W_{l+1} &= W_{l+1}^{l} + \gamma W_l^{l-1} + \gamma^2 W_{l-1} && (4.2)\\ &= W_{l+1}^{l} + \gamma W_l^{l-1} + \gamma^2 (W_{l-1}^{l-2} + \gamma W_{l-2}) && (4.3)\\ &= W_{l+1}^{l} + \gamma W_l^{l-1} + \gamma^2 W_{l-1}^{l-2} + \gamma^3 W_{l-2} && (4.4) \end{aligned}$$

The sum over all levels for a cycle with m levels in total is

$$W_m = \sum_{i=1}^{m} \gamma^{m-i} W_i^{i-1} + \gamma^{m-1} W_1^0 \tag{4.5}$$

Let N_l be the number of nodes at level l and d be the number of dimensions. Assuming that the work is a (small) constant multiplied by a number of nodes (i.e., $W_l \leq cN_l$) and letting the coarsening be done by a factor of 2 in each spatial direction (i.e., $N_l = \frac{N_m}{2^{d(m-l)}}$), the cycle cost can be evaluated as

$$W_m = cN_m \sum_{i=1}^{m} \frac{\gamma^{m-i}}{2^{d(m-i)}} + \gamma^{m-1} W_1^0. \tag{4.6}$$

where the last term is negligible.

4.3 Types of multigrid

The basic types of multigrid include *full* multigrid (FMG), *geometric* multigrid (GMG) and *algebraic* multigrid (AMG) Due to the fact that the codes for the upcoming experiments only deploy AMG, we will not go into detail about the two other types. Nevertheless, a short review of the ideas behind FMG and GMG will precede the introduction to AMG.

Full multigrid

Using the multigrid cycle algorithm above in a nested setting, the algorithm begins at the coarsest level in the grid hierarchy (where iterations are expensive) and works its way up to the finest level.

Geometric multigrid

Geometric multigrid applies only to systems obtained from the discretization of PDEs. Here, the grid hierarchy is defined based on the geometrical problem. Then the problem is discretized on this series of grids. (GMG can be applied to both implicit and explicit solvers.) When working with unstructured grids, this approach becomes very challenging. In the presence of anisotropy, coarsening should be ”aligned” with the strongest interface connnections, by which we mean the direction in which the permeability is largest. This makes the multigrid converge better.

4.3.1 Algebraic multigrid

The approach in algebraic multigrid (AMG) is based on the algebraic system $\mathbf{A}\mathbf{x} = \mathbf{b}$, rather than on the physical PDEs on a geometry where we have the notion of grids (such as in GMG). AMG can be applied to any system of equations. This type of multigrid was developed to solve matrix equations using the principles of usual multigrid methods, but can be used in many kinds of problems where the application of standard multigrid is difficult or impossible. In AMG the multilevel hierarchy is constructed without using any information about the PDE or the geometrical problem. Instead of solving on coarser and coarser grids, we generate a sequence of smaller and smaller systems of equations which will essentially play the same role as coarse grid equations. Among other things, AMG should therefore work better for complicated geometries (like unstructured grids) and highly anisotropic problems than does GMG.

The AMG approach is to split the grid into two parts at each level: ”fine cells” (F) and ”coarse cells” (C), with $F \cap C = \emptyset$. All cells start off as ”undecided” (U). Two requirements for the coarsening are:

1. The number of coarse cells at each level should be limited and decrease for every coarser level, in order to reach the sufficiently coarse level (where all cells are deemed ”fine”) as fast as possible.

2. The distribution of coarse cells should be relatively uniform, so that every fine cell is close enough as to be associated with a coarse cells. The reason why will soon become apparent.

A central idea of AMG coarsening is to split the *matrix connections* (from $\mathbf{A}$) into strong and weak couplings based on their *connection strengths*. This is to deal with problems with varying coefficient matrix due to anisotropy and/or a complicated grid. Note that there can be a coupling between two cells without these cells being neighbors in a physical sense.

Couplings have to do with how cells are connected to each other with respect to the non-zero entries of $\mathbf{A}$, which might be easier to see by looking at the graph of $\mathbf{A}$. From now on, the term ”neighbors” is used in a graph sense.

Connections

A *strong connection* from cell i to cell j (i.e., i strongly connected to j) is defined by

$$|a_{ij}| \geq \varepsilon_{str} \max_{k \neq i} |a_{ik}| \tag{4.7}$$

for some fixed $\varepsilon_{str} \in (0, 1)$. A reasonable default value is $\varepsilon_{str} = 0.25$, which is for instance used for Ruge-Stüben coarsening (the ”standard” coarsening). A cell might be strongly connected to another cell without this being true in return. Define:

- N_i: index set of all couplings for cell i
- $S_i = \{j \in N_i \mid i \text{ strongly connected to } j\}$: all neighbors, j, of a cell i that i is strongly connected to
- $S_i^T = \{i \in S_j\}$: all neighbors, j, of cell i that see i as a strong neighbor (i.e., are strongly connected to i)
- $\lambda_i = |S_i^T \cap U| + 2|S_i^T \cap F|$: measure of importance

The measure of importance for a particular cell increases for every undecided or (especially) fine neighbor that is strongly connected to that cell. At each step, this measure is calculated for every cell, and the cell i with largest λ_i is then added to coarse set C. Simultaneously, any cells that depend strongly ($j \in S_i^T$) on this coarse cell, are added to the fine set F. This process is repeated until all cells have measure of importance equal to zero. The relation between the fine grid (FG) and the coarse grid (CG)[1] satisfies the following:

- Each cell in FG belongs to one, and only one, block in CG
- Each block in CG consists of a connected subset of cells from FG
- CG is defined by a partition vector, π, where $\pi(i) = j$ if cell i in FG belongs to block j in CG

As mentioned above, the standard type of coarsening is the Ruge-Stüben coarsening [16, p.131]. Other popular choices are aggregation, with or without smoothed iterates [7].

Interpolation

In geometric context smooth (or low-frequent) error modes are defined in terms of spatial variations, whereas in algebraic context ”smooth error” is defined to be error where the smoother has poor convergence properties. The latter generally has very small residual compared to the magnitude of the error itself, i.e., $||\mathbf{Ae}|| \ll ||\mathbf{e}||$, which is approximately

$$a_{ii}e_i + \sum_{j \in N_i} a_{ij} e_j \approx 0 \tag{4.8}$$

[1]CG in this setting must not be confused with the Conjugate Gradient method.

for very smooth error. Our goal is to interpolate smooth error modes as exactly as possible between levels, such that they can be efficiently dealt with at the level where they appear non-smooth. To be able to interpolate correctly from coarse to finer grid it is important that no additional errors are introduced in the process. By defining the following

- w_{ij}: interpolation weights
- $P_i = C \cap N_i$: interpolatory set

we can express the interpolation formula

$$e_i = \sum_{k \in P_i} w_{ik} e_k, \quad i \in F. \tag{4.9}$$

Direct interpolation accounts for strong connections to coarse cells only, and calculates the interpolation values e_i by splitting the sum in Equation (4.8) into coarse and non-coarse variables and assuming the value over the weak connections to be the value at the node itself. An improvement of this technique is *standard interpolation*, which accounts for strong connections to both coarse and fine cells. It is derived similarly to direct interpolation, except for the elimination of all fine strong connections beforehand, using

$$e_j = -\sum_{k \in N_j} \frac{a_{jk} e_k}{a_{jj}}, \quad j \in F_i^s. \tag{4.10}$$

Aggregation-based algebraic multigrid

Also known as *agglomeration-based* algebraic multigrid, this is an alternative approach to AMG that exploits the cheapness and simplicity of constant interpolation,

$$(I_H^h)_{ij} = \begin{cases} 1, & \text{if fine cell } i \text{ is inside coarse cell } j \\ 0, & \text{otherwise,} \end{cases}$$

in the normally expensive/slow setup phase. Instead of distinguishing between fine and coarse unknowns, the unknowns are aggregated into categories so that a single coarse unknown corresponds to a subset of fine cells. An example is AGMG, where the new ”measure of importance” is $\lambda_i = |S_i \cap U|$ for $i \in U$, which is the number of undecided neighbors that cell i is strongly connected to. For every step the algorithm selects the cell with the smallest measure, finds the strongest connection from that cell and, if that connection is deemed ”strong”, aggregates those two cells. Moreover, if the next strongest connection is not deemed ”strong”, the cell in question is defined as a *singleton*.

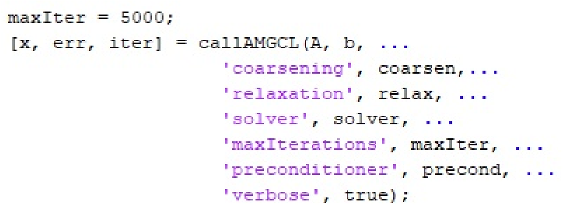
Chapter 5

Preparation and experimental set-up

Having introduced the basics, we are now ready to take a look at what makes up the different iterative solvers we are going to test. Following this is a brief overview of the test models on which the experiments will be conducted. Then we will explain the procedure of selecting solvers (or, combinations) for further analysis. Lastly, we ”warm up” to the upcoming experiments by looking at some of the results obtained from applying all combinations on the different models.

5.1 The iterative solvers

For solving the linear systems that arise from the TPFA discretization of different model problems, we use the AMG library called AMGCL. For the parallelized parts we have set 2 as the maximum number of computational threads. As mentioned in the introduction, we will hand AMGCL four things in addition to the linear system: a preconditioner, a Krylov subspace solver, a coarsening strategy and a smoother.



```
maxIter = 5000;
[x, err, iter] = callAMGCL(A, b, ...
                    'coarsening', coarsen,...
                    'relaxation', relax, ...
                    'solver', solver, ...
                    'maxIterations', maxIter, ...
                    'preconditioner', precond, ...
                    'verbose', true);
```

The solver outputs three things: the solution `x`, the residual `err` and the number of iterations `iter` used to reach this solution. Essentially, each combination of one preconditioner, one Krylov solver, one type of coarsening and one smoother is one linear iterative solver. The choices are as follows:

Preconditioners:

1. Relaxation
2. Algebraic multigrid

Krylov subspace solvers:

1. Conjugate gradient, CG [15, p.196]
2. Biconjugate gradient stabilized, BiCGstab [15, p.244]
3. BiCGstab(l)
4. Generalized minimal residual, GMRES, for solving large sparse non-symmetric linear systems [15, p.171]
5. A modified variant of GMRES, LGMRES(m, l) [3]
6. Flexible GMRES, FGMRES [15, p.287]
7. Induced dimension reduction method, IDR(s)

Coarsening strategies:

1. Ruge-Stüben coarsening (R-S) [16, p.131]
2. Aggregation (aggr.)
3. Aggregation with smoothed iterates (sm. aggr.) [7]
4. Smoothed aggregation with energy minimization (sm. aggr. emin.)

Smoothers:

1. SPAI0 [10]
2. SPAI1
3. Gauss-Seidel (G-S)
4. ILU(0) [15, p.307]
5. ILU(l), level-based [15, p.311]
6. ILUT, threshold-based [15, p.321]
7. Damped Jacobi (DJ)
8. Chebyshev

The maximum number of iterations was set to 5000. Any combinations that exceeded 5000 iterations were judged as ”not converging”. Likewise, if combinations reached a solution in less than 5000 iterations, but the residual $\mathbf{b} - \mathbf{A}\mathbf{x}_i$ was greater than or equal to 10^{-6} (which was the chosen tolerance), they were also judged as ”not converging”. Whenever relaxation is chosen as preconditioner, no multigrid is used in the computations.

5.2 The reservoir models

We are now almost ready to put our solvers to the test. In this section we will define and explain the models we will base our experiments on.

5.2.1 Simple case

As a warm-up of sorts to the more interesting cases, we consider a regular grid that is $m \times m$ in the horizontal xy-plane and has height h. The height is typically set to equal 1, as m is the variable which this model is mainly being tested for. The rock consists of void only ($\varphi = 1$) and is given a homogeneous permeability of 1 D. The model has an injection well in one corner and a production well in the diagonally opposite corner, both vertical. The injection well is rate-controlled, with an injection rate of 1 m^3/s (signifying the volume injected per second). The production well is controlled by bottom-hole pressure (BHP) with a value set to 0 Pa.

5.2.2 SPE10

The SPE10 benchmark has a regular grid, with dimensions $60 \times 220 \times 85$, that is, 85 layers with 60 blocks in x-direction and 220 blocks in y-direction each. The challenging part is the permeability. The SPE10 reservoir consists of two formations: Tarbert (layers 1-35) and Upper Ness (layers 36-85). Both formations exhibit large permeability variations, but in addition Upper Ness has much more heterogeneous structure. The porosity field shows strong correlation to the permeability, with 2,5% of the blocks being considered inactive as they have zero porosity and, thus, do not transmit any fluids. The model has four vertical production wells, one in each corner, and one vertical injection well in the center. The center well injects water at a rate of 5000 stb/d, and the reservoir is produced from the corner wells with a BHP of 4000 psi.

5.2.3 Olympus

The Olympus model is a synthetic reservoir model developed as a benchmark study for field development optimization, satisfying a number of complexity criteria. It is 9×3 km, bounded by a boundary fault on one of its sides, and 50 m thick. The model has been divided into 16 horizontal layers and contains approximately 341 728 grid cells in total, 192 750 of which are active. Further details can be found in this document[6]. A collection of 50 different Olympus cases (so-called realizations) was generated as a part of the optimization challenge, though only a selection of these will be tested in this thesis. All realizations share the same grid, faults and oil-water contact. The difference from realization to realization lies within the four uncertain properties

1. Porosity
2. Permeability
3. Net-to-gross (NTG)
4. Initial water saturation

5.3 Experimental procedure

Most of the code, especially the heavy kind, was run on the NTNU computing cluster called Markov. Markov is used by many people and has 28 processors with 28 threads each (two per core), a total of 784 threads. If there are not enough threads for all assignments to be executed, some of them will have to be set on hold until more threads are available. This kind of delay might affect the time used to run a program, but also the number of iterations.

In the upcoming chapters we will consider two ”experiments” on which discussion and conclusions will be based:

1. Run-time as a function of the number of grid cells
2. Mean values and corresponding standard deviations

Note that results, analysis and discussion will not be kept entirely separate. As we observe behaviour in the results, it follows naturally to comment on these observations and, perhaps, attempt to reflect on what might explain such behaviour. However, another chapter will follow that is dedicated to discussion and overview.

The experiments are based on results[1] obtained from running the main code on different models with different properties and variables for all combinations. In order to make the combinations more visible in plots, lines are drawn between result points that correspond to the same Krylov solver (represented by different colors) and coarsening (represented by different line styles). The smoothers are placed on the x-axis, as the variable against which time or number of iterations is plotted. Each run uses only one preconditioner.

For the first experiment, each code scenario of interest will be executed multiple times, out of which the results from the ”best run” (i.e., with the overall shortest run-times and fewest iterations) are selected for plotting and analysis. An improvement to this approach would be to look at the average run-time and average number of iterations over a larger number of runs. However, what is important to us is not to obtain the exact values produced by each solution method, but that the combinations[2] and `mldivide` perform similarly relative to one another. Information on which combinations perform well, which do not, in what cases, and if they are better or worse than direct methods, is what we are interested in.

The performance of combinations (i.e., of the iterative linear solvers we are testing) will be based on their respective run-times and iteration counts. We seek iterative solvers with traits such as short run-time and a low iteration count (i.e., convergence in few iterations), preferably both – this is deemed good performance. Each combination will be judged on this basis and compared to the other combinations.

When processing the results and looking for trends and interesting behaviour we will have

[1]The results from a run on some test case is a list consisting of run-time, number of iterations, residual and a flag evaluating convergence for every iterative solver that has been applied to that case.

[2]Recall that ”combination” is shorthand for ”combination of one preconditioner, one Krylov solver, one coarsening and one smoother”.

to limit ourselves to a smaller subset of combinations, since there are 224 combinations in total for each of the two possible preconditioners. Since combinations that do not perform well in any way are not of much interest, except for when discussing which combinations were not preferable and why, we will pick out a selection of the best-performing combinations for every test case where this is necessary.

The selections will consist of combinations that performed well with respect to both run-time (preferably, better than `mldivide`[3]) and number of iterations. The selection process begins by looking at the results plots for run-time and number of iterations to see which combinations converged fastest and/or after fewest iterations. I make a list of the possible candidates, ranking them by their relative performance both time-wise and iteration-wise, before summing up the scores. The better the combination, the smaller the score. Needless to say, only combinations that have reached convergence are selected.

5.4 Typical results from the different test models

The simple case will only be considered in this section. It is included for purely illustration purposes and as a ”warm-up” to the SPE10 model and the Olympus model.

5.4.1 Simple case

This model was tested for a handful of values of m ranging from 10 to 1000. Some tests were merely for observational purposes, since small values of m often make for uninteresting and unrealistic cases and many of the larger values had problems with convergence. The latter, however, may have to do with lack of computer power, not necessarily the problem size itself.

We will include results for the runs where $m = 100$ and $m = 500$. Preconditioning with relaxation proved to be inferior to AMG, as expected. Only a small subset of the combinations with relaxation converged, and those that did required large numbers of iterations. Because of this, only results for AMG as preconditioner will be plotted. Figures 5.1 and 5.2 show that for the different smoothers, there are many combinations that gives simmilar run-times. However, when it comes to iterations, ILUT, ILU(l) and Chebyschev gives the best results. We also note that the solvers BiCGstab and LGMRES in many cases performs quite poorly, not even reaching convergence. Note that for this small case, `mldivide` has better run-time than all of the combinations.

For the 500$\times$500 simple case, we obtain simmilar results as evident in figures 5.3 and 5.4, except that the overall run time is increased. We now also see that for the smoothers ILU0 and ILU(l) some of the combinations outperform `mldivide`. We thus suspect (and expect) that as the number of grid cells increases, the iterative solvers will perform better than standard direct solvers.

[3]It does not make sense to attempt comparison between direct and iterative solvers based on number of iterations, since ”iterations” are not defined for direct solvers. Therefore, the iterative solvers we are testing will only be compared to `mldivide` with respect to run-time.

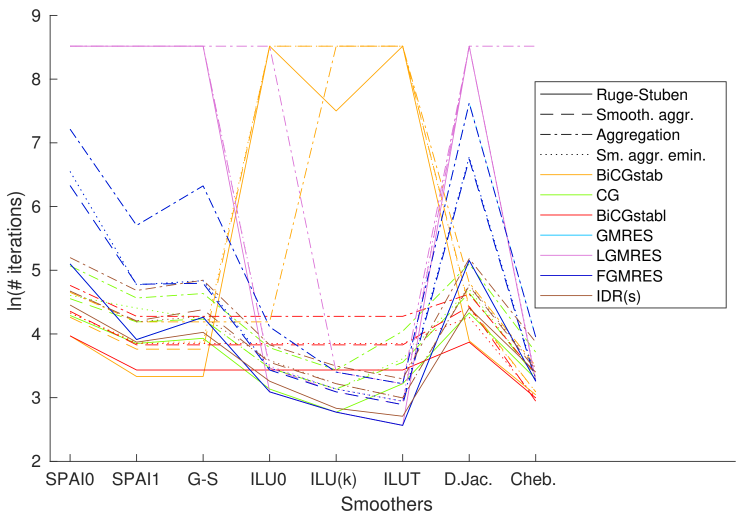


Figure 5.1: Simple case for a 100×100 grid: Number of iterations for all combinations with AMG as preconditioner

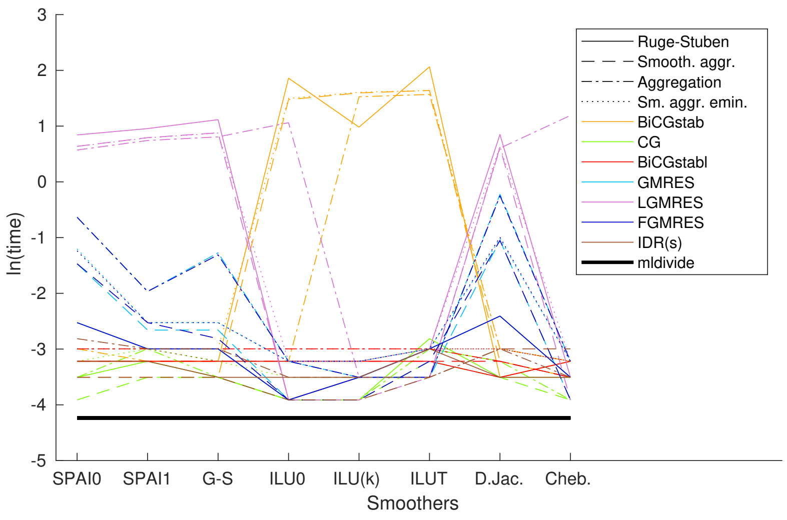


Figure 5.2: Simple case for a 100×100 grid: Run-time for all combinations with AMG as preconditioner

5.4.2 SPE10

For the SPE10 model, we will consider the results from layers 1-35 for both preconditioners. When it comes to the number of iterations (see Figure 5.6), notice that with relaxation

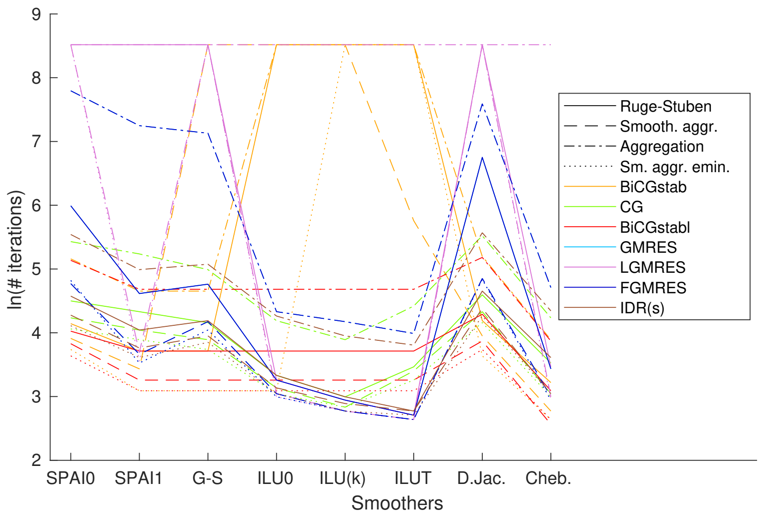


Figure 5.3: Simple case for a 500×500 grid: Number of iterations for all combinations with AMG as preconditioner

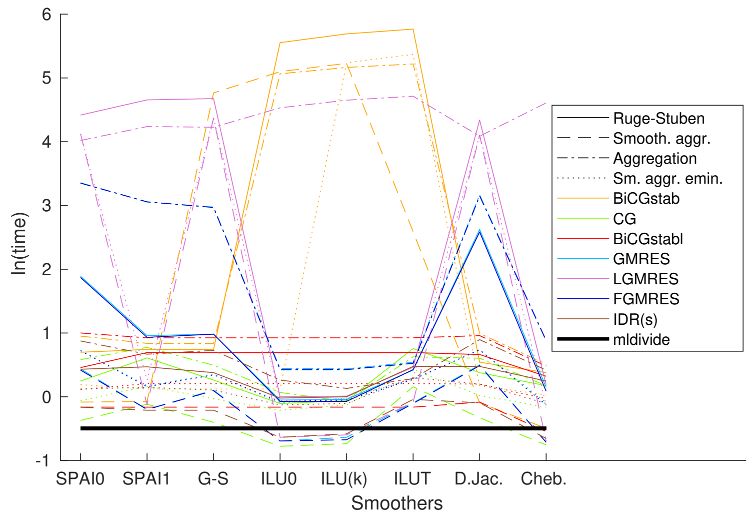


Figure 5.4: Simple case for a 500×500 grid: Run-time for all combinations with AMG as preconditioner.

as preconditioner the results for all four types of coarsening overlap, as opposed to the results from when AMG is used as preconditioner. As was explained in Section 4, the coarsening strategies have to do with the process of moving between different multigrid levels. Whenever relaxation is used as preconditioner, none of the multigrid concepts applies and thus the combinations will perform independent of the choice of coarsening. It is therefore enough to consider only one (arbitrary) coarsening strategy. The only thing that might vary is the run-time, but this kind of variation may very well be caused by random factors, such as delay when running the calculations on Markov.

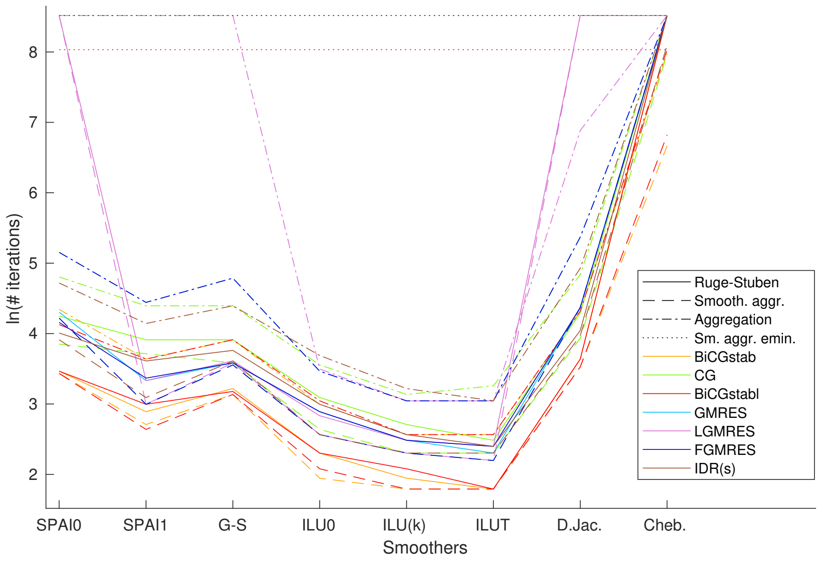


Figure 5.5: Layers 1-35 of SPE10: Number of iterations for all combinations with AMG as preconditioner.

It appears, from Figures 5.5 and 5.6, that preconditioning with AMG is superior to preconditioning with relaxation, as it results in the majority of combinations reaching convergence in less than 150 iterations, while with relaxation the majority require many more than 150 iterations. Additionally, we see from Figures 5.7 and 5.8 that the combinations with AMG are much faster then those with relaxation. Notice how almost none of the latter combinations converge faster than `mldivide`. On the other hand, it appears that most of the combinations using AMG are faster than `mldivide`. A reason why some of the combinations require more time to converge than `mldivide` might be that the problem size is too small, or that these combinations simply do not work well on the given problem.

Letting the result speak for us, we can conclude with AMG being superior to relaxation when it comes to preconditioning.

5.4.3 Olympus

For the Olympus model we will consider Case 1 (also known as Realization 1). Because of unforeseen technical issues when attempting to run code in the final stage of writing this

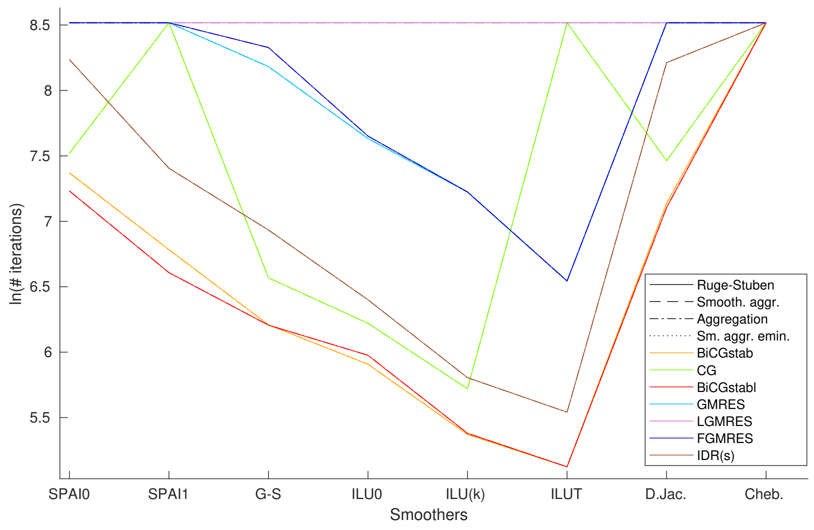


Figure 5.6: Layers 1-35 of SPE10: Number of iterations for all combinations with relaxation as preconditioner.

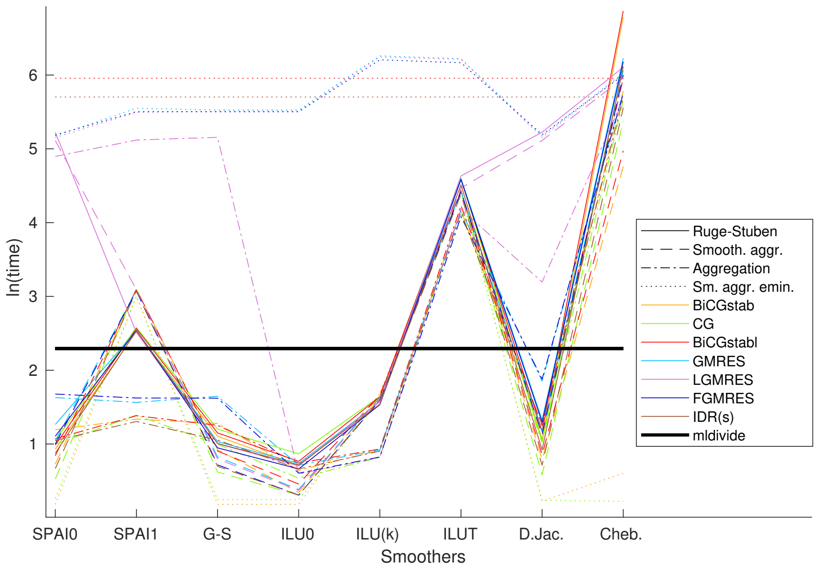


Figure 5.7: Layers 1-35 of SPE10: Run-time for all combinations with AMG as preconditioner.

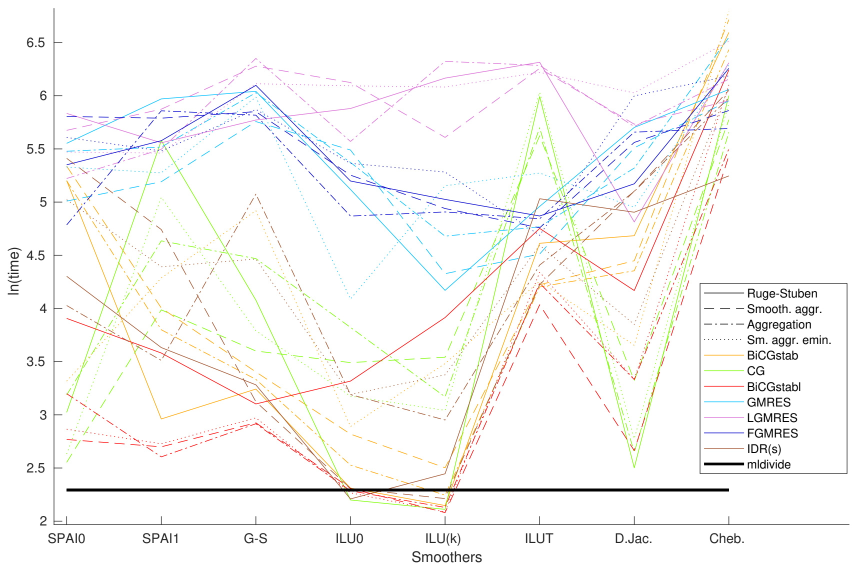


Figure 5.8: Layers 1-35 of SPE10: Run-time for all combinations with relaxation as preconditioner.

thesis, we are not able to present reliable results for any Olympus cases with relaxation as preconditioner. Therefore, the following plots for Case 1 involve AMG preconditioning only.

From Figures 5.9 and 5.10 it is evident that most combinations are faster than `mldivide`. It might appear that the best possible combinations are CG with SPAI0. However, the reason why these combinations finished after only one iteration is because they did not have any chance at converging. From Figure 5.10 one may also be led to believe that CG and BiCGstab with smoothed aggregation emin is the fastest, but looking at the number of iterations in Figure 5.9, these combinations reached the maximum number of iterations, and did not converge.

Overall, the Figures show that ILU(l) and ILUT combined with Ruge-Stüben gives the least number of iterations, while ILU0 combined with smoothed aggregation gives the shortest run-time.

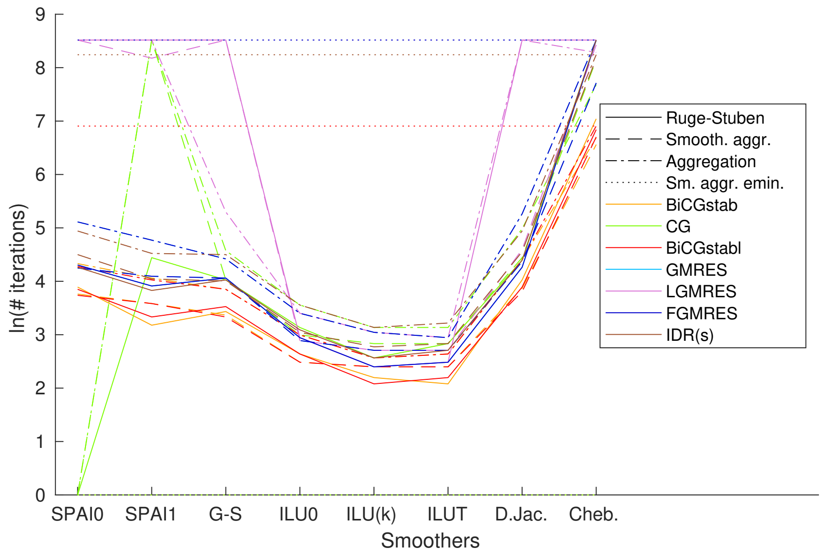


Figure 5.9: Olympus Case 1: Number of iterations for all combinations with AMG as preconditioner.

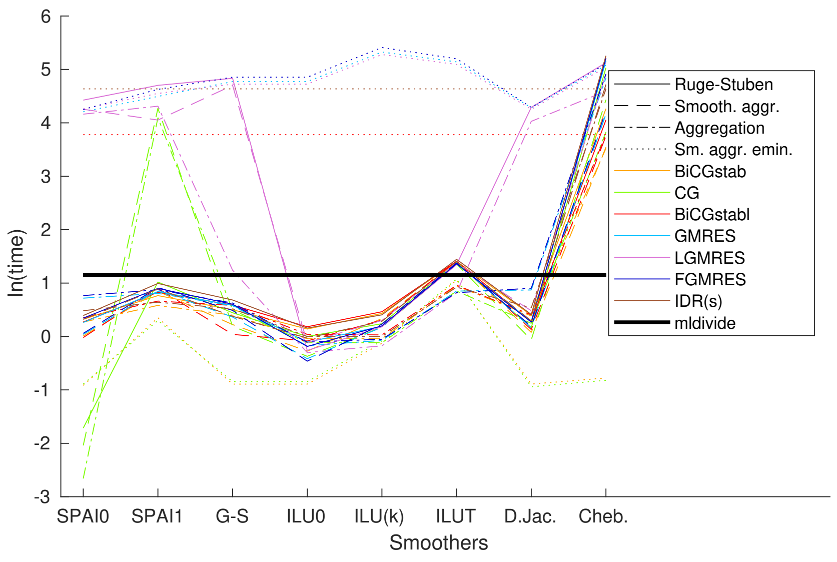


Figure 5.10: Olympus Case 1: Run-time for all combinations with AMG as preconditioner.

Chapter 6

Experiments & Analysis Part 1: Time as a function of the number of grid cells

In this section, the SPE10 model will be the only test model. Since all Olympus realizations have the same number of grid cells, it would be cumbersome to check how time behaves as a function of the number of grid cells here. Because of this, Olympus will not be among the models explored in this section. Furthermore, the simple case and the hybrid case were only intended to be ”warm-up”, and they will not be considered any further.

We will consider three main ”areas” of the SPE10 model: the Tarbert formation, the Upper Ness formation, and the borderline between them (and the layers surrounding it). The borderline case can eventually be extended to include the entire model. It is expected that Tarbert and Upper Ness might differ in which combinations converge faster, considering, among other things, the difference in permeability.

In preparation for the tests we are about to conduct and, later, analyze, we will have to cut down on the number of combinations involved. We can not test all 448 combinations on every test case. Throughout this chapter the preconditioner will be AMG, as it is superior to relaxation methods, which was illustrated earlier. Moreover, for each case being considered, we will base a selection of combinations on the results from said case. Out of all the combinations tested, a selection of the ones that perform best will be picked. Of course, the combinations that performed well in Case 1 might not do so in the other (and larger) cases. Since this section is only concerned with how run-time is affected by the number of grid cells, it is primarily run-time results we are interested in. However, one must not forget about the number of iterations, as they might have great impact on performance as well. A combination might converge in a few second, but at the same time take hundreds of iterations to get there. This must also be considered. Naturally, the same goes for the opposite case.

Low run-time might usually seem more appealing than low iteration count, if one had to choose. However, it is possible that the opposite might prove to be a better choice as the problem size increases – that the combinations that, initially, score high on iteration count and low on run-time eventually will ”catch up” time-wise. To explain, consider some iteration scheme, based on the typical linear system $\mathbf{Ax} = \mathbf{b}$, of the form

$$\mathbf{M}\mathbf{x}^{n+1} = \mathbf{N}\mathbf{x}^{n} + \mathbf{b}.$$

Let Method A be an iterative method that is fast but requires many iterations before convergence, and let Method B be an iterative method that is slow but requires few iterations. The reason why Method B requires much time might be that it has a time-consuming setup phase, which is the phase where the new ”iterative system” (6) is set up. The computation of iterates might, in itself, be as fast as in the case of Method A, or faster even, but because of the slow set-up phase we are left with the impression that the performance of Method B is poor with respect to run-time.

6.1 SPE10 – Tarbert

Starting with the Tarbert formation, I chose five cases on which to evaluate different combinations of Krylov solvers, coarsening strategies and smoothers.

Tarbert cases:

1. Layer 1	(one layer in total: 13 200 grid cells)
2. Layers 1-5	(five layers in total: 66 000 grid cells)
3. Layers 1-10	(ten layers in total: 132 000 grid cells)
4. Layers 1-20	(20 layers in total: 264 000 grid cells)
5. Layers 1-35	(35 layers in total: 462 000 grid cells)

The first selection was taken from the results for Case 1:

Selection 1.

- BiCGstab + Ruge-Stüben + ILUT[1]
- BiCGstab + Ruge-Stüben + ILU(l)
- BiCGstab(l) + smoothed aggregation + ILUT
- BiCGstab(l) + Ruge-Stüben + ILUT

[1] X + Y + Z signifies the combination of Krylov solver X, coarsening Y and smoother Z.

- GMRES + smoothed aggregation + ILUT

I ran Selection 1 on all five cases and plotted the performance of the five combinations, alongside `mldivide`. The number of grid cells is increased by increasing the number of layers, each layer containing $220 \times 60 = 13200$ grid cells. To enhance the differences, the plots are log-log[2] plots. In Figure 6.1 we see that `mldivide` starts out as the

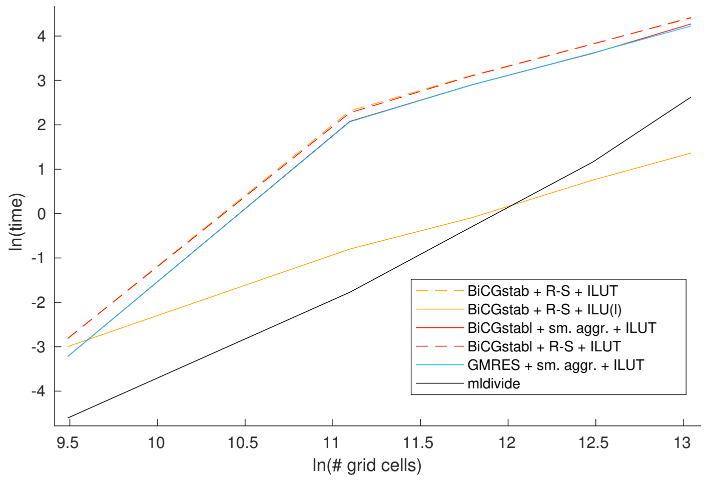


Figure 6.1: Selection 1 tested on all Tarbert cases (log-log plot). The red line is hidden behind the blue line, and the orange dashed line is hidden behind the red dashed line.

fastest-converging solution method. This is as expected, since direct solvers often outperforms iterative solvers on smaller problems. One combination, BiCGstab + Ruge-Stüben + ILUK(k), is performing quite well, competing closely with `mldivide` during the first three cases. Somewhere between Case 3 and Case 4 (for approximately 166 720 grid cells), `mldivide` falls behind said combination, leaving it as the clear winner in the last two cases. We see that the four remaining combinations seem to behave very similarly for each number of grid cells. Between Case 1 and Case 2 they all exhibit a sharp increase in run-time, before slowing down and appearing almost linear. `mldivide` beats them in all cases. Notice how they all share the same smoother, ILUT, and how the only combination to beat `mldivide` did not (as it used ILU(l)).

For comparison, another selection of combinations was chosen for Case 1:

Selection 2.

- BiCGstab + Ruge-Stüben + ILU(l)
- CG + Ruge-Stüben + ILU(l)
- BiCGstabl + Ruge-Stüben + ILUT

[2]In MATLAB, the in-built function `log` uses the natural logarithm (often called ”ln”), unless otherwise stated.

- BiCGstabl + Ruge-Stüben + ILU(l)
- IDR(s) + Ruge-Stüben + ILU(l)

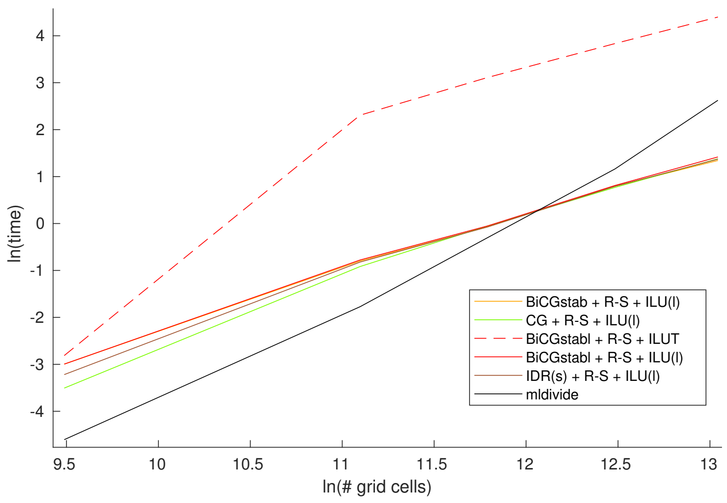


Figure 6.2: Selection 2 tested on all Tarbert cases (log-log plot). The orange line is hidden partly behind the red line (left side) and partly behind the brown line (right side).

In Selection 1, four out of five combinations used ILUT as smoother and performed very poorly, while the last one used ILU(l) as smoother and did very well. Notice that for Selection 2, seen in Figure 6.2, the picture is turned upside down (so to speak): four out of five perform quite well, eventually beating `mldivide` in the last two cases, while the last combination performs badly and is converging much slower than `mldivide` in every case. Now, notice the smoothers: Up until now, every combination that has performed well has used ILU(l), while every combination that has performed poorly has used ILUT. This will be discussed further in the next chapter.

Neither Selection 1 nor Selection 2 is representative for the entire Tarbert formation as they are only based on one layer. Therefore, we need at least one more selection of combinations, this time based on larger cases. To avoid spending too much time scanning all Tarbert results for the top combinations, I decided to base the new selection on the results for Case 5 (all Tarbert layers).

Selection 3.

- BiCGstab + Ruge-Stüben + ILU(0)
- BiCGstab + Ruge-Stüben + ILU(l)
- BiCGstab + smoothed aggregation + ILU(0)

- CG + smoothed aggregation + ILU(0)
- BiCGstab(l) + smoothed aggregation + ILU(0)
- GMRES + smoothed aggregation + ILU(0)
- LGMRES + smoothed aggregation + ILU(0)
- FGMRES + smoothed aggregation + ILU(0)
- IDR(s) + smoothed aggregation + ILU(0)

This time I selected nine combinations, almost twice the previous number. The seven bottom combinations were (almost) equally good and by far the best compared to the remaining combinations, as can be seen in Figure 6.3. Even more interesting is the fact that all these have smoothed aggregation for coarsening and ILU(0) as smoother. A natural assumption would be that smoothed aggregation and ILU(0) together, independent of the choice of Krylov solver, is what we have been looking for. I included all seven combinations because I wanted to see how each solver reacted to different numbers of grid cells, when combined with said coarsening and smoother choices. The only solver that stands out is LGMRES, which for Case 1 takes over 60 times longer to converge than the next worst combination. For variation, I added two more combinations to the selection (the two first on the list above). The best of these two also used ILU(0) as smoother, so for the second I chose a combination with another smoother. Since ILU(l) performed so well in the previous selections, it seemed fitting to include it now.

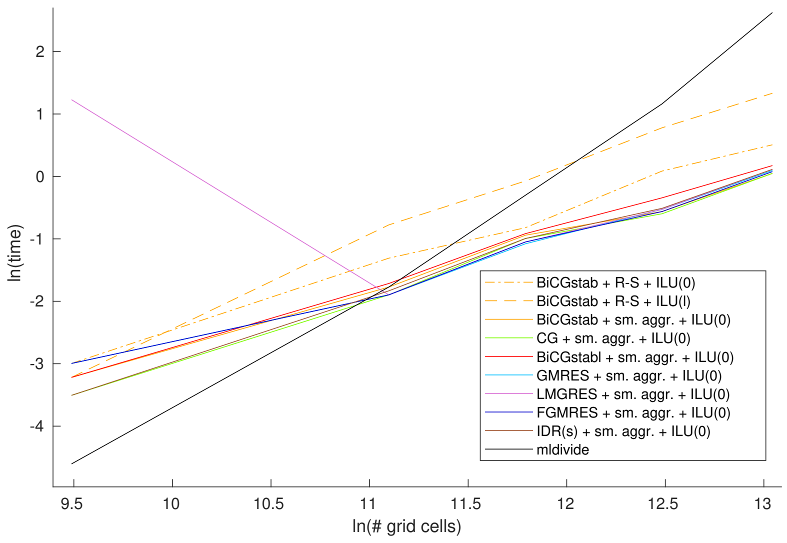


Figure 6.3: Selection 3 tested on all Tarbert cases (log-log plot). On the left side of the plot, the orange line is hidden behind the red line.

Note that Selection 3 and Selection 1 only share one combination: BiCGstab + Ruge-Stüben + ILU(l). As part of Selection 1 this combination was the fastest to converge, but as part of Selection 3 it was actually the slowest in all cases but the first (see Figure 6.3). As before, `mldivide` starts out in lead. Between Case 2 and Case 3, all combinations including ILU(0) catch up with it and takes the lead, and by Case 4 the last combination also passes it. Both selections seem to support that variants of incomplete LU factorization (not threshold-based) work well as smoothers and that variants of the Conjugate Gradient method are good solvers, at least for the Tarbert formation. The standard coarsening, Ruge-Stüben, was the only type of coarsening in Selection 1 (both new and old version), but in Selection 3 aggregation with smoothed iterates took over the spotlight, both with regard to number of combinations and, most importantly, performance.

6.2 SPE10 – Upper Ness

Since the Upper Ness formation spans a total of 50 layers, where the Tarbert formation spans only 35, I decided to increase the number of cases from five to six.

Upper Ness cases:

1. Layer 45	(one layer in total: 13 200 grid cells)
2. Layers 41-45	(five layers in total: 66 000 grid cells)
3. Layers 41-50	(ten layers in total: 132 000 grid cells)
4. Layers 36-55	(20 layers in total: 264 000 grid cells)
5. Layers 41-75	(35 layers in total: 462 000 grid cells)
6. Layers 36-85	(50 layers in total: 660 000 grid cells)

This time I chose Case 3 and Case 6[3] to be the bases of two selections. From Case 3 the following combinations were selected:

Selection 4.

- BiCGstab + smoothed aggregation + SPAI0
- BiCGstab + smoothed aggregation + Gauss-Seidel
- CG + smoothed aggregation + ILU(0)
- BiCGstab(l) + smoothed aggregation + ILU(0)
- BiCGstab(l) + smoothed aggregation + Gauss-Seidel
- BiCGstab(l) + Ruge-Stüben + ILU(l)

[3]These are the Upper Ness cases. Not to be confused with the Tarbert cases or the borderline cases.

- GMRES + smoothed aggregation + ILU(0)
- FGMRES + smoothed aggregation + ILU(0)
- IDR(s) + smoothed aggregation + ILU(0)

My first observation was that smoothed aggregation coarsening together with ILU(0) smoothing are included in five out of nine combinations. Overall, these are also the combinations that has performed best on the Upper Ness formation so far (see Figure 6.4). Ruge-Stüben and smoothed aggregation are still unchallenged as coarsening strategies, even though the latter is now present in eight of the nine combinations. Furthermore, both SPAI0 and Gauss-Seidel have entered the race as potentially good smoothers. However, we see in Figure 6.4 that the three combinations containing these smoothers, as well as the one combination with Ruge-Stüben coarsening, take longer to converge than the remaining combinations in four of five cases. Between Case 2 and Case 3, `mldivide` goes from being the fastest solver to being the slowest. This comes as less of a surprise considering that the selection is based on Case 3.

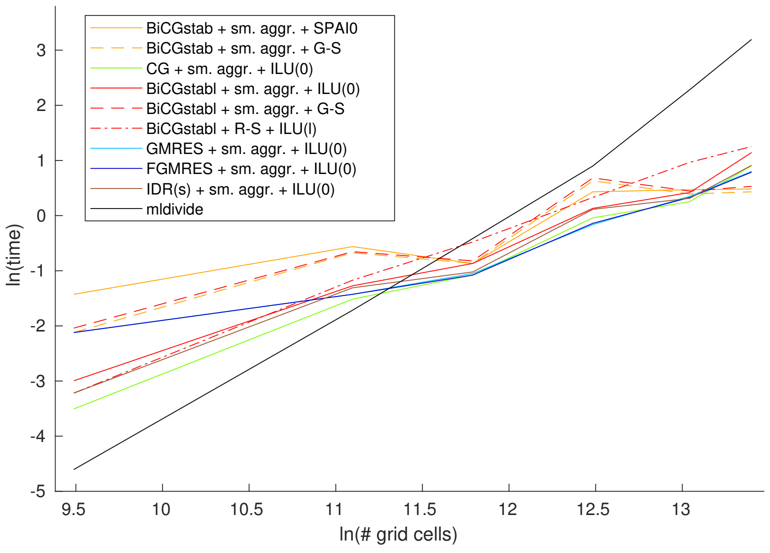


Figure 6.4: Selection 4 tested on all Upper Ness cases (log-log plot).

The second selection from Upper Ness was taken from Case 6 and contains ten combinations:

Selection 5.

- BiCGstab + smoothed aggregation + SPAI0
- BiCGstab(l) + smoothed aggregation + damped Jacobi
- BiCGstab(l) + smoothed aggregation + Gauss-Seidel
- BiCGstab(l) + Ruge-Stüben + ILU(0)
- GMRES + smoothed aggregation + damped Jacobi
- LGMRES + smoothed aggregation + ILU(0)
- FGMRES + smoothed aggregation + SPAI0
- IDR(s) + smoothed aggregation + damped Jacobi
- IDR(s) + smoothed aggregation + Gauss-Seidel
- IDR(s) + smoothed aggregation + SPAI0

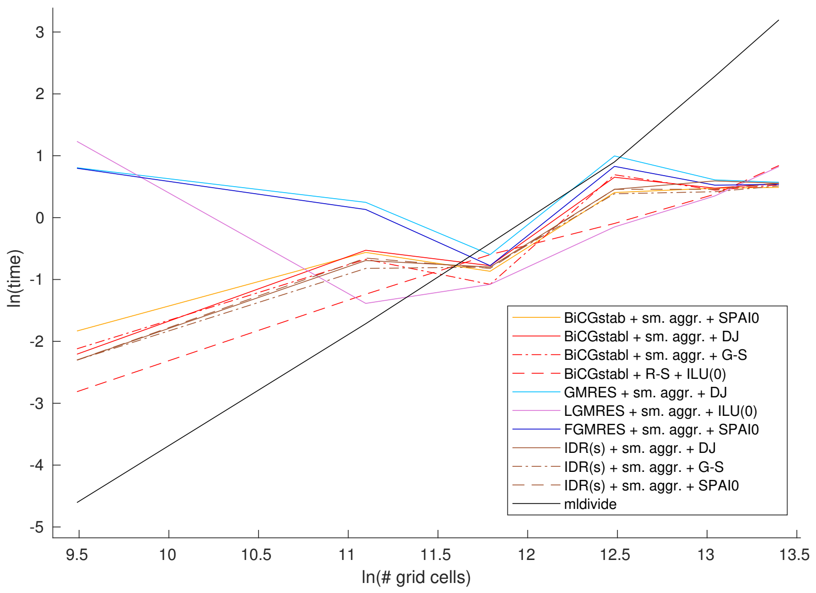


Figure 6.5: Selection 5 tested on all Upper Ness cases (log-log plot).

This time most of the combinations (eight) have SPAI0 and Gauss-Seidel, along with damped Jacobi, as smoothers. The two remaining combinations have ILU(0). This is a sudden change in direction. But, as we see in Figure 6.5, the latter two are still among the best in several cases. GMRES + smoothed aggregation + damped Jacobi and FGMRES + smoothed aggregation, however, are the slowest-converging combinations.

6.3 SPE10 – borderline area

Now we want to look at how different combinations behave when both Tarbert and Upper Ness layers are in play. The cases we will consider therefore include layers from both formations, and we will not limit this to only layers in close proximity of the border. These are the chosen cases:

Borderline cases:

1. Layers 35-36 (two layers in total: 26 400 grid cells)
2. Layers 32-38 (seven layers in total: 92 400 grid cells)
3. Layers 30-45 (16 layers in total: 211 200 grid cells)
4. Layers 21-50 (30 layers in total: 396 000 grid cells)
5. Layers 1-85 (85 layers in total: 1 122 000 grid cells)

We will consider one selection of combinations based on the results of Case 2 and one based on Case 5. The latter covers the entire SPE10 model and might be a bit heavy to run tests on.

Case 2 was chosen because it is ”close” to the boundary between Tarbert and Upper Ness, but not the smallest one. It is interesting to see how all combinations tackle this case, and whether this selection will differ much from the one that follows.

Selection 6.

- BiCGstab + smoothed aggregation + ILU(0)
- BiCGstab + smoothed aggregation + ILU(l)
- BiCGstab + aggregation + ILU(l)
- BiCGstab + Ruge-Stüben + ILU(l)
- CG + smoothed aggregation + ILU(0)
- CG + aggregation + ILU(l)
- LGMRES + aggregation + ILU(l)
- LGMRES + Ruge-Stüben + ILU(k)
- FGMRES + smoothed aggregation + ILU(l)

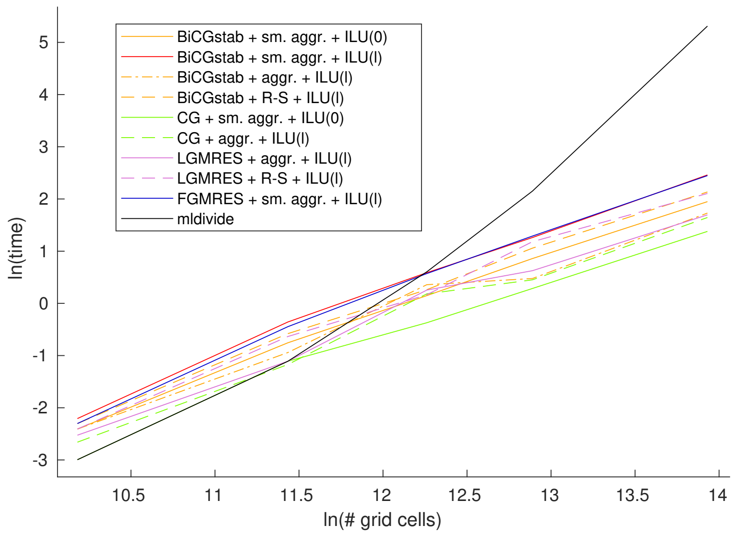


Figure 6.6: Selection 6 tested on all borderline cases (log-log plot).

Figure 6.6 shows the resulting plot over time versus number of grid cells: The next selection was, as mentioned, taken from Case 5

Selection 7.

- BiCGstab + smoothed aggregation + SPAI0
- BiCGstab + smoothed aggregation + Gauss-Seidel
- BiCGstab + smoothed aggregation + damped Jacobi
- CG + smoothed aggregation + damped Jacobi
- CG + smoothed aggregation + ILU(0)
- BiCGstab(l) + smoothed aggregation + SPAI0
- BiCGstab(l) + smoothed aggregation + Gauss-Seidel
- BiCGstab(l) + smoothed aggregation + damped Jacobi
- GMRES + smoothed aggregation + damped Jacobi
- IDR(s) + smoothed aggregation + Gauss-Seidel

The results can be seen in Figure 6.7

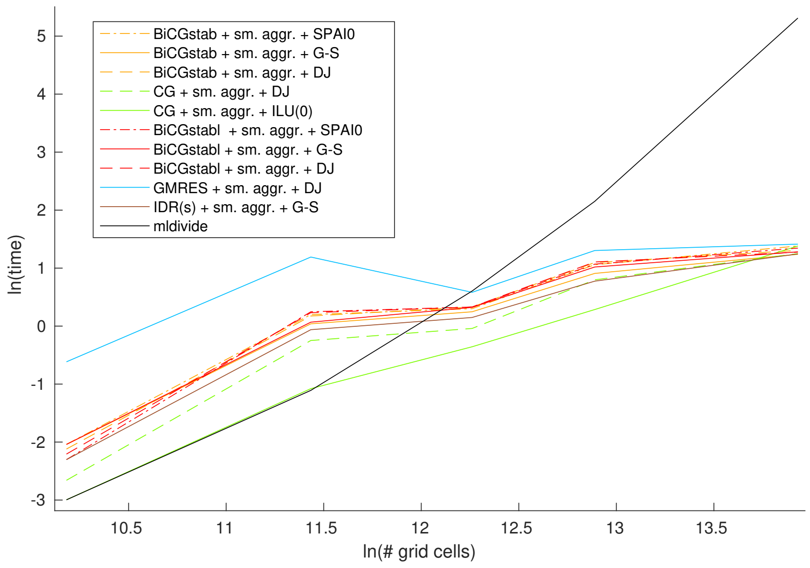


Figure 6.7: Selection 7 tested on all borderline cases (log-log plot).

Chapter 7

Experiments & Analysis Part 2: Mean value and standard deviation

For a population of real numbers $\{x^1, ..., x^m\}$, the *mean value* is given as

$$\bar{x} = \frac{1}{m}\sum_{i=1}^{m} x^i,$$

and the *standard deviation* as

$$s = \sqrt{\frac{1}{m}\sum_{i=1}^{m}(x^i - \bar{x})^2}.$$

where x^i denotes the i'th number, <u>not</u> the number x to the power of i.

For a population of vectors $\{\mathbf{x}^i\}_{i=1}^m \in \mathbb{R}^N$, we will simply calculate mean value and standard deviation for each set of corresponding vector components. That is, for each case we get a mean value vector $\bar{\mathbf{x}}$ and a standard deviation vector $\mathbf{s}$ with components

$$\bar{x}_j = \frac{1}{m}\sum_{i=1}^{m} x_j^i \tag{7.1}$$

$$s_j = \sqrt{\frac{1}{m}\sum_{i=1}^{m}(x_j^i - \bar{x}_j)^2} \tag{7.2}$$

for $j = 1, ..., N$. For every case, we will select a number of the combinations that performed best and plot these using the MATLAB function `errorbar`. When using `errorbar` it is important that all the combinations being plotted have mean values and standard deviations of similar magnitude, to avoid a situation where some bad results dominate the whole plot. Additionally, including too many combinations would result in a chaotic plot.

7.1 Olympus

Each of the ten cases was run three times and a selection of well-performing combinations was picked, the number of which varies from case to case. The selections will typically include some combinations that perform better with respect to run-time than with respect to number of iterations, and some in which the opposite case applies. All calculations of mean values and corresponding standard deviation are based on these selections. Following are the selections for Cases 1-10, as well as plots where mean values are represented as circular points and standard deviations are represented by the error bars associated with each point. Overall, we are limiting our ”explorations” to the Olympus result in which AMG was used as preconditioner, the reason being the superiority of preconditioning with AMG as opposed to with relaxation.

7.1.1 Case 1

Selection 8.

1. BiCGstab + Ruge-Stüben + ILU(0)
2. BiCGstab + Ruge-Stüben + ILU(l)
3. BiCGstab + smoothed aggregation + ILU(0)
4. CG + smoothed aggregation + ILU(0)
5. BiCGstabl + Ruge-Stüben + ILU(0)
6. BiCGstabl + Ruge-Stüben + ILU(l)
7. BiCGstabl + smoothed aggregation + ILU(0)
8. GMRES + Ruge-Stüben + ILU(0)
9. GMRES + Ruge-Stüben + ILU(l)
10. GMRES + smoothed aggregation + ILU(0)
11. LGMRES + Ruge-Stüben + ILU(0)
12. LGMRES + Ruge-Stüben + ILU(l)
13. FGMRES + Ruge-Stüben + ILU(0))
14. FGMRES + Ruge-Stüben + ILU(l)
15. FGMRES + smoothed aggregation + ILU(0)

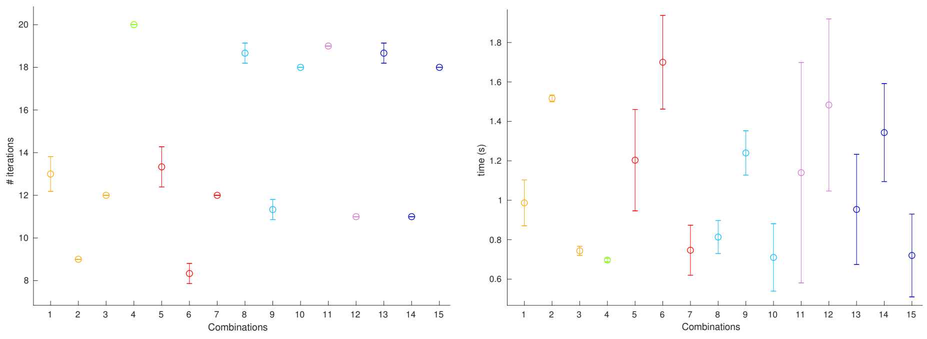


Figure 7.1: Olympus Case 1: Mean values with corresponding standard deviations for number of iterations (top) and time (bottom).

Observations

Looking at Figure 7.1, note that Combinations 2 and 6 reach convergence in fewest iterations, but that they also use the most time. About half of the combinations from Selection 8 reach convergence in approximately the same short time, but it is difficult to determine which are the fastest. Overall, Combinations 3 and 7 perform best when both number of iterations and run-time are taken into account, both having smoothed aggregation coarsening and ILU(0) as smoother.

7.1.2 Case 2

Selection 9.

1. BiCGstab + Ruge-Stüben + ILU(0)
2. BiCGstab + smoothed aggregation + ILU(0)
3. CG + Ruge-Stüben + ILU(0)
4. CG + smoothed aggregation + ILU(0)
5. BiCGstabl + Ruge-Stüben + ILU(0)
6. BiCGstabl + smoothed aggregation + ILU(0)
7. GMRES + Ruge-Stüben + ILU(0)
8. GMRES + smoothed aggregation + ILU(0)
9. LGMRES + Ruge-Stüben + ILU(0)
10. FGMRES + Ruge-Stüben + ILU(0)
11. FGMRES + smoothed aggregation + ILU(0)
12. IDR(s) + Ruge-Stüben + ILU(0)

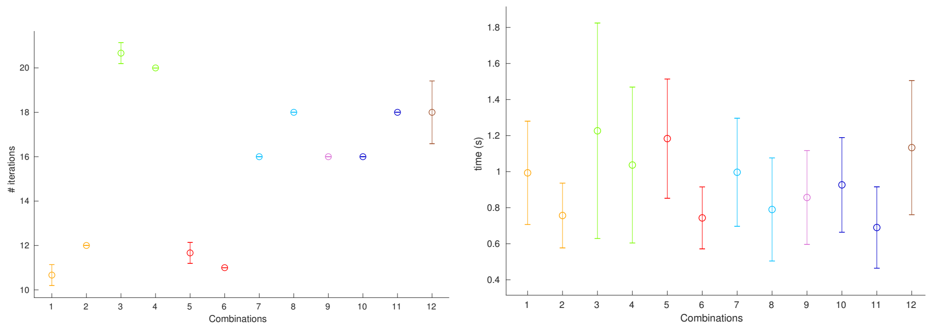


Figure 7.2: Olympus Case 2: Mean values with corresponding standard deviations for number of iterations (left) and time (right).

Observations

Judging by Figure 7.2, Combinations 1, 2, 5 and 6 (that is, all combinations in which the Krylov solver is BiCGstab or BiCGstabl) are by far the best when it comes to number of iterations. Moreover, Combinations 2 and 6 are among the fastest solvers too, together with Combinations 8 and 11. Overall, Combinations 2 and 6 are performing best with respect to both iteration count and run-time. Notice that both combinations include smoothed aggregation as coarsening strategy and ILU(0) as smoother.

7.1.3 Case 3

Selection 10.

1. BiCGstab + smoothed aggregation + ILU(0)
2. BiCGstabl + smoothed aggregation + ILU(0)
3. BiCGstabl + aggregation + ILU(0)
4. GMRES + Ruge-Stüben + ILU(0)
5. GMRES + smoothed aggregation + ILU(0)
6. LGMRES + smoothed aggregation + ILU(0)
7. FGMRES + Ruge-Stüben + ILU(0)
8. FGMRES + smoothed aggregation + ILU(0)
9. FMGRES + aggregation + ILU(0)
10. FGMRES + aggregation + ILU(l)

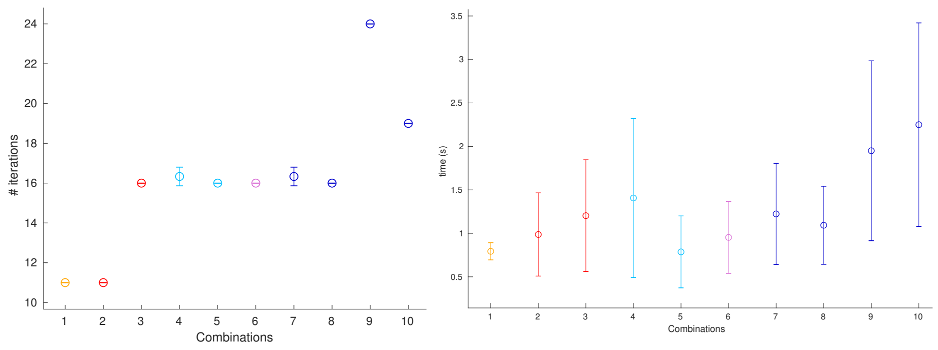


Figure 7.3: Olympus Case 3: Mean values with corresponding standard deviations for number of iterations (left) and time (right).

Observations

Combinations 1 and 2 are clearly converging in the least number of iterations (see Figure 7.3. Additionally, they are among the fastest combinations, from which we claim that they are the two overall best solvers. Again, the best combinations consist of coarsening strategy smoothed aggregation and smoother ILU(0).

7.1.4 Case 4

Selection 11.

1. BiCGstab + Ruge-Stüben + ILU(0)
2. BiCGstab + smoothed aggregation + ILU(0)
3. BiCGstab + aggregation + ILU(0)
4. BiCGstab + aggregation + ILU(l)
5. CG + Ruge-Stüben + ILU(0)
6. CG + smoothed aggregation + ILU(0)
7. BiCGstabl + smoothed aggregation + ILU(0)
8. GMRES + Ruge-Stüben + ILU(0)
9. GMRES + smoothed aggregation + ILU(0)
10. LGMRES + Ruge-Stüben + ILU(0)
11. FGMRES + smoothed aggregation + ILU(0)
12. IDR(s) + smoothed aggregation + ILU(0)

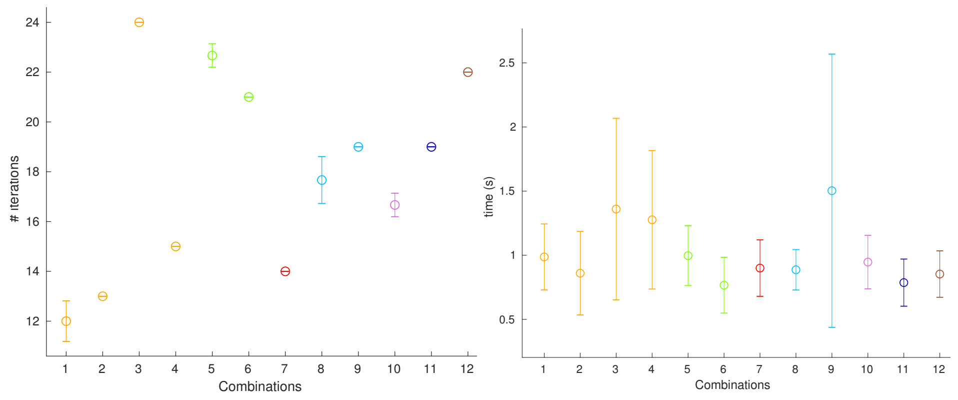


Figure 7.4: Olympus Case 4: Mean values with corresponding standard deviations for number of iterations (left) and time (right).

Observations

Observe, in the iteration plot to the left of Figure 7.4, that Combinations 1, 2 and 7 have the lowest iteration counts. These combinations are also performing well with respect to run-time, being among the fastest combinations to converge, and two of them are smoothed by ILU(0) and have smoothed aggregation coarsening.

7.1.5 Case 5

Selection 12.

1. BiCGstab + smoothed aggregation + ILU(0)
2. GMRES + Ruge-Stüben + ILU(0)
3. GMRES + Ruge-Stüben + ILU(l)
4. GMRES + smoothed aggregation + ILU(0)
5. LGMRES + smoothed aggregation + ILU(0)
6. LGMRES + aggregation + ILU(0)
7. LGMRES + aggregation + ILU(l)
8. FGMRES + smoothed aggregation + ILU(0)
9. FGMRES + aggregation + ILU(0)
10. FGMRES + aggregation + ILU(l)
11. IDR(s) + Ruge-Stüben + ILU(0)

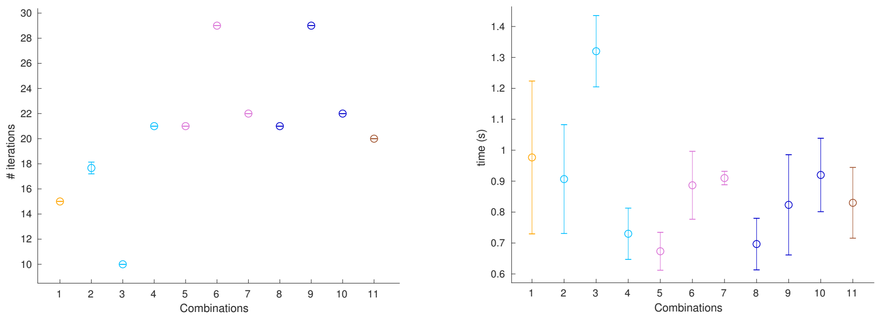


Figure 7.5: Olympus Case 5: Mean values with corresponding standard deviations for number of iterations (left) and time (right).

Observations

For this Olympus case, no. 5, BiCGstabl was not included as Krylov solver in any of the selected combinations. The results for Selection 12, seen in Figure 7.5, were generally scattered, with no combinations being especially good at both run-time and number of iterations. We observe that Combination 3 is the best when it comes to iteration count but worst when it comes to run-time. It it important to keep in mind that we are looking at selections of the combinations that performed best on the different cases and that, when speaking of bad performance, this is all relative to the performance of the other combinations.

7.1.6 Case 6

Selection 13.

1. BiCGstab + smoothed aggregation + ILU(0)
2. BiCGstab + aggregation + ILU(0)
3. BiCGstab + aggregation + ILU(l)
4. CG + smoothed aggregation + ILU(0)
5. BiCGstabl + aggregation + ILU(0)
6. BiCGstabl + aggregation + ILU(l)
7. LGMRES + Ruge-Stüben + ILU(0)
8. LGMRES + Ruge-Stüben + ILU(l)
9. FGMRES + Ruge-Stüben + ILU(0)
10. IDR(s) + smoothed aggregation + ILU(0)

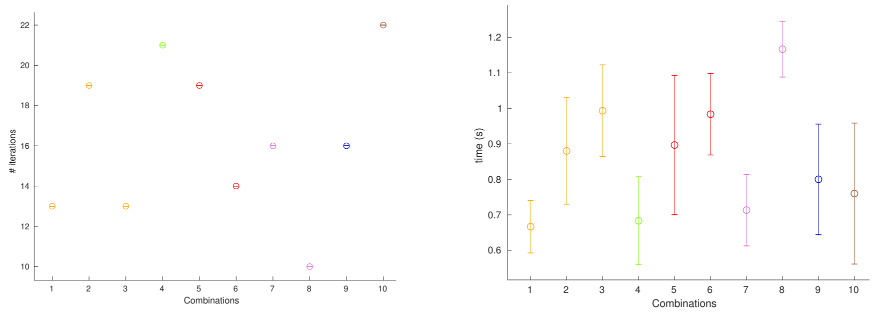


Figure 7.6: Olympus Case 6: Mean values with corresponding standard deviations for number of iterations (left) and time (right).

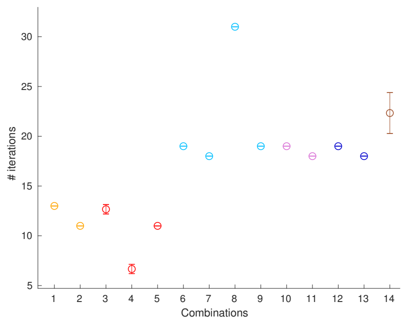
Observations

From Figure 7.6 we observe that Combinations 3, 4, 8 and 10 clearly perform well for either iteration count or run-time, and poorly for the other. This is particularly visible in Combination 8, which uses fewest iterations but the most time to converge, and Combinations 4 and 10, which are among the fastest to converge but require the most iterations to do so. However, Combination 1 is both fast and requires few iterations.

7.1.7 Case 7

Selection 14.

1. BiCGstab + Ruge-Stüben + ILU(0)
2. BiCGstab + smoothed aggregation + ILU(0)
3. BiCGstabl + Ruge-Stüben + ILU(0)
4. BiCGstabl + Ruge-Stüben + ILU(l)
5. BiCGstabl + smoothed aggregation + ILU(0)
6. GMRES + Ruge-Stüben + ILU(0)
7. GMRES + smoothed aggregation + ILU(0)
8. GMRES + aggregation + ILU(0)
9. GMRES + aggregation + ILU(l)
10. LGMRES + Ruge-Stüben + ILU(0)
11. LGMRES + smoothed aggregation + ILU(0)
12. FGMRES + Ruge-Stüben + ILU(0)
13. FGMRES + smoothed aggregation + ILU(0)
14. IDR(s) + Ruge-Stüben + ILU(0)



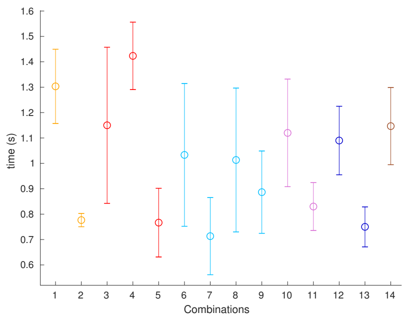


Figure 7.7: Olympus Case 7: Mean values with corresponding standard deviations for number of iterations (left) and time (right).

Observations

All combinations with either BiCGstab or BiCGstabl as Krylov subspace solver, i.e., Combinations 1-5, are at the bottom of the iteration plot i Figure 7.7, requiring few iterations. Out of these, Combination 4 is clearly the best. This combination is also among the very best when it comes to run-time too. Combination 4 is BiCGstabl + Ruge-Stüben + ILU(0). Combinations 2 and 5 are also both fast and require few iterations, and both of these consist of smoothed aggregation coarsening and ILU(0) smoothing. Combinations 7, 11 and 13 are also converging fast. Another thing to notice is that the majority of the combinations using variants for GMRES as Krylov solver has similar iteration count. However, this might be simply a coincidence.

7.1.8 Case 8

Selection 15.

1. BiCGstab + smoothed aggregation + ILU(0)
2. CG + smoothed aggregation + ILU(0)
3. BiCGstabl + smoothed aggregation + ILU(0)
4. GMRES + Ruge-Stüben + ILU(0)
5. GMRES + Ruge-Stüben + ILU(l)
6. GMRES + smoothed aggregation + ILU(0)
7. LGMRES + Ruge-Stüben + ILU(0)
8. LGMRES + smoothed aggregation + ILU(0)
9. FGMRES + Ruge-Stüben + ILU(0)
10. FGMRES + smoothed aggregation + ILU(0)
11. IDR(s) + smoothed aggregation + ILU(0)

Observations

As is evident from Figure 7.8, Combinations 1, 3 and 5 have the lowest iteration count, the latter combination being the best. Combinations 1 and 3 are also among the fastest combinations, along with Combinations 2, 8 and 10. We can conclude that Combinations 1 and 3 are performing very well with respect to both iteration count and run-time, which they have proved to be in several of the earlier cases too. Furthermore, observe that Combinations 2 and 11 converge fast but require the most iterations of all. The opposite goes for Combination 5.

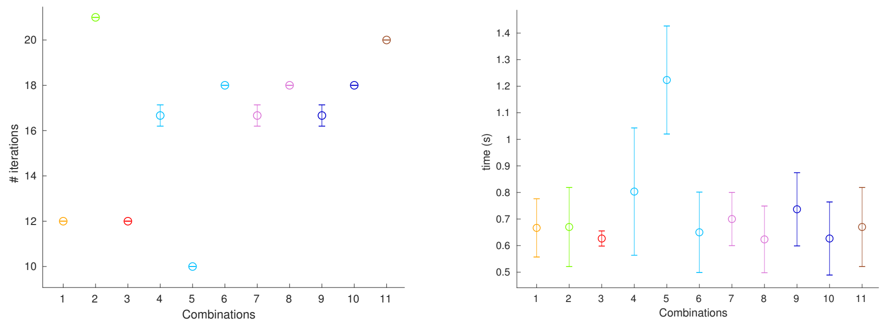


Figure 7.8: Olympus case 8: Mean values with corresponding standard deviations for number of iterations (left) and time (right).

7.1.9 Case 9

Selection 16.

1. BiCGstab + Ruge-Stüben + ILU(0)
2. BiCGstab + Ruge-Stüben + ILU(l)
3. BiCGstab + smoothed aggregation + ILU(0)
4. BiCGstabl + Ruge-Stüben + ILU(0)
5. BiCGstabl + Ruge-Stüben + ILU(l)
6. BiCGstabl + smoothed aggregation + ILU(0)
7. GMRES + Ruge-Stüben + ILU(0)
8. GMRES + smoothed aggregation + ILU(0)
9. LGMRES + Ruge-Stüben + ILU(0)
10. FGMRES + Ruge-Stüben + ILU(0)
11. FGMRES + smoothed aggregation + ILU(0)
12. FGMRES + smoothed aggregation + ILU(l)
13. FGMRES + aggregation + ILU(0)
14. FGMRES + aggregation + ILU(l)

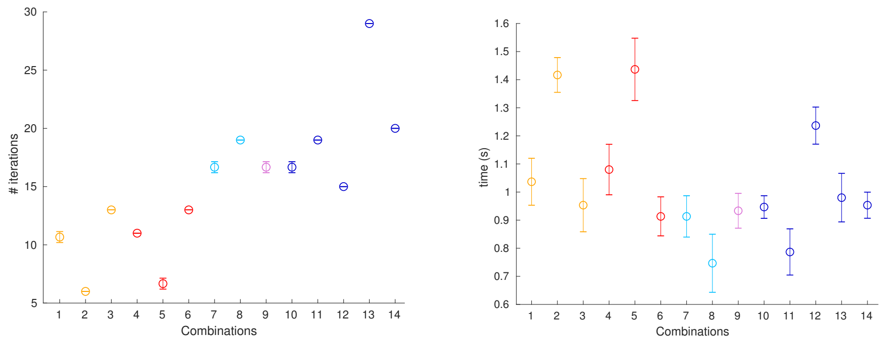


Figure 7.9: Olympus case 9: Mean values with corresponding standard deviations for number of iterations (left) and time (right).

Observations

Studying Figure 7.9 we see that Combinations 1-6, all with variants of BiCGstab as Krylov solver, require few iterations before converging. Out of these, Combinations 2 and 5 require fewest iterations, but they are also the combinations with the longest run-times. The combinations converging fastest are Combinations 8 and 11. Combinations 3 and 6 appear to be good at both run-time and number of iterations.

7.1.10 Case 10

Selection 17.

1. BiCGstab + Ruge-Stüben + ILU(0)
2. BiCGstab + Ruge-Stüben + ILU(l)
3. BiCGstab + smoothed aggregation + ILU(0)
4. BiCGstab + aggregation + ILU(0)
5. BiCGstab + aggregation + ILU(l)
6. CG + smoothed aggregation + ILU(0)
7. BiCGstabl + Ruge-Stüben + ILU(0)
8. BiCGstabl + Ruge-Stüben + ILU(l)
9. BiCGstabl + smoothed aggregation + ILU(0)
10. BiCGstabl + aggregation + ILU(0)
11. BiCGstabl + aggregation + ILU(l)
12. GMRES + Ruge-Stüben + ILU(0)

13. GMRES + smoothed aggregation + ILU(0)
14. LGMRES + Ruge-Stüben + ILU(0)
15. LGMRES + smoothed aggregation + ILU(0)
16. FGMRES + Ruge-Stüben + ILU(0)
17. FGMRES + smoothed aggregation + ILU(0)

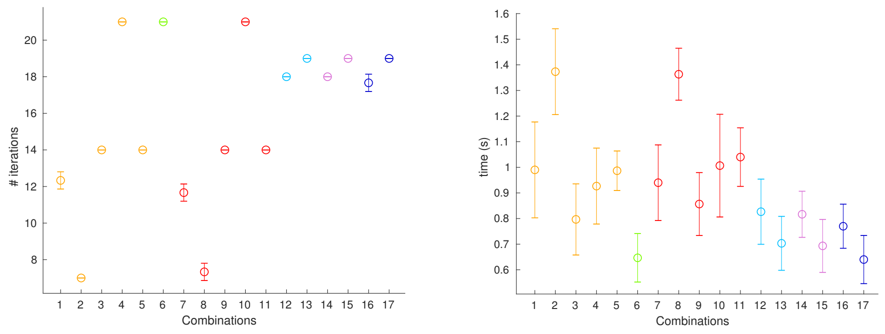


Figure 7.10: Olympus case 10: Mean values with corresponding standard deviations for number of iterations (left) and time (right).

Observations

By far, Combinations 2 and 8 are the combinations performing best with respect to number of iterations, but they also have the longest run-times. The fastest-converging combinations are Combinations 6, 13, 15 and 17, all having smoothed aggregation coarsening and ILU(0) as smoother.

7.1.11 Overall observations

One might expect the number of iterations to stay fixed for each combination between different runs. The iteration counts for the ten Olympus cases do not vary much between runs, but the fact is that they do vary. This variation might be a result of random factors affecting the parallelization of the code and, thus, the iteration count and run-time – for instance, variations in the number of available threads or in how much ”work” each thread is able to process at a time.

It seems to be some kind of inverse correlation between time and iteration count, at least when it comes to the ”extreme” values. As is evident from the figures (i.e., Figure 7.10), combinations that require particularly few iterations typically need much more time to converge than other combinations, and combinations with iteration count above the average are typically among the fastest. Clear examples are combinations that use ILUT as smoother, typically having the lowest iteration count but at the same time being among the slowest-converging combinations (see Section 6). This is also often the case for the Olympus model but, because of its bad performance with respect to run-time, none of the selected combinations have included ILUT.

Another interesting trend is that each Krylov solver generally seems to achieve convergence fastest (i.e., in shortest run-time) when combined with smoothed aggregation coarsening and ILU(0) as smoother. Likewise, Krylov solvers combined with Ruge-Stüben and ILU(l) are typically the combinations that require fewest iterations in order to converge.

Moreover, it appears that variants of GMRES often have the same, or very similar, mean values and standard deviations with respect to iteration count when combined with the same coarsening strategy and smoother. This pattern can be recognized in all of the figures. Consider, for instance, Figure 7.3. The mean values and standard deviations for Combination 4 and 7 look very similar. These two combinations correspond to GMRES + Ruge-Stüben + ILU(0) and FGMRES + Ruge-Stüben + ILU(0), respectively – both the same type of coarsening and the same smoother. Now, consider Combinations 5, 6 and 8, which also appear to share the same mean value and standard deviation for number of iterations. Checking Selection 10 we see that all three combinations have the same coarsening and the same smoother, namely smoothed aggregation and ILU(0).

7.2 SPE10

For the calculation of mean values and standard deviations I had originally decided to look at the following three areas of the SPE10 model: layers 1-35 (Tarbert), layers 41-75 (Upper Ness) and layers 21-50 (borderline). For each of these areas, a selection of well-performing combinations would be picked based on three runs. I had already run the three areas once each and obtained results where nothing seemed out of the ordinary. However, when I later ran the code two more times for layers 21-50 and layers 41-75, something went wrong and none of the combinations converged. It appears that the code must have crashed for almost every single combination, since the majority of them never reached the result file, not even with a flag signifying divergence. Those that did not crash, for which the results were written to the result file, did not converge. Despite having only one reliable run of both layers 21-50 and layers 41-75, the results for both cases will also be presented, however, only for AMG as preconditioner and without proper mean values or standard deviations. Before this, we will begin by calculating mean values and standard deviations for the one case that was not affected by any unforeseen technical issues, namely layers 1-35. Here, we will consider both the case where AMG is the preconditioner and the case where relaxation is the preconditioner, and pick one selection for each case. Observations will be made on behaviour and trends, along with attempts to analyze and discuss the findings.

7.2.1 Tarbert layers 1-35 with relaxation as preconditioner

As we have seen before, with relaxation as preconditioner (and not AMG) coarsening strategies are irrelevant and will all yield the same results. Thus, in the selection of combinations below we have simply decided on using the standard Ruge-Stüben coarsening

Selection 18.

1. BiCGstab + Ruge-Stüben + ILU(0)
2. BiCGstab + Ruge-Stüben + ILU(l)
3. CG + Ruge-Stüben + ILU(0)
4. CG + Ruge-Stüben + ILU(l)
5. BiCGstabl + Ruge-Stüben + ILU(0)
6. BiCGstabl + Ruge-Stüben + ILU(l)
7. IDR(s) + Ruge-Stüben + ILU(0)
8. IDR(s) + Ruge-Stüben + ILU(l)

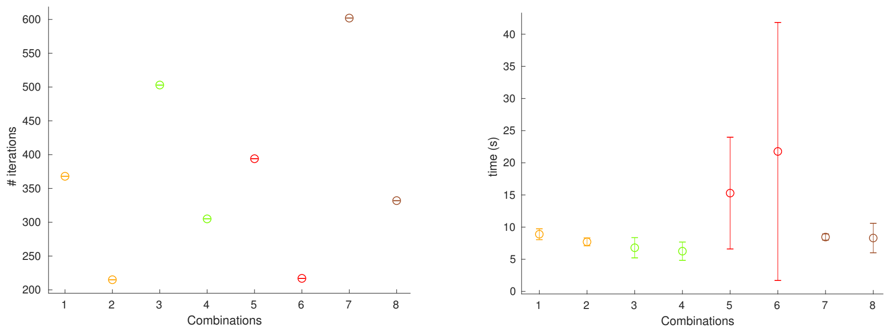


Figure 7.11: Layers 1-35 of SPE10, relaxation as preconditioner: Mean values with corresponding standard deviations for number of iterations (left) and run-time (right).

Observations

As we see in Figure 7.11, Combinations 2 and 6 are the combinations that converge after the lowest number of iterations. The majority of combinations are converging relatively fast. Notice that Combinations 1, 2, 4 and 8 perform well with respect to both iteration count and run-time, out of which Combinations 2 and 4 are the best. Furthermore, the standard deviations of Combinations 5 and 6 with respect to time are very large compared to the other combinations. Inspecting the results from the three runs used to compute mean values and standard deviations, we can see that Combination 5 has converged in 27.58, 9.01 and 9.27 seconds, and Combination 6 has converged in 50.13, 7.48 and 7.69 seconds. The longest run-time for both combinations belongs to the first run. Also, Combination 6 (BiCGstabl + Ruge-Stüben + ILU(0)) is next in line after the code has completed the computations for Combinations 5 (BiCGstabl + Ruge-Stüben + ILU(l)). As mentioned, all computations are executed on the Markov cluster and are therefore prone to delay due to lack of available threads. Since Combinations 5 and 6 are consecutively executed, it is likely that they both were affected by the same delay during the first run. A more reliable approach to this issue would be to run the codes locally on a computer, rather than using the Markov cluster.

7.2.2 Tarbert layers 1-35 with AMG as preconditioner

We use the same selection as in Section 6.1, namely Selection 3:

1. BiCGstab + Ruge-Stüben + ILU(0)
2. BiCGstab + Ruge-Stüben + ILU(l)
3. BiCGstab + smoothed aggregation + ILU(0)
4. CG + smoothed aggregation + ILU(0)
5. BiCGstabl + smoothed aggregation + ILU(0)
6. GMRES + smoothed aggregation + ILU(0)
7. LGMRES + smoothed aggregation + ILU(0)
8. FGMRES + smoothed aggregation + ILU(0)
9. IDR(s) + smoothed aggregation + ILU(0)

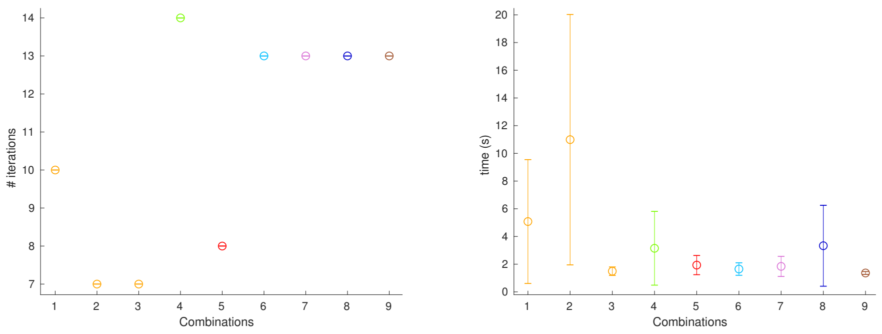


Figure 7.12: Layers 1-35 of SPE10, AMG as preconditioner: Mean values with corresponding standard deviations for number of iterations (left) and run-time (right).

Observations

First of all, notice that the majority of combinations appears to converge fast but only after relatively many iterations. Combinations 2, 3 and 5 converge in fewest iterations, while Combinations 3, 4, 5, 7 and 9 converge in the shortest amount of time. Notice how Combination 3 has the best performance with both regards. Another combination that, overall, performs well is Combination 5. Both Combination 3 and 5 have the same coarsening and the same smoother: smoothed aggregation and ILU(0).

7.2.3 Upper Ness layers 41-75 with AMG as preconditioner

Selection 19.

1. CG + smoothed aggregation + ILU(0)
2. CG + smoothed aggregation + damped Jacobi
3. BiCGstabl + Ruge-Stüben + ILU(0)
4. BiCGstabl + smoothed aggregation + ILU(l)
5. BiCGstabl + smoothed aggregation + damped Jacobi
6. GMRES + smoothed aggregation + damped Jacobi
7. FGMRES + Ruge-Stüben + ILU(0)
8. FGMRES + smoothed aggregation + ILU(0)
9. IDR(s) + smoothed aggregation + ILU(0)

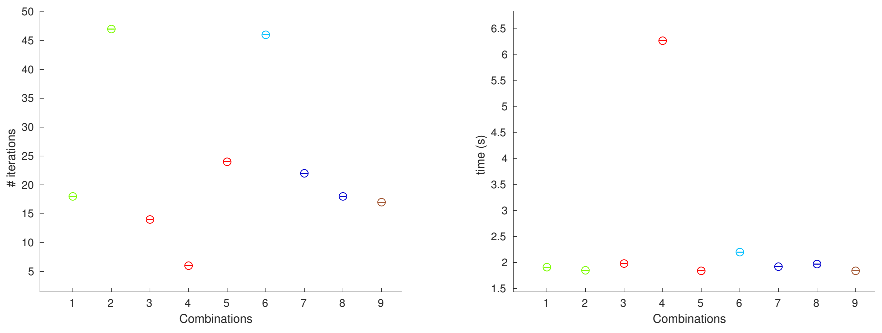


Figure 7.13: Layers 41-75 of SPE10, AMG as preconditioner: Number of iterations (left) and run-time (right).

Observations

Inspecting Figure 7.13 we see that all combination, except for Combination 4, are close to equally fast. Combination 4 requires much more time to converge, but it is also the combination that needs fewest iterations. It might be coincidental, but notice that none of the combinations in Selection 19 uses BiCGstab as Krylov solver, whereas three out of nine combinations do so in Selection 3 (layers 1-35).

7.2.4 Borderline layers 21-50 with AMG as preconditioner

Selection 20.

1. BiCGstabl + Ruge-Stüben + ILU(0)
2. BiCGstabl + smoothed aggregation + ILU(0)
3. BiCGstabl + aggregation + ILU(l)
4. GMRES + smoothed aggregation + ILU(0)
5. LGMRES + smoothed aggregation + ILU(0)
6. LGMRES + aggregation + ILU(l)
7. FGMRES + Ruge-Stüben + ILU(0)
8. FGMRES + aggregation + ILU(l)
9. IDR(s) + smoothed aggregation + damped Jacobi
10. IDR(s) + aggregation + ILU(0)

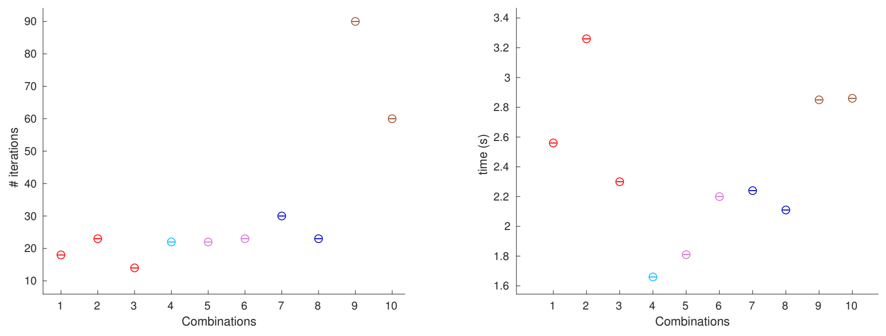


Figure 7.14: Layers 21-50 of SPE10, AMG as preconditioner: Number of iterations (left) and run-time (right).

Observations

As is evident in Figure 7.14, the two combinations having IDR(s) as Krylov solver are requiring several times as many iterations to converge as the other combinations. Combination 3 is the combination that requires the fewest iterations. Combinations 4 and 5 are the fastest and have relatively low iteration counts as well.

7.2.5 Overall observations

As for the Olympus cases in the previous section, there seems to be an inverse correlation between the extreme values for iteration count and run-time, in which combinations often perform well with respect to one and poorly with respect to the other.

Another observation that also applies for SPE10 is the pattern that variants of GMRES appear to follow with regard to iteration count: that, for the same choice of coarsening and smoother, they achieve the same, or very similar, mean values and standard deviations.

When it comes to number of iterations, we have seen that BiCGstab and BiCGstabl combined with Ruge-Stüben coarsening and ILU(l) smoothing, or combined with smoothed aggregation coarsening and ILU(0) smoothing, typically outperform the other combinations. This behaviour has been observed for both the Olympus model and the SPE10 model. When the goal is short run-time, we observe that smoothed aggregation coarsening and ILU(0) as smoother, together with almost any Krylov solver, make up the combinations that are the best (i.e., fastest) by far.

Chapter 8

Conclusions and outlook

In this thesis we have explored a variety of iterative solvers (i.e., combinations) applied to different test models, the most important of which were the SPE10 model and the Olympus model. In addition, we have compared two types of preconditioning, algebraic multigrid (AMG) and relaxation. We have studied how time behaves as a function of the number of grid cells, based on SPE10 results. Finally, we have evaluated mean values and corresponding standard deviations for a selection of Olympus cases and different areas of the SPE10 model. In this final chapter we will discuss all observations and trends noticed during the experiments and, on this basis, attempt to draw conclusions.

8.1 Discussion and conclusion

We have seen a distinct superiority in the performance of combinations using AMG preconditioning as opposed to combinations preconditioning with relaxation (see Section 5.4). This is as expected, since multigrid exploits how relaxation on coarser versions of the original grid can deal with low-frequent error modes which relaxation on the original grid struggles to smoothen out (see Section 4).

In Chapter 6 we saw that the iterative solvers outperform `mldivide` when the number of grid cells increases. This is as expected, being the main reason for inventing iterative solvers in the first place. The reason why the run-time of iterative solvers does not increase as fast as the run-time of `mldivide` when the system increases in size, is that direct solvers require more memory due to, among other things, a large number of expensive operations. The performance of `mldivide` typically behaves like $\mathcal{O}(N^2)$, while the performance of most of the iterative solvers we have considered appears to behave like $\mathcal{O}(N^a)$ for approximately $1 < a < 1.5$.

In Chapter 7 we observed that when combined with the same coarsening and the same smoother, Krylov solvers GMRES, LGMRES and FGMRES appear to behave similarly with respect to number of iterations, yielding very similar mean values and standard deviations. When applied to the SPE10 model, the three variants of GMRES have, generally, performed poorly for both choices of preconditioner. The majority of combinations using variants of GMRES does not reach convergence, at least not within the limit of 5000 iterations. Out of these, LGMRES leads to fewest convergences.

Except for in the simple case in Section 5.4, combinations involving ILUT as smoother have repeatedly proved to perform very well with respect to iteration count but very poorly with respect to run-time. Additionally, in none of the runs have SPAI1 or Chebyshev performed well as smoothers. The iteration counts and run-times for combinations with these smoothers are always among the highest. However, in most tests we recognize a clear trend: the majority of combinations that converged fastest used ILU(0) as smoother. Likewise, the majority of the combinations that achieved convergence in fewest iterations used ILU(l) as smoother. Often, ILU(0) performed best (with respect to run-time) when combined with smoothed aggregation coarsening. ILU(l) typically performed best (with respect to iteration count) when combined with Ruge-Stüben coarsening, but did also perform well combined with smoothed aggregation. This is especially evident in the Olympus cases, e.g., in Figures 5.9 and 5.10. Considering these two figures leads us to question two things:

1. Coarsening with energy-minimizing smoothed aggregation appears to give faster convergence than for ILU(0) with smoothed aggregation. Why is this combination not the best?

2. Using CG as Krylov solver and SPAI0 as smoother appears to give both the fastest run-time and the lowest iteration count. Why is this combination not among the best?

The answers can be found in the result file, where we will notice that all combinations involving energy-minimizing smoothed aggregation and all combinations involving CG and SPAI0 have ”converged in 1 iteration”. What is actually happening is that these combinations, when applied to Case 1, immediately diverge. We can see that the corresponding residuals are logged as NaN (not a number), which often signifies that the number has approached infinity.

8.2 Future work

Having found a handful of combinations of Krylov solvers, coarsenings and smoothers that work well with our test cases, it would be interesting to experiment more thoroughly with these. Especially in the Olympus cases, the standard deviations in the time-plots were quite large, which could be improved with more measurements.

A natural extension of the tests conducted on the Olympus model, where we have only considered one case at a time, would be to combine the results from all cases (1-10) into one large comparison (i.e., as one plot for run-time and one plot for iteration count). This would enable us to evaluate how combinations from a selection based on one case perform with respect to the other cases, giving insight into which combinations perform better than others in general.

Perhaps just as interesting is the experiment exploring time as a function a the number of grid size for the SPE10 case. In particular, one could experiment with even finer grids to see if different iterative solvers scale in different ways.

Further considerations should also be given to the combinations that performed poor in this experiment. Perhaps there are other problems where for instance GMRES is an excellent solver. Looking closer at this could also give us a clue as to why certain combinations perform well on certain problems, while others don't.

Though we in this thesis have focused much on run-time and number of iterations needed before convergence is reached, it would also be of great interest to see how memory usage scales with the size of the problem. Memory and time usage are somewhat related, but most importantly, knowing how much memory is required to do a calculation can help us set requirements for the equipment needed to solve a certain problem. The original plan was to include the exploration of memory usage in the experiments, but we were not able to produce dependable results, partly because memory usage is a somewhat vague concept that is difficult to obtain reliable measures of.

Bibliography

[1] R. Bank et al. *Multigrid Methods*. Society for Industrial and Applied Mathematics (SIAM), 1987. ISBN: 9781611971880.

[2] Sophia Rare Books. *'Numerical Inverting of Matrices of High Order,' pp. 1021-1099 in Bulletin of the American Mathematical Society, Vol. 53, No. 11, November, 1947. [Offered with:] 'Numerical Inverting of Matrices of High Order II,' pp. 188-202 in Proceedings of the American Mathematical Society, Vol. 2, No. 2, April, 1951.* This is from an online book store reviewing the paper 'Numerical Inverting of Matrices of High Order' by John von Neumann and Herman Goldstine. URL: https://www.sophiararebooks.com/pages/books/4641/john-von-neumann-herman-h-goldstine/numerical-inverting-of-matrices-of-high-order-pp-1021-1099-in-bulletin-of-the-american.

[3] Juan Carlos Cabral F. and Christian E. Schaerer. *On Adaptive Strategy For Overcome Stagnation in LGMRES(m, l)*. 2017. URL: http://www.cimapy.org/images/descargas/cabibeskry/CILAMCE2017_Juan_Cabral.pdf.

[4] Gui-Qiang G. Chen. *The Tricomi equation*. URL: http://people.maths.ox.ac.uk/chengq/outreach/The%20Tricomi%20Equation.pdf.

[5] Michael Christie and M.J. Blunt. "Tenth SPE Comparative Solution Project: A Comparison of Upscaling Techniques". In: *SPE Reservoir Evaluation & Engineering* 4 (Feb. 2001). DOI: 10.2118/66599-MS.

[6] R.M. Fonseca, C.R. Geel, and O. Leeuwenburgh. *Description of OLYMPUS reservoir model for optimization challenge*. 2017. URL: http://www.isapp2.com/downloads/olympus-reservoir-model.pdf.

[7] Rajesh Gandham, Kenneth Esler, and Yongpeng Zhang. "A GPU accelerated aggregation algebraic multigrid method". In: *Computers & Mathematics with Applications* 68.10 (2014), pp. 1151–1160. ISSN: 0898-1221. DOI: https://doi.org/10.1016/j.camwa.2014.08.022. URL: http://www.sciencedirect.com/science/article/pii/S0898122114004143.

[8] Martin H. Gutknecht. *A Brief Introduction to Krylov Space Methods for Solving Linear Systems*. ETH Zurich, Seminar for Applied Mathematics. URL: http://www.sam.math.ethz.ch/~mhg/pub/biksm.pdf.

[9] *Handbook of grid generation*. eng. Boca Raton, Fla, 1999.

[10] Thomas Huckle and Matous Sedlacek. "SPAI (SParse Approximate Inverse)". In: *Encyclopedia of Parallel Computing*. Ed. by David Padua. Boston, MA: Springer US, 2011, pp. 1867–1870. ISBN: 978-0-387-09766-4. DOI: 10.1007/978-0-387-09766-4_144. URL: https://doi.org/10.1007/978-0-387-09766-4_144.

[11] Knut-Andreas Lie. *An Introduction to Reservoir Simulation Using MATLAB/GNU Octave*. Cambridge University Press, 2019. ISBN: 9781108492430.

[12] Anders Logg and Kent-Andre Mardal. *Lectures on the Finite Element Method*. 2017. URL: https://folk.uio.no/kent-and/sommerskole/material/book_20april.pdf.

[13] Olav Møyner. *MA8001: Notes on the implementation of multigrid*. 2017.

[14] John von Neumann and Herman Goldstine. "Numerical Inverting of Matrices of High Order". In: *Bulletin of the AMS* 1 (1947). URL: https://pdfs.semanticscholar.org/503b/f08383134ce107d870982fc50f96b80881f7.pdf.

[15] Yousef Saad. *Iterative Methods for Sparse Linear Systems*. 2nd ed. Other Titles in Applied Mathematics. SIAM, 2003. ISBN: 978-0-89871-534-7. URL: https://www-users.cs.umn.edu/~saad/IterMethBook_2ndEd.pdf.

[16] *Scientific Computing and Algorithms in Industrial Simulations: Projects and Products of Fraunhofer SCAI*. eng. Cham: Springer International Publishing, 2017. ISBN: 9783319624570.

[17] Kungliga Tekniska Högskolan (SE). *Partial Differential Equations: Time-Dependent Problems*. Notater fra forelesning i faget Numeriska metoder för mikroelektronikprogrammet. URL: http://www.csc.kth.se/utbildning/kth/kurser/DN1213/numme06/utdelat/kap10.pdf.

Appendix A

Theorems

A.1 Gershgorin circle theorem

Theorem A.1.1 (Gershgorin circle theorem)**.** *Let* $\mathbf{G}$ *be a complex* $N \times N$ *matrix with entries* g_{ij}*. Let* $D(g_{ii}, R_i)$ *be the closed disc centered at* g_{ii} *with radius* R_i*, where*

$$R_i = \sum_{j \neq i} |g_{ij}| \tag{A.1}$$

for $i = 1, ..., N$ *and* $j = 1, ..., N$*. Then every eigenvalue of* $\mathbf{G}$ *lies within at least one of the Gershgorin discs* $D(g_{ii}, R_i)$*.*

A.2 Divergence theorem

The Divergence theorem is also known as Gauss' theorem or Gauss-Ostrogradsky's theorem.

Theorem A.2.1 (Divergence theorem)**.** *Suppose* $V \subset \mathbb{R}^N$ *is compact and has piecewise smooth boundary* ∂V*, and suppose* $\mathbf{F}$ *is a continuously differentiable vector field defined on a neighborhood of* V*. Then:*

$$\int_V (\nabla \cdot \mathbf{F})\, dV = \int_{\partial V} (\mathbf{F} \cdot \mathbf{n})\, d\mathbf{s}. \tag{A.2}$$

The left-hand side of Equation (A.2) is a volume integral over V, while the right-hand side is a surface integral over the boundary of V.

A.3 Levy-Desplanques theorem

Theorem A.3.1 (Levy-Desplanques theorem)**.** *If the matrix* $\mathbf{A} = (a_{i,j})$ *is strictly diagonally dominant, then* $\mathbf{A}$ *is invertible.*

Appendix B

Function spaces

Keep in mind that we are working with real numbers only, not complex, which simplifies many thing. It is assumed that the reader is familiar with the definition of vector spaces, norms and other necessities in order to comprehend the following.

Definition B.0.1 (Cauchy sequence)**.** If $(E, \|\cdot\|_E)$ is a normed vector space and $(v^m)_{m\in\mathbb{N}}$ is a sequence of elements in E satisfying

$$\forall\varepsilon > 0,\ \exists M \text{ such that } \|v^p - v^q\|_E \leq \varepsilon\ \forall p, q \geq M, \tag{B.1}$$

then $(v^m)_{m\in\mathbb{N}}$ is a *Cauchy sequence* in E.

Definition B.0.2 (Banach space)**.** A *Banach space* $(E, \|\cdot\|_E)$ is a normed vector space that is complete with resect to the norm $\|\cdot\|_E$, meaning that Cauchy sequences converge in E.

Definition B.0.3 (Hilbert space)**.** If E is an $\mathbb{R}$-vector space and $\langle\cdot,\cdot\rangle$ is a bilinear form, i.e. satisfying

$$\langle x_1 + x_2, y\rangle = \langle x_1, y\rangle + \langle x_2, y\rangle \tag{B.2}$$

$$\langle x, y_1 + y_2\rangle = \langle x, y_1\rangle + \langle x, y_2\rangle \tag{B.3}$$

$$\langle \lambda x, y\rangle = \lambda\langle x, y\rangle \tag{B.4}$$

$$\langle x, \lambda y\rangle = \overline{\lambda}\langle x, y\rangle \tag{B.5}$$

and positive definite on E, i.e.

$$\langle x, x\rangle > 0\,\forall x \neq 0_E \tag{B.6}$$

then $(E, \langle\cdot,\cdot\rangle)$ is a *pre-Hilbertian space*. If E is complete with respect to the norm defined by $\langle\cdot,\cdot\rangle$, then $(E, \langle\cdot,\cdot\rangle)$ is a *Hilbert space*.

Definition B.0.4 (Support of a function)**.** The *support* of a function f on a domain Ω is the subset of Ω containing the elements that f does not map to zero:

$$\text{supp}(f) := \{\mathbf{x} \in \Omega |\ f(\mathbf{x}) \neq 0\} \tag{B.7}$$

By the Heine-Borel theorem, for $\Omega \subset \mathbb{R}^N$, the support is *compact* if and only if it is a closed and bounded subset of Ω.

B.1 Spaces of continuous functions

$$\mathcal{C}^k(\Omega) = \left\{ f \in \mathcal{C}^0(\Omega) \middle| \, f' \in \mathcal{C}^{k-1}(\Omega) \right\} \tag{B.8}$$

$$\mathcal{C}_c^\infty(\Omega) = \{ f \in \mathcal{C}^\infty(\Omega) \text{ with compact support in } \Omega \} \tag{B.9}$$

If $f \in \mathcal{C}^0(\Omega)$, then f is a continuous function. If $f \in \mathcal{C}^1(\Omega)$, then f and its first derivative f' are continuous functions.

B.2 Lebesgue spaces

$$L^p(\Omega) = \left\{ f \text{ such that } \int_\Omega |f(\mathbf{x})|^p \, \mathrm{d}\mathbf{x} < \infty \right\}, \quad 1 < o < \infty \tag{B.10}$$

with the norm

$$\|\cdot\|_{L^p(\Omega)} = \left(\int_\Omega |u(\mathbf{x})|^p \right)^{1/p} \tag{B.11}$$

There are three special cases: L^1, L^2 and L^∞. Lebesgue spaces are Banach spaces. L^2 is also a Hilbert space, with the inner product

$$\langle f, g \rangle_{L^2(\Omega)} = \int_\Omega f(\mathbf{x}) \, g(\mathbf{x}) \, \mathrm{d}\mathbf{x}, \tag{B.12}$$

and is often referred to as the space of square-integrable functions.

B.3 Sobolov spaces

Sobolev spaces are Hilbert spaces. The general definition is:

$$W^{s,p}(\Omega) = \{ f \in L^p(\Omega) | \, \mathbf{D}^\alpha f \in L^p(\Omega), 1 \le \alpha \le s \} \tag{B.13}$$

We will focus on the type obtained when setting $p = 2$, often referred to as Hilbert-Sobolev spaces:

$$H^s(\Omega) = \left\{ f \in L^2(\Omega) | \, \mathbf{D}^\alpha f \in L^2(\Omega), 1 \le \alpha \le s \right\} = W^{s,2} \tag{B.14}$$

There are more than one possible norm, but a common choice is

$$\|f\|_{H^s(\Omega)} = \sqrt{\sum_{|\alpha| \le s} \|\mathbf{D}^\alpha f\|_{L^2(\Omega)}} \qquad \text{(norm)} \tag{B.15}$$

$$|f|_{H^s(\Omega)} = \sqrt{\sum_{|\alpha| = s} \|\mathbf{D}^\alpha f\|_{L^2(\Omega)}} \qquad \text{(semi-norm)} \tag{B.16}$$

where $\mathbf{D}^\alpha f$ is the distributional derivative of f. It is the *semi-norm* we will see the most of. For us, some central Sobolev spaces are:

$$H^1(\Omega) = \left\{ f : \Omega \to \mathbb{R} | \, f \in L^2(\Omega) \text{ and } f_x, f_y \in L^2(\Omega) \right\} \tag{B.17}$$

$$H_0^1(\Omega) = \left\{ f \in H^1(\Omega) | \, f = 0 \text{ on } \partial\Omega \right\}. \tag{B.18}$$

with inner product

$$\langle f, g \rangle_{H^1(\Omega)} = \int_\Omega fg \, \mathrm{d}\mathbf{x} + \int_\Omega \nabla f \cdot \nabla g \, \mathrm{d}\mathbf{x} \tag{B.19}$$

B.4 Some rules for norms and inner products

From Equation (B.19) we can derive

$$\|f\|^2_{H^1(\Omega)} = \|f\|^2_{L^2(\Omega)} + |f|^2_{H^1(\Omega)} \tag{B.20}$$

Cauchy-Schwarz inequality:

$$\langle f, g \rangle_E \leq \|f\|_E \, \|g\|_E \, , \qquad \text{for inner product space } E \tag{B.21}$$

Generalized Hölder inequality:
For $f \in L^p(\Omega)$ and $g \in L^q(\Omega)$ with $1 \leq p \leq \infty$ and $\frac{1}{r} = \frac{1}{p} + \frac{1}{q}$

$$\|fg\|_{L^r(\Omega)} \leq \|f\|_{L^q(\Omega)} \, \|g\|_{L^p(\Omega)} \, , \tag{B.22}$$

Minkowski inequality:

$$\|f + g\|_{L^p(\Omega)} \leq \|f\|_{L^p(\Omega)} + \|g\|_{L^p(\Omega)} \tag{B.23}$$

Mona-Lena Dordi Norheim

NTNU

Kunnskap for en bedre verden