PRECONDITIONING FOR LINEAR SYSTEMS

Giampaolo Mele
Emil Ringh
David Ek
Federico Izzo
Parikshit Upadhyaya
Elias Jarlebring

Printed by Kindle Direct Publishing

First Printing, 2020

ISBN 9798646306334

www.preconditioning.se

The image in the cover is the mesh used for discretizing the Poisson equation with a satellite geometry as domain. See Problem 1.14 for details.

Table of Contents

Preface **v**

Notation and abbreviations **vii**

1 Iterative solution methods **1**
1.1 Iterative algorithms 2
1.2 Preconditioned iterative algorithms 4
1.3 Convergence theory 6
1.4 Problems 10

2 General preconditioners **17**
2.1 Basic iterative methods 18
2.2 LU factorization and sparse matrices 22
2.3 Incomplete LU factorization 29
2.4 Problems 31

3 Preconditioning for Helmholtz equation **41**
3.1 Damping and oscillations 42
3.2 The two grids method 44
3.3 Shifted-Laplacian preconditioner 45
3.4 Perfectly Matched Layers 46
3.5 Sweeping factorization 48
3.6 Problems 55

4 Domain decomposition **63**
4.1 Non-overlapping domains 64
4.2 Overlapping domains 69
4.3 Problems 75

5 Multigrid **87**
5.1 Introduction to the geometric multigrid 87
5.2 Multigrid methods 92
5.3 Algebraic multigrid 95
5.4 Problems 106

6 Saddle point matrices **115**
6.1 Saddle point systems 116
6.2 Fixed point iterations 119
6.3 Block preconditioners 121
6.4 Problems 123

Solutions to the problems **137**
Solutions to Chapter 1 137
Solutions to Chapter 2 151
Solutions to Chapter 3 172
Solutions to Chapter 4 186
Solutions to Chapter 5 206
Solutions to Chapter 6 226

References **253**

Index **273**

Preface

Solving *linear systems of equations* is a fundamental problem in most scientific fields involving computation, i.e., given $A \in \mathbb{C}^{n\times n}$ and $b \in \mathbb{C}^n$ compute x such that

$$Ax = b. \tag{1}$$

Of specific interest in applications are situations where n is large and the matrix A is sparse. This matrix is often stemming from a discretization of a model of a physical system such as the model on the cover page. Typically, in the large-scale cases, one resorts to iterative methods to compute solutions or approximations to the solution. The standard application of an iterative methods to a difficult application problem, does often not give satisfactory performance. However, many iterative methods can be adapted to take into account additional information, e.g., a method which computes an approximation of the action of A^{-1}. Techniques to form such actions, called preconditioners, is the topic of this book. The design of preconditioners is highly dependent on the origin of the problem. We give an overview of many preconditioners from applications as well as application indenpendent preconditioners.

This book originates from material developed by the students and teacher of the PhD-level course SF3584 (7.5 ECTS) given at the Department of Mathematics at KTH Royal Institute of Technology in Stockholm, during the spring of 2018. The chapters in the book are based on blocks in the course. A substantial part of the work of the course consisted of summarizing the material and formulating problems and solutions, which formed the starting point of the book. The main references for each chapter are listed at the beginning of each chapter. Those, and references therein, can serve as a starting point for a more through treatment of each topic. Before summarizing the material of each chapter, we also provide a short discussion of further literature, which was beyond the scope of the summary and problems. Although all authors of the book have been involved in all parts of the book, each chapter has some designated main authors:

- Chapter 1: Elias Jarlebring
- Chapter 2: David Ek and Emil Ringh
- Chapter 3: Giampaolo Mele and Federico Izzo
- Chapter 4: Giampaolo Mele and Parikshit Upadhyaya
- Chapter 5: Federico Izzo and Emil Ringh
- Chapter 6: David Ek and Parikshit Upadhyaya

Our objective is to provide a consistent introduction to some of the many techniques for preconditioning. The target audience is students, e.g., graduate students in applied and computational mathematics, or researchers in other scientific fields. The summary of the topics should be viewed as a preparation to an deeper study

in the topic, and can be read before the cited articles and books. We provide a large number problems for the material of the chapters. Many problems involve matrices and files from applications. These data files and other supplementary information are available in the official website of the book

`www.preconditioning.se`

Most preconditioners need to take application dependent information and structures into account, and they are therefore often developed with application specific techniques and terminology. Therefore, providing a complete and concise overview of preconditioners is hard. We refer to the summary works of Benzi [33] and Wathen [269]. Although we have tried to collect the most important preconditioning techniques in this book, it is not a complete summary of preconditioners. We now list some topics relevant for preconditioning, not covered in this book.

The sparse approximation and sparse inverses, is a well-studied topic from a number perspectives, as described in the overview papers [44, 182]. High-performance computing aspects and parallelization techniques for sparse approximation can be found in [25, 26, 40, 85, 126, 143, 180], exploitation of block structures and ordering aspects are given, e.g., in [27, 45, 64, 87]. See [34, 41, 90] for algorithm specific research, [1, 43, 183] for applications, and related results in [65, 77, 86, 95, 160–162, 178]. Multilevel methods are also only vaguely covered in this book; see [56, 105, 175, 186, 199, 200, 210, 211, 246, 275, 276]. Polynomial preconditioning is covered in [8, 167, 190, 191]. Further examples of application specific research which can be viewed as preconditioners can be found in [3, 38, 39, 46, 70, 74–76, 115, 158, 173, 205, 241, 242, 268, 270, 277]. Our focus is on large linear systems, although preconditiong techniques are present in other fields as well, such as eigenvalue computation [49, 50, 179, 203, 273], dense linear systems arising in the boundary element method (BEM) [81–83] and ill-conditioned problems [57, 214]. See further preconditioning surveys [12, 71, 233] and the books [52,84]. Most of the preconditioning techniques we present are often combined with Krylov methods, whose theory and property is described, e.g., in [230, 245, 249].

The cover of this page corresponds to the finite element discretization of a satellite. It is described in Problem 1.14 on page 15 and page 148. We are greatful for the comments and ideas about the design of the front page by Kristofer Bohlin.

It has been a pleasure for us to work out the material of this book, and we hope we managed to convey this message such that readers obtain the same pleasure.

Stockholm
May 2020

Giampaolo Mele
Emil Ringh
David Ek
Federico Izzo
Parikshit Upadhyaya
Elias Jarlebring

Notation and abbreviations

Notation

$\mathbb{R}, \mathbb{R}^n, \mathbb{R}^{n\times m}$	set of real numbers, vectors, and matrices
$\mathbb{C}, \mathbb{C}^n, \mathbb{C}^{n\times m}$	set of complex numbers, vectors, and matrices
$a_{i,j}, [A]_{i,j}, A(i,j)$	(i,j) element of the matrix A
v_i	i-th element of the vector v
$A(p_1 : p_2, q_1 : q_2)$	sub-matrix of of A
$\kappa(A)$	condition number of the matrix A
$\sigma(A)$	set of the eigenvalues of A
$\sigma_\varepsilon(A)$	pseudospectra of A
$\text{span}(v_1, \ldots, v_k)$	span defined by vectors $v_1, \ldots, v_k$
$\text{rank}(A)$	rank of a matrix A
$\det(A)$	determinant of the matrix A
$\text{Re}(z)$, $\text{Im}(z)$	real and imaginary part of the complex number z
$\mathcal{K}_m(A, b)$	Krylov subspace
$\lambda_j, \lambda_j(A)$	j-th eigenvalue of A
$\sigma_j, \sigma_j(A)$	j-th singular value of A
$\mathcal{O}(\cdot)$	computational complexity
$\|A\|, \|v\|$	Euclidean norm of the matrix A and vector v
$\|\cdot\|_\infty, \|\|\|_p$	Infinity and p-norm operator
$\|\cdot\|_B$	The B-weighted Euclidean norm

Software

MATLAB	`www.matlab.com`
Python	`www.python.org`
FEniCS	`www.fenicsproject.org`
COMSOL	`www.comsol.com`

Abbreviations

ABC	Absorbing boundary conditions
AMG	Algebraic multigrid
Bicg(Stab)	Biconjugate gradient (stabilized) method
CG	Conjugate gradients method
FD	Finite difference method
FEM	Finite element method
GMRES	Generalized minimal residual method
LU	LU factorization

ILU	Incomplete LU factorization
KKT	Karush–Kuhn–Tucker
PCG	Preconditioned conjugate gradients method
PDE	Partial differential equation
PML	Perfectly matched layers
RHS	Right-hand side
SOR	Successive over relaxation method
SSOR	Symmetric successive over relaxation method
SPD	Symmetric positive definite

Chapter 1

Iterative solution methods

Let us consider the linear systems

$$Ax = b \tag{1.1}$$

where $A \in \mathbb{C}^{n\times n}$ and $b \in \mathbb{C}^n$, for the moment without any specific additional assumptions. The iterative methods we consider in the book generate a sequence of approximations $x_1, x_2, \ldots$ of the solution to (1.1). These methods aim to gradually improve a solution approximation during the iterations until a specified error tolerance is reached. They are usually presented as a complement to direct methods, which lead to an exact solution after a finite number operations, under the exact arithmetic assumption. We summarize several of the common iterative methods and illustrate several ways to characterize the convergence. The success of iterative methods is highly dependent on properties of the matrix, which is the justification for preconditioners, i.e., all the specific techniques in the following chapters.

Literature

This chapter assumes that the reader is already familiar with standard iterative methods in numerical linear algebra [257] or [103].

✎ Convergence bounds for iterative methods:

- Eigenvalue and pseudospectra: [258, pp. 12–23, 244–253], and the software for pseudospectra computation [156]
- Right-hand side dependence: [256]

✎ Preconditioning variations:

- Left-right and split preconditioner: sections 9.2 and 9.3 of [237]
- Flexible GMRES: [232]

Further reading

Iterative methods for linear systems are presented in the book of Saad [237], GMRES convergence results can be found in the books of Greenbaum [149] and Strakoš and Liesen [249] and the review paper of Simoncini and Szyld [245]. Further characterizations can be found in various papers, e.g., bounds based on the field of values in [30, 116, 120]. When the worst case starting vector for GMRES is

considered, we obtain what is called the ideal GMRES minimization problem which has been studied, e.g., in [149] and [187]. More results concerning the convergence of GMRES and related iterative methods can be found in [110, 147, 262]. The flexible GMRES has been adapted to other iterative methods, e.g., CG [212]. Further right-hand side dependent bound characterizations can be found in [7], and [73]. The matrix computation toolbox [156] contains several tools for studying the pseudospectra, among these functions we use `pscont` to compute the contour of the pseudospectra. CG has been adapted and extended to problems with specific structures and settings, see e.g., [2, 128, 141]. CG-specific preconditioners can be found in [41, 93, 171]. See also further work on CG [92, 100, 114, 119, 125, 223] and GMRES [94]. Iterative methods relevant for preconditioning which we do not discuss in this chapter are: BiCGSTAB [263], QMR [131, 132, 253] and Lanczos methods [139, 148, 185]. More information about iterative methods can be found in the books [14, 28, 127, 133, 135, 152, 154] and review papers [177, 198, 215].

1.1 Iterative algorithms

The derivation of many iterative algorithms often stem from searching for solutions in Krylov subspaces.

Definition 1.1 (Krylov subspace)**.** Given $b \in \mathbb{C}^n$, the Krylov subspace is defined as

$$\mathcal{K}_m(A, b) := \text{span}(b, Ab, A^2b, \ldots, A^{m-1}b).$$

The generalized minimum residual method (GMRES), introduced in by Saad and Schultz in [238], is a procedure to find the best approximation of the solution to (1) in a Krylov subspace. The m-th iterate of GMRES satisfies

$$x_m = \underset{x \in \mathcal{K}_m(A,b)}{\text{argmin}} \|Ax - b\| \tag{1.2}$$

The minimizer (1.2) can be computed by solving the linear system obtained by projecting (1) into the Krylov space. This procedure, which is based on the computation of an orthogonal basis to the Krylov space, is summarized in Algorithm 1. The computation of the orthogonal basis of the Krylov space is often a dominating part of the GMRES algorithm. More precisely, in Step 3 of Algorithm 1, at each iteration the orthogonalization of the new vector against all previous basis vectors is peformed. The number of operations per iteration increases with iteration count, and usually eventually becomes significant in comparison to other operations. Namely, the complexity of GMRES depends on the total number of iterations, and therefore slow convergence of the method makes the whole approach unfeasible. Moreover, the orthogonalization step makes the algorithm inherently difficult to properly parallelize, due to many-to-many communication.

Algorithm 1: GMRES - Generalized Minimum Residual Method

Input: Matrix A and vector b

Output: An approximate solution x to the linear system $Ax = b$

1 Set $q_1 = b/\|b\|$, H_0 =empty matrix

for $m = 1, 2, \ldots$ **do**

2 Compute $x = Aq_m$

3 Orthogonalize x against $q_1, \ldots, q_m$ by computing $h = Q^H x$ and setting $x_\perp = x - Qh$.

4 Set $\beta = \|x_\perp\|$

5 Set $q_{m+1} = x_\perp/\beta$

6 Set

$$\underline{H}_m = \begin{pmatrix} \underline{H}_{m-1} & h \\ 0 & \beta \end{pmatrix}$$

end

7 Solve the overdetermined linear system by computing $y_m \in \mathbb{R}^m$ such that:

$$y_m = \underset{z \in \mathbb{R}^m}{\operatorname{argmin}} \|\underline{H}_m z - e_1 \|b\|\|$$

8 Return the approximate solution $x = Q_m y_m$

Two standard approaches to lessen the problems generated by basis orthogonalization are: restarting the method (which we do not consider in this work) and variants which do not require orthogonalization against all vectors; so called short-term-recurrence methods. For symmetric positive definite matrices, the most popular short-term-recurrence method is the *conjugate gradients method* (CG). The CG method can also be viewed as the minimizer of a the residual, but with respect to a different norm. The m-th iterate of CG satisfies

$$x_m = \underset{x \in \mathcal{K}_m(A,b)}{\operatorname{argmin}} \|Ax - b\|_{A^{-1}}. \tag{1.3}$$

Somewhat remarkably, this minimization problem can be solved in a similar way as in GMRES but only requiring orthogonalization against two previous vectors. More precisely, CG also computes an orthogonal basis of the Krylov space and, in each step, the vector Aq_m is already orthogonal to $q_1, \ldots, q_{m-2}$, therefore it should be orthogonalized only with respect to q_m and q_{m-1}. CG can be derived from the application of the Lanczos method (symmetry exploiting method) and a Cholesky factorization. There are many other viewpoints of the CG method [243]. One of the first variants of CG, presented in [155], is given in Algorithm 2.

Algorithm 2: Conjugate Gradient (CG) method

Input: Matrix A and vector b

Output: An approximate solution x to the linear system $Ax = b$

$x_0 = 0$, $r_0 = b$, $p_0 = r_0$

for $m = 1, 2, \ldots$ **do**

  $\alpha_m = \frac{r_{m-1}^T r_{m-1}}{p_{m-1}^T A p_{m-1}}$

  $x_m = x_{m-1} + \alpha_m p_{m-1}$

  $r_m = r_{m-1} - \alpha_m A p_{m-1}$

  $\beta_m = \frac{r_m^T r_m}{r_{m-1}^T r_{m-1}}$

  $p_m = r_m + \beta_m p_{m-1}$

end

Return the approximate solution x_m

1.2 Preconditioned iterative algorithms

In most situations, improvements of the standard iterative methods above are needed in order to make the converge sufficiently fast. In a first approach to achieve this, we view preconditioning as applying the same algorithm to a transformed problem:

- Left preconditioned considers the linear system

$$M_1 A x = M_1 b \tag{1.4}$$

- Right preconditioned considers the linear system

$$A M_2 z = b \tag{1.5}$$

  where $x = M_2 z$.

- Left and right preconditioning considers the linear system

$$M_1 A M_2 z = M_1 b \tag{1.6}$$

  where $x = M_2 z$.

The linear systems (1.4) - (1.5) - (1.6) are referred to as *preconditioned linear systems*. The matrices M_1 and M_2 are called *preconditioners* and need to be chosen such that:

(i) linear systems defined by the preconditioners, i.e., $Mz = c$, can efficiently be solved,

(ii) the algorithms perform better when applied to the associated preconditioned linear system with respect to the original problem (1.1).

In theory, we can apply one of the iterative methods to the preconditioned linear systems. We now illustrate how, for two methods, this can be efficiently done in an algorithmic level without never explicitly compute the matrix defining the preconditioned linear system.

PCG: Preconditioned Conjugate Gradient

The CG method is designed for symmetric positive definite (SPD) problems. Therefore, we now assume that the matrix A is SPD. The modified matrices in (1.4) and (1.5) are, in general, not SPD. Fortunately, if we couple M_1 and M_2 in a particular way, the transformed problem satisfies that assumption of CG. The two-sided preconditioning (1.6) is SPD if $M_1 = M_2^T$ and if M_1 and M_2 are nonsingular. The PCG method can be derived from the application of CG to the preconditioned system (1.6). A transformation allows us to carry out the algorithm without explicitly constructing the preconditioned system. Let us denote by $M = M_1 M_2$ the product between the left and right preconditioner, PCG require an efficient solver for the linear systems defined by M. In particular PCG is derived by replacing, in Algorithm 2, the matrix A with the preconditioned system, and by properly manipulating these steps involving the preconditioned matrices, resulting in Algorithm 3. The full derivation is presented, e.g., in [237, s. 9.2.1].

Algorithm 3: Preconditioned conjugate gradients method

Input: Matrices A and M and vector b

Output: An approximate solution x to the linear system $Ax = b$

$x_0 = 0$, $r_0 = b$, $z_0 = M^{-1}b$, $p_0 = z_0$

for $m = 1, 2, \ldots$ **do**

  $\alpha_m = \frac{r_{m-1}^T z_{m-1}}{p_{m-1}^T A p_{m-1}}$

  $x_m = x_{m-1} + \alpha_m p_{m-1}$

  $r_m = r_{m-1} - \alpha_m A p_{m-1}$

  $z_m = M^{-1} r_m$

  $\beta_m = \frac{r_m^T z_m}{r_{m-1}^T z_{m-1}}$

  $p_m = z_m + \beta_m p_{m-1}$

end

Return the approximate solution x_m

Flexible GMRES

As we will see in later chapters, it may be advantageous to change the preconditioner throughout the iterations. Unfortunately, the standard GMRES method may have slower convergence or stagnate, if one modifies the preconditioned operator throughout the iterations. This is due to the fact that the space spanned by the iterates no longer span a Krylov subspace, and the minimization problem does not longer corresponds to the minimizer of the residual over a subspace. One of the most popular ways to provide the possibility of changing the preconditioner throughout the iterations is called flexible GMRES [232]. This approach is summarized in Algorithm 4, where it is assumed that the preconditioner changes at every iteration. More precisely, M_m denotes the preconditioner used at the m-th iteration. In comparison to standard GMRES, this algorithm requires the storage of one more matrix $Z_m \in \mathbb{C}^{n\times m}$. Moreover, it no longer exhibits the polynomial characterization of the convergence, but it does correspond to the minimization over a specific subspace (which is in general is not a Kryov subspace).

1.3 Convergence theory

Eigenvalue based bounds

The standard bound for GMRES convergence is based on a minimization over polynomials evaluated in the eigenvalues of the matrix A, with a coefficient containing the condition number of the eigenvector matrix. We denote the set of polynomials normalized in zero by

$$\mathcal{P}_m^0 := \{p \text{ polynomial of degree } m \text{ such that } p(0) = 1\}.$$

We denote by $r_m := \|Ax_m - b\|$ the residual of GMRES at the m-th iteration.

Theorem 1.2 (Min-max GMRES-bound)**.** *Suppose that A is diagonalizable with the following eigendecomposition $A = V\Lambda V^{-1}$ and let $r_0, r_1, r_2, \ldots$ be the residuals of GMRES applied to the linear system $Ax = b$. Then,*

$$\frac{\|r_m\|}{\|r_0\|} \leq \|V\|\|V^{-1}\| \min_{p\in\mathcal{P}_m^0} \max_{i=1,\ldots,n} |p(\lambda_i)|. \tag{1.7}$$

To gain further intuition based on this bound, we specialize it by using a result for minimization over a disk, from [231, Lemma 6.26].

Lemma 1.3 (Zarantonello bound)**.** *Let $B(c, r)$ denote a disk centered in $c \in \mathbb{C}$ with radius $r > 0$, i.e.,*

$$B(c, r) := \{\lambda \in \mathbb{C} \text{ such that } |c - \lambda| \leq r\}.$$

Algorithm 4: Flexible GMRES

Input: Matrix A and vector b a starting guess x_0 and $M_1, M_2, \ldots,$ which are a sequence of right preconditioners

Output: An approximate solution x to the linear system $Ax = b$

Compute $r_0 = b - Ax_0$, $\beta_0 = \|r_0\|$ and $r_1 = r_0/\beta_0$

for $m = 1, 2, \ldots$ **do**

  Compute $z_m = M_m^{-1} q_m$

  Compute $w = Az_j$

  Orthogonalize w against $q_1, \ldots, q_m$ by computing $h = Q^H w$ and setting $w_\perp = w - Qh$

  Set $\beta = \|w_\perp\|$

  Set $q_{m+1} = x_\perp/\beta$

  Set

$$\underline{H}_m = \begin{pmatrix} \underline{H}_{m-1} & h \\ 0 & \beta \end{pmatrix}$$

end

Solve the overdetermined linear system by computing $y_m \in \mathbb{R}^m$ such that:

$$y_m = \underset{z \in \mathbb{R}^m}{\operatorname{argmin}} \|\underline{H}_m z - e_1 \beta_0\|$$

Return the approximate solution

$$x = x_0 + Z_m y_m,$$

where $Z_m = [z_1, \ldots, z_m]$.

The polynomial of $\mathcal{P}_m^0$ that takes the minimum value on this disk is explicitly given by

$$\underset{p \in \mathcal{P}_m^0}{\operatorname{argmin}} \max_{z \in B(c,r)} |p(z)| = \frac{(c-z)^m}{c^m}.$$

This result can be combined with the fact that, if the eigenvalues lie in the disk $B(c, r)$, for any function $g(\lambda)$, with domain containing $B(c, r)$, we have

$$\max_{\lambda \in \{\lambda_1, \ldots, \lambda_n\}} g(\lambda) \leq \max_{\lambda \in B(c,r)} g(\lambda).$$

Therefore, we obtain the following bound which is useful if the eigenvalues are clustered close to one point, far from the origin in a relative sense.

Corollary 1.4 (GMRES disk bound)**.** *With the same assumptions as in Theorem 1.2, assume that all eigenvalues are contained in a disk $B(c, r)$, namely*

$\lambda_i \in B(c,r)$ for $i = 1,\ldots,n$. Then,

$$\frac{\|r_m\|}{\|r_0\|} \leq \|V\|\|V^{-1}\|\frac{r^m}{|c|^m}$$

The bound in the above corollary does not involve the starting vector, and also not the b-vector. The convergence will certainly depend on this. Variations of the bound that take this into account can be found in [256]. The following bound, which is a slightly modified formulation of [256, Theorem 2.2], illustrates that the individual eigenvalue contributions can be weighted in a way that depends on essentially the vector $V^{-1}b$, if $r_0 = b$.

Theorem 1.5 (RHS-dependent eigenvalue bound)**.** *Under the same assumptions as Theorem 1.2, let $w := V^{-1}r_0/(\|V^{-1}\|\|r_0\|)$, then*

$$\frac{\|r_m\|}{\|r_0\|} \leq \|V\|\|V^{-1}\| \min_{p\in P_m^0} \left(\sum_{i=1}^{n} |w_i|^2|p(\lambda_i)|^2\right)^{1/2} \tag{1.8}$$

This can be related to the min-max bound (1.7) as follows. Note that

$$\begin{aligned}\sum_{i=1}^{n} |z_i|^2 = \|z\|^2 = \|\operatorname{diag}(z)^2 e\| &\leq \|\operatorname{diag}(z)^2\|\|e\|^2 \\ &= \|e\|^2 \max_{i=1,\ldots,n} |z_i|^2 = n \cdot \max_{i=1,\ldots,n} |z_i|^2\end{aligned}$$

where $e = [1,\ldots,1]^T$. If we select $z_i = |w_i| \cdot |p(\lambda_i)|$ we can further bound the quantity in (1.8) obtaining min-max expressions similar to Theorem 1.2 but with weighting:

$$\frac{\|r_m\|}{\|r_0\|} \leq \sqrt{n}\|V\|\|V^{-1}\| \min_{p\in P_m^0} \max_{i=1,\ldots,n} |w_i||p(\lambda_i)| \tag{1.9}$$

Note that w satisfies

$$Vw = r_0/(\|V^{-1}\|\|r_0\|)$$

which means that it can be interpreted as the weighting coefficients of the starting residual in terms of the eigenvectors. For instance, if we do not need eigenvector j to express the starting residual, we have $w_j = 0$ and that eigenvalue will not contribute to the min-max expression in the bound (1.9). Similarly, if the contribution of the direction of an eigenvector in the starting residual is small, the corresponding eigenvalue will carry less weight in the bound (1.9).

Pseudospectra based bounds

The convergence of many iterative methods can be further characterized by the pseudospectra.

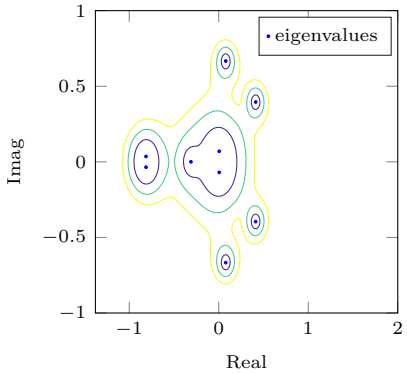


Figure 1.1: Pseudospectra of a radom matrix of size 10 for ε equal to $10^{-1.5}, 10^{-1.2}$ and 10^{-1}.

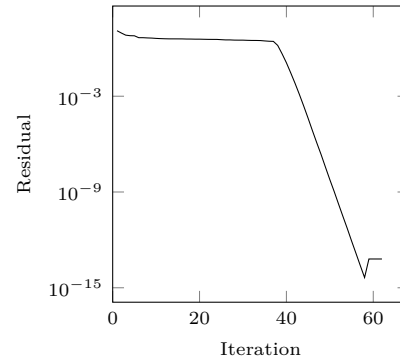


Figure 1.2: Convergence of GMRES with transient phase.

Definition 1.6 (Pseudospectra)**.** Given $\varepsilon > 0$ the pseudospectra of the matrix A is defined as

$$\sigma_\varepsilon(A) := \{z \in \sigma(A + E) \text{ such that } \|E\| < \varepsilon\}.$$

Observe that $\sigma(A) \subseteq \sigma_\varepsilon(A)$ for every value of ε. The pseudospectra describes how sentive is the spectrum of the matrix A with respect to perturbations. An example of a pseudospectra (of a small random matrix) is given in Figure 1.1, computed with `pscont` [156]. The convergence of GMRES can be described with a min-max bound involving the pseudospectra.

Theorem 1.7 (Pseudospectra convergence bound)**.** *Let L_ε the length of the contour of $\sigma_\varepsilon(A)$, then*

$$\frac{\|r_m\|}{\|r_0\|} \leq \frac{L_\varepsilon}{2\pi\varepsilon} \min_{p \in \mathcal{P}_m^0} \max_{i=1,\ldots,n} |p(\lambda_i)| \tag{1.10}$$

The above theorem states that GMRES may have a *transient phase*. The rate of convergence in the beginning of the iteration may be substantially different than when the iterate is close to a solution. This initial phase, usually referred to as *transient phase*, is characterized by the pseudospectra of the matrix, namely by the sensitivity of the eigenvalues. After the transient phase, GMRES enters in the *asymptotic regime* and it converges with a converge rate which in general described by Corollary 1.4. We illustrate these concepts in Figure 1.2 where GMRES is applied to a matrix with the following block structure:

$$A = \begin{pmatrix} A_1 & 0 \\ 0 & A_2 \end{pmatrix}$$

where $A \in \mathbb{R}^{500\times 500}$, $A_1 \in \mathbb{R}^{30\times 30}$ is a companion matrix with random coefficients and $A_2 \in \mathbb{R}^{470\times 470}$ has eigenvalues clustered close to one. The spectrum of companion matrices is very sensitive with respect to small perturbations, i.e., $\sigma_\varepsilon(A_1)$ is a large set even for small ε. As expected the stagnation phase takes around 30 iteration due to the pseudospectra of A_1. After 40 iterations GMRES converges with convergence rate given by Theorem 1.2 and/or Corollary 1.4.

1.4 Problems

Problem 1.1. Consider the following two definitions of pseudospectra

- Definition 1

$$\sigma_\varepsilon := \left\{ z \in \mathbb{C} \text{ such that } \|(zI - A)^{-1}\| > \varepsilon^{-1} \right\}$$

- Definition 2

$$\sigma_\varepsilon := \left\{ z \in \mathbb{C} \text{ such that } z \in \sigma(A + E) \text{ with } E \in \mathbb{C}^{n\times n} \text{ and } \|E\| \leq \varepsilon \right\}$$

(a) One of the two definitions is not correct. Identify and correct the wrong definition.

(b) Prove that the two definitions, after the correction in (a), are equivalent.

Problem 1.2. Consider the following definition for pseudospectra given in [258],

$$\sigma_\epsilon(A) = \{ z \in \mathbb{C}^n, \|(zI - A)v\| < \epsilon, \text{ where } v \in \mathbb{C}^n, \ \ \|v\| = 1 \}$$

Prove that this definition is equivalent to the second (corrected) definition of pseudospectra in Problem 1.1.

Problem 1.3. Compute the 50 largest and smallest eigenvalues of the K-matrix from problem `capacitor_tunable_piezo_domain_K.mat`

(a) Consider the linear system $Kx = b$. Why GMRES is expected to have slow convergence for this problem?

(b) Consider the shifted linear system $(K - I)x = b$. Is the convergence of GMRES expected to be faster with respect to (a). Why?

Problem 1.4. *Discovering properties of pseudospectra*

(a) Prove that $\sigma_{|c|\varepsilon}(cA) = c\sigma_\varepsilon(A)$ where $c \in \mathbb{C}$.

(b) Let D_ε be the disk (in the complex plane) centered in zero with radius ε. For which class of matrices it holds $\sigma_\varepsilon(A) = \sigma(A) + D_\varepsilon$?
Hint: start testing this properties for diagonal matrices, Jordan blocks, 2x2 matrices, etc. No closed answer required. If it is too hard, just show that $\sigma(A) + D_\varepsilon \subseteq \sigma_\varepsilon(A)$.

(c) It is known that if A and B are diagonalizable and commute, i.e., $AB = BA$, then $\sigma(A) + \sigma(B) = \sigma(A + B)$. Is this true for the pseudospectra? More precisely is $\sigma_\varepsilon(A) + \sigma_\varepsilon(B) = \sigma_\varepsilon(A + B)$? If yes prove it, otherwise show a counterexample.

Problem 1.5. In this problem we look at eigenvalue sensitivity, pseudospectra, and GMRES convergence.

(a) Consider the following MATLAB code:

```
[Q,~]=qr(rand(n,n));
a=[-30-3*rand(n/3,1);-20-2*rand(n/3,1);-10-2*rand(n/3,1)];
A=Q*diag(a)*Q';
A=(A+A')/2; S=balance(compan(poly(a)));
```

This script creates a "well-behaved" matrix A which is symmetric and with prescribed eigenvalues which are distributed in three regions of the complex plane. The matrix S is a balanced version of the companion matrix to A, and hence mathematically it has the same eigenvalues as A. Thus, from an eigenvalue analysis, GMRES should have similar convergence behavior. However, the operations involved in constructing S are not well conditioned.

How can we see this if we compute the eigenvalues and the pseudospectra? Try $n = 12$ and $n = 21$, a plot like

```
pscont(A, 4, 100, [-40,0,-4,4]))
```

which is a function provided by the toolbox [156].

(b) Now instead consider the MATLAB code:

```
[Q,~]=qr(rand(n,n));
a=[-30-3*rand(n/3,1);-20-2*rand(n/3,1);-10-2*rand(n/3,1)];
S=balance(compan(poly(a)));
aa=eig(S); A=Q*diag(aa)*Q';
```

This script is similar as above in (a), but we know that that A and S have the same eigenvalues, not only in exact arithmetic but also numerically. Create a random right-hand-side and test the GMRES convergence behavior.

What can we say? How does the GMRES convergence look like? How does the pseudospectra of A and S look like? This is slightly more stable and hence we can consider n = 66, a plot like

```
pscont(A, 4, 100, [-80,0,-45,45], -10:1:0).
```

Problem 1.6. *Krylov subspaces properties and RHS*
We want to use GMRES for solving $Ax = b$ where $A \in \mathbb{R}^{n\times n}$ is diagonalizable.

(a) Assume that b can be expressed as a linear combination of $m \ll n$ eigenvectors of A (for example $m = 10$ eigenvectors). After exactly how many iterations GMRES converges?

(b) Let W an invariant space for A, i.e., $AW \subseteq W$ with $\dim(W) = m \ll n$. Assume that $b \in W$. After exactly how many iterations GMRES converges?

Hint: these properties directly come from the Krylov space structure. Try to answer the question for $m = 1$ and then generalize.

Problem 1.7. *Learning RHS-bounds on your own. This is suitable to be solved after a careful reading of [256].*

Due to the minimization property of GMRES, the residual can be expressed as

$$\|r_m\| = \min_{p \in \mathcal{P}_m^0} \|p(A)b\|.$$

Let consider the eigendecomposition $A = Z\Lambda Z^{-1}$. Let $w = Z^{-1}b/\|b\|$.

(a) Show that $p(A)b = ZWp(\Lambda)e$. What is W? what is e?

(b) Show that $\|r_m\| \leq \|Z\| \min_{p \in \mathcal{P}_m^0} \|Wp(\Lambda)e\|$. What are the elements of $Wp(\Lambda)e$ explicitly?

(c) How is the derivation related to the derivation in [256]? (It's not exactly the same.)

Problem 1.8. Run GMRES on a diagonal matrix consisting of the elements `1:n`.

```
>> n=100;
>> A=spdiags((1:100)',0,100,100);
>> b=randn(100,1);
```

The MATLAB implementation of GMRES (without restarting) can be run in this way in order to get a convergence diagram:

```
>> [X,FLAG,RELRES,ITER,RESVEC] = gmres(A,b,length(b));
>> semilogy(RESVEC)
```

We obtain the convergence diagram in Figure 1.3.

One of the curves is with random RHS the other with

```
>> b=zeros(n,1); b(50:60)=1;
```

(a) Which RHS correspond to which curve? Relate to theory for RHS-convergence of GMRES

(b) If we take RHS: `b=zeros(n,1); b(90:100)=1;` Do we get faster or slower convergence. Relate to RHS-convergence theory.

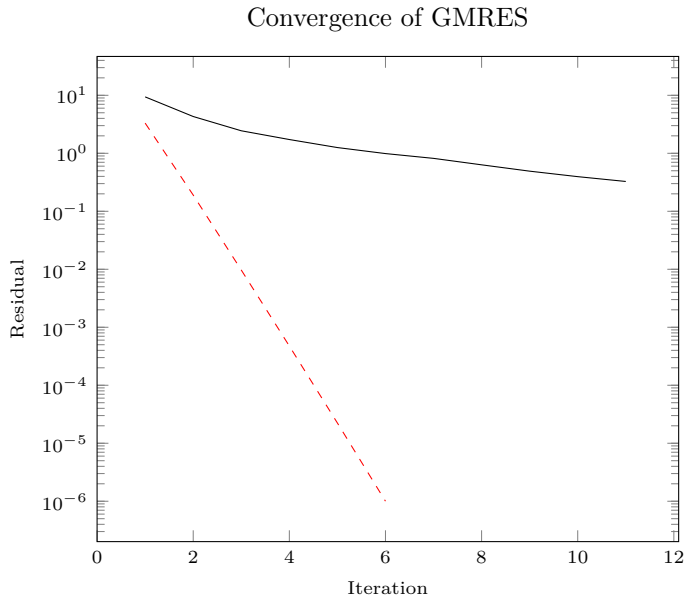


Figure 1.3: Convergence of GMRES applied to the diagonal matrix in Problem 1.8.

Problem 1.9. *Flexible GMRES*
Equation (1) in Saad's paper [232] states

$$AZ_m = V_m \underline{H}_m$$

Prove that this relation reduces to the standard Arnoldi relation if $M_j = M$ for $j = 1, \ldots, m$.

Problem 1.10. *Flexible GMRES with BiCGSTAB as preconditioner.*
We want to solve in MATLAB the system $Ax = b$ with $A \in \mathbb{R}^{n \times n}$ with flexible GMRES. As preconditioner we use BiCGSTAB with variable number of iterations.

(a) Complete the following MATLAB script of flexible GMRES. More precisely as preconditioner $M_j x$ we consider the function $M(j, x)$ that perform j steps of BiCGSTAB.

```
n=200; e = ones(n,1);
A = spdiags([e 2*e e], -1:1, n, n);
b=rand(n,1); norm_b=norm(b);
V(:,1)=b/norm_b;
m=30;
M=@(j,b) ........................... ;
for j=1:m
```

```
    Z(:,j)=M(j,.................);
    V(:,j+1)=.................;
    h=V(:,1:j)'*V(:,j+1);
    V(:,j+1)=V(:,j+1)-V(:,1:j)*h;
    g=V(:,1:j)'*V(:,j+1);
    V(:,j+1)=V(:,j+1)-V(:,1:j)*g;
    H(1:j,j)=h+g;
    H(j+1,j)=norm(V(:,j+1));
    V(:,j+1)=V(:,j+1)/H(j+1,j);
    e1=eye(j+1); e1=e1(:,1);
    y=H\(norm_b*e1);
    xx=..........;
    err(j)=norm(A*xx-b);
end
semilogy(err);
```

(b) Assume that we perform in total $m = 30$ iterations of flexible GMRES. Now change the preconditioner in the previous script in a way that $M(j, x)$ does $m - j + 1$ steps of BiCGSTAB. Does the situation change? Notice that the total complexity of the two script is the same.

(c) Now change the preconditioner in a way that $M(j, x)$ does m steps of BiCGSTAB (notice that this is still not a constant preconditioner). Does the situation change? Notice that the total complexity is now increased. Do we get any benefit in comparison with (b)?

(d) As you have noticed in the previous points (a) and (b), if we have the preconditioners $M_1, M_2, M_3, \ldots$ the order in which we use them is important. After the point (a) and (b), can you reach any conclusion? It is better to use "better" preconditioners at the first iterations and "worse" preconditioners at the last iterations or vice-versa? Do you think this is a general property? Test these ideas on a more changing matrix A.

Note: BiCGSTAB requires, as memory $\mathcal{O}(n)$ independently on the number of iterations, whereas GMRES requires $\mathcal{O}(mn)$ memory. However, for many problems BiCGSTAB may not work but still be a valid preconditioner and GMRES may require too many iterations (and memory) without the usage of a preconditioner.

Problem 1.11. Let us consider the linear system $Ax = b$. Consider the preconditioned conjugate gradient method, given in Algorithm 9.1 of [237, Ch. 9], with the symmetric positive definite preconditioner $M = LL^T$, where L is the Choleksy factor. Show that this method is equivalent to CG applied to the preconditioned system $L^{-1}AL^{-T}u = L^{-1}b$ where $u = L^T x$.

Problem 1.12. Show that the spectra of the operators corresponding to left, right and split preconditioning are identical, i.e., show that if the preconditioning matrix M is factorized as $M = LU$ then $M^{-1}A$, AM^{-1} and $L^{-1}AU^{-1}$ have the same eigenvalues.

Problem 1.13. Setup a sparse linear system of equations using FEM-solver system FeNICS. You will need to install the Python framework for FeNICS. This tutorial may be of use
`https://fenicsproject.org/pub/tutorial/sphinx1/._ftut1004.html`
This is not really related to this chapter, but the problem setup will serve as useful example in later problems.

Problem 1.14. Setup a sparse linear system of equations to use as benchmark problem using COMSOL. Examples and comsol-multiphysics files:

`https://www.comsol.com/model/12385`
`https://www.comsol.com/model/8577`
`https://www.comsol.com/model/3576`
`https://www.comsol.com/model/847`

Load in MATLAB the "eliminated stiffness matrix" from the file `satellite.mph` which contains the discretization of the Poisson equation with domain the satellite illustrated in the cover of this book. Test CG and GMRES applied to the linear system defined by such matrix with random right-hand side.

Problem 1.15.

(a) Derive the following convergence bound on GMRES

$$\frac{\|r_m\|}{\|r_0\|} \leq \min_{p \in \mathcal{P}_m^0} \|p(A)\|.$$

(b) By using the Cauchy integral formula [157, Definition 1.11], and what you proved in the previous point, derive following convergence bound on GMRES

$$\frac{\|r_m\|}{\|r_0\|} \leq \frac{L_\varepsilon}{2\pi\varepsilon} \min_{p \in \mathcal{P}_m^0} \max_{z \in \Gamma_\varepsilon} \|p(z)\|.$$

This convergence bound is related to Theorem 1.7. Recall that the Cauchy integral formula states that: if $\Gamma \subset \mathbb{C}$ in a closed smooth curve containing all the eigenvalue of $A \in \mathbb{C}^{n \times n}$ and if $f(z)$ is analytic inside the contour of Γ, then it holds

$$f(z) = \frac{1}{2\pi i} \oint_\Gamma f(z)(zI - A)^{-1} dz.$$

Problem 1.16. Suppose that $A \in \mathbb{C}^{n \times n}$ is diagonalizable as $A = Z\Lambda Z^{-1}$. Show the the following bound on the residual of GMRES holds

$$\frac{\|r_m\|}{\|r_0\|} \leq \|K\| \|Kw\| \operatorname{cond}(K^{-1}Z) \min_{p \in \mathcal{P}_m^0} \max_{\lambda \in \sigma(A)} |p(\lambda)|$$

where $w = r_0/\|r_0\|$ and K is any invertible matrix.

Problem 1.17. Show that the eigenvalues of the matrix H_m, obtained by extracting the first m rows and columns from $\underline{H}_m$ generated in Algorithm 1 (GMRES), belongs to the pseudospectra $\sigma_{H_m}(A)$.

Problem 1.18. Given $A \in \mathbb{C}^{n\times n}$ we define the field of values as

$$W(A) := \{z \,|\, z = v^* A v,\ v \in \mathbb{C}^n,\ \|v\|_2 = 1\}$$

Show that $\sigma_\varepsilon(A) \subseteq W(A) + D_\varepsilon$.

Problem 1.19. We apply GMRES to the linear system $Ax = b$ with preconditioned P. Is it true that a necessary condition for P to be a good preconditioner is that $P \approx A^{-1}$?

Problem 1.20. Given $M, P \in \mathbb{C}^{n\times n}$ such that $\sum_{j=0}^{m} M^j = I$, let us consider $A = MP$. We apply GMRES to a linear system defined by the matrix A and we use P as left preconditioner. Explain why GMRES will converge at most after m iterations.

Chapter 2

General preconditioners

This first chapter on preconditioners contains a summary of several techniques which can be useful on their own, but also form building blocks. The preconditioners in this chapter do not take the origin of matrix into account, although the effectiveness of the preconditioners depend heavily on certain matrix structures, such as diagonal dominance or sparsity patterns. Most of the basic versions of the techniques we present, e.g., Gauss–Seidel iterations and variations of the LU-factorization, were developed quite long ago and were used as methods to solve linear systems. These techniques are still very important, since approximate variations can be effective preconditioners and reappear in other forms in later chapters.

Literature

✎ Iterative methods:

- Classical iterations: sections 11.1 and 11.2 of [140]
- Basic iterative methods and splitting: sections 4.1 and 4.2 of [237]
- Special matrices: section 4.1 and 4.2 of [140]

✎ Sparse LU factorization:

- Graphs and matrices: sections 3.1 and 3.3 of [237]
- Incomplete LU factorization: sections 10.3 and 10.4 of [237]

Further reading

Iterative methods presented in this chapter are deeply analyzed in the books of Greenbaum [146] and Barret [28]. Some of the methods discussed are based on the concept of M-matrices. This class of matrices is often defined in different ways depending on the context. Many equivalent definitions are collected in [218]. Other references on M-matrices are [51, 197]. More literature about ILU [24, 37, 54, 55, 58, 60, 80, 88, 89, 106, 121, 163, 164, 174, 234–236, 239, 240, 266, 271, 276] and other incomplete factorizations [13, 15, 23, 42, 47, 78, 117, 118, 150, 165, 170, 172, 181, 188, 189, 195, 208, 225, 229, 252, 255, 260, 261], reordering/permutations [9,62,66,98,99,107,111,137,138,169,213,275]. More methods based on reducing the bandwidth by permutations are presented in [4, 36, 96, 112–114, 192, 247]. Classical iterative methods such as Jacobi, Gauss–Seidel, SOR, etc, and their variations are deeper analyzed in [10].

2.1 Basic iterative methods

In this section we consider linear systems of the form

$$Ax = b, \qquad \text{where } A \in \mathbb{R}^{n\times n}, b \in \mathbb{R}^n. \tag{2.1}$$

In particular we assume that n is large and it is computationally expensive to use direct methods for solving (2.1), e.g., Gaussian elimination, but reasonably cheap to compute the matrix-vector product with A, e.g., because A is sparse or has other exploitable structures. Basic iterative methods arise from situation where it is desired to solve, or approximately solve, linear systems on the form (2.1). The idea is to start from an initial approximation of the solution and successively improve one or several components of the solution at each iteration. The techniques in the basic methods might not be used so often by themselves, however they constitute some important ideas which combined with other methods can make them more efficient. Here we give a brief introduction to mainly three basic iterative methods together with some theoretical results which provide convergences guarantees. For a throughout introduction we refer to the material mentioned above.

Jacobi, Gauss–Seidel and SOR

The three most well-known basic iterative methods are *Jacobi*, *Gauss–Seidel* and *successive over relaxation* (SOR). These methods are introduced in this section on "component form" and on "matrix form" to illustrate their individual structure and their differences. The matrix form will then also be used to put these basic iterative methods in the context of fixed-point iterations for which an extensive theoretical foundation exist. The matrix form utilizes the decomposition

$$A = D - E - F, \tag{2.2}$$

where D is the diagonal of A, $-E$ and $-F$ is it the strict lower triangular part respectively the strict upper triangular part, i.e.,

$$A = \begin{pmatrix} \ddots & & -F \\ & D & \\ -E & & \ddots \end{pmatrix}.$$

Equation (2.1) can on component form be written as

$$a_{ii}x_i + \sum_{j\neq i} a_{ij}x_j = b_i, \quad i = 1, \ldots, n, \tag{2.3}$$

and with the decomposition of (2.2) as $Dx - Ex - Fx = b$.

Jacobi

In Jacobi iterations a sweep over all components i is performed at every iteration k. The update on component form is obtained by solving for x_i in (2.3) and can be written as

$$x_i^{k+1} = \frac{1}{a_{ii}} \left(-\sum_{j \neq i} a_{ij} x_j^k + b_i \right), \quad i = 1, \ldots, n,$$

or equivalently

$$x^{k+1} = D^{-1}(E + F)x^k + D^{-1}b.$$

Gauss–Seidel

Note that for components $1 < i \leq n$ newer information is available in terms of the previously computed components. The idea in Gauss–Siedel is to use this information instead, this leads to the component form

$$x_i^{k+1} = \frac{1}{a_{ii}} \left(-\sum_{j=1}^{i-1} a_{ij} x_j^{k+1} - \sum_{j=i+1}^{n} a_{ij} x_j^k + b_i \right), \quad i = 1, \ldots, n, \tag{2.4}$$

which can be written as

$$Dx^{k+1} = Ex^{k+1} + Fx^k + b.$$

Solving for x^{k+1} gives the Gauss–Seidel matrix form update

$$x^{k+1} = (D - E)^{-1}Fx^k + (D - E)^{-1}b.$$

Successive over relaxation (SOR)

The SOR update can be viewed in various ways. One is to relate it to the component-wise Gauss–Seidel update in (2.4). Addition and subtraction of x_i^k on the right-hand-side of (2.4) gives

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

A relaxation parameter ω is then introduced as

$$x_i^{k+1} = x_i^k + \omega \frac{1}{a_{ii}} \left(-\sum_{j=1}^{i-1} a_{ij} x_j^{k+1} - \sum_{j=i}^{n} a_{ij} x_j^k + b_i \right), \quad i = 1, \ldots, n,$$

which gives the SOR component form

$$x_i^{k+1} = (1-\omega)x_i^k + \omega\frac{1}{a_{ii}}\left(-\sum_{j=1}^{i-1} a_{ij}x_j^{k+1} - \sum_{j=i+1}^{n} a_{ij}x_j^k + b_i\right), i = 1,\ldots,n. \quad (2.5)$$

The second term in the right-hand-side of (2.5) is (2.4), i.e. the Gauss–Seidel update. In consequence, the SOR update may be viewed as $x_i^{k+1} = (1-\omega)x_i^k + \omega x_i^{GS}$, $i = 1,\ldots,n$, where x_i^{GS} is given by (2.4). The choice $\omega = 1$ gives Gauss–Seidel whereas, $0 < \omega < 1$ corresponds to an under relaxed update and $1 < \omega < 2$ corresponds to a over relaxed update. For the corresponding matrix form, note that (2.5) is with the decomposition of (2.2) equivalent to

$$Dx^{k+1} = (1-\omega)Dx^k + \omega(Ex^{k+1} + Fx^k + b).$$

Solving for x^{k+1} gives the SOR matrix form update

$$x^{k+1} = (D-\omega E)^{-1}\left(\omega F + (1-\omega)D\right)x^k + \omega(D-\omega E)^{-1}b. \quad (2.6)$$

Another approach is to consider (2.1) scaled by ω, i.e. $\omega Ax = \omega b$, while noting that

$$\omega A = \omega(D-E-F) + D - D = (D-\omega E) - (\omega F + (1-\omega)D).$$

With the corresponding partition of components this directly gives the SOR update in (2.6). A symmetric successive over relaxed (SSOR) can also be defined and consist of one regular SOR step combined with a backward step as

$$\begin{aligned} x^{k+1/2} &= (D-\omega E)^{-1}\left(\omega F + (1-\omega)D\right)x^k + \omega(D-\omega E)^{-1}b, \\ x^{k+1} &= (D-\omega F)^{-1}\left(\omega E + (1-\omega)D\right)x^{k+1/2} + \omega(D-\omega F)^{-1}b. \end{aligned} \quad (2.7)$$

Fixed-point and precondition view-point

Basic iterative methods can also be viewed as fixed-point iterations. The fixed-point view yields a straightforward connection to preconditioning and convergence results. Let M, N define the splitting such that $A = M - N$. Equation (2.1) is then equivalent to $Mx + Nx = b$. If M is nonsingular, solutions may be generated as

$$x^{k+1} = M^{-1}Nx^k + M^{-1}b. \quad (2.8)$$

The choice $G = M^{-1}N$ and $f = M^{-1}b$ gives

$$x^{k+1} = Gx^k + f. \quad (2.9)$$

Let us define the linear function $\phi(x) := Gx + f$; if x^k converges to x with (2.9) then x is a fixed point of ϕ and satisfies

$$x = Gx + f, \text{ or } (I-G)x = f. \quad (2.10)$$

By using that $(I - G) = I - M^{-1}N = I - M^{-1}(M - A) = M^{-1}A$, (2.10) can be written as

$$M^{-1}Ax = M^{-1}b. \tag{2.11}$$

The fixed-point iteration (2.8), or equivalently (2.9), may thus be seen as a fixed-point iteration on the preconditioned system (2.11).

The relation between specific splitting matrices M, N and the decomposition matrices D, E, and F in Section 2.1 for the different methods are summarized in Table 2.1.

Table 2.1: Relation between the different splittings of A and G.

	M	N	G
Jacobi	D	$E + F$	$I - D^{-1}A$
Gauss–Seidel	$D - E$	F	$I - (D - E)^{-1}A$
SOR	$\frac{1}{\omega}(D - \omega E)$	$\frac{1}{\omega}(F + (1 - \omega)D)$	$I - \omega(D - \omega E)^{-1}A$

In general there are not many restrictions on how to choose the splitting $A = M - N$ as long as M is nonsingular and the sequence (2.8) is convergent. Primarily, the aim is to choose M such that it approximates A in some sense and such that equations on the form $Mz = d$ are relatively easy to solve.

Convergence

The fixed-point view and its standard convergence results provide conditions on the splitting matrices M, N, and hence also on G, for convergence. In consequence, these results provide conditions for convergence of the basic iterative methods of Section 2.1.

A standard illustrative example, which can be found in Saad [237, s. 4.2.1], is the following. A fixed point x^* satisfies (2.10). Subtraction of (2.9) from (2.10) gives

$$x^{k+1} - x^* = G(x_k - x^*) = \cdots = G^{k+1}(x^0 - x^*),$$

and it follows that x^k converges to x^* if and only if the spectral radius of G is less than one. This result is formalized in the following theorem.

Theorem 2.1 ([237, Theorem 4.1])**.** *Let $G \in \mathbb{R}^{n\times n}$ be such that $\rho(G) < 1$. Then $I-G$ is nonsingular and the iteration* (2.9) *converges for any f and x_0. Conversely, if the iteration* (2.9) *converges for every f and x_0, then $\rho(G) < 1$.*

The above result can also be stated in terms of the splitting matrices M and N as follows.

Theorem 2.2 ([140, Theorem 11.2.1])**.** *Suppose $A = M - N$ is a splitting of a nonsingular matrix $A \in \mathbb{R}^{n\times n}$. Assuming M is nonsingular then the iteration* (2.8) *converges to $x = A^{-1}b$ for all x_0 if and only if $\rho(M^{-1}N) < 1$.*

Theorem 2.1 and Theorem 2.2 provide necessary and sufficient conditions on M, N and G such that the fixed-point iterations of (2.8) respectively (2.9) converge. The spectral radius may be computationally expensive to compute. Instead, sufficient conditions can be considered, e.g. $\rho(M^{-1}N) \leq \|M^{-1}N\| < 1$, which holds for any norm. Alternatively, conditions such that $\rho(M^{-1}N) < 1$ can also be given with the concept of regular splittings, see Definition 2.3. One such condition is given in Theorem 2.4 below.

Definition 2.3 (Regular splitting)**.** For a given matrix $A \in \mathbb{R}^{n\times n}$ let the splitting M, N be defined by $A = M - N$. The splitting M, N is a regular splitting of A if M is nonsingular and M^{-1}, N are non-negative.

Theorem 2.4 ([237, Theorem 4.4])**.** *Let $A = M - N$ be a regular splitting of a nonsingular matrix $A \in \mathbb{R}^{n\times n}$. Then $\rho(M^{-1}N) < 1$ if and only if A is nonsingular and A^{-1} is nonnegative.*

From Theorem 2.4 it follows that the iterations of (2.9) converges if M, N is a regular splitting of an *M-matrix*, see Definition 2.5.

Definition 2.5 (M-matrix)**.** A matrix $A \in \mathbb{R}^{n\times n}$ is an *M-matrix* if $A = sI - B$, where $b_{ij} \geq 0$ and $s \geq \rho(B)$, where $\rho(B)$ is the spectral radius of B, i.e., the maximum moduli of the eigenvalues of B.

2.2 LU factorization and sparse matrices

In this section *sparse matrices* and *preservation of sparsity* will be key. This is simply because for a general matrix $A \in \mathbb{R}^{n\times n}$ a matrix vector product costs $\mathcal{O}(n^2)$ operations for a dense matrix, but for a sparse matrix the cost is proportional to the number of non-zero elements. Hence there is potentially a lot to gain by preserving sparsity in the computations.

LU factorization and fill-in

The goal of the LU factorization is to write a matrix $A \in \mathbb{R}^{n\times n}$ as the product of an upper triangular matrix $U \in \mathbb{R}^{n\times n}$ and a unit lower triangular matrix $L \in \mathbb{R}^{n\times n}$, where with unit lower triangular we mean that the diagonal elements are 1. The goal is hence to write $A = LU$ and, instead of solving a linear system with A, we solve two linear systems with L and U. Triangular linear systems can be efficiently solved by backward substitution. The LU factorization appears in connection with Gaussian elimination. In each step of the elimination, there is a leading entry which is called *pivot*. This pivot needs to be non-zero in order to continue the elimination. Although the matrix is nonsingular, it might be the case that some pivots will be zero and hence there are techniques, called pivoting, that rearranges the rows and the columns to continue the elimination. From a

numerical point of view a small but non-zero element may also cause trouble. However, we will not consider this further and henceforth only consider pivoting as a tool for minimizing fill-in, see below. In the case of a non-pivoted matrix the following can be said.

Theorem 2.6 ([140, Theorem 3.2.1])**.** *If $A \in \mathbb{R}^{n\times n}$ and $\det(A(1:k,1:k)) \neq 0$ for $k = 1, 2, \ldots, n$. Then there exist a unique unit lower triangular $L \in \mathbb{R}^{n\times n}$ and upper triangular $U \in \mathbb{R}^{n\times n}$ such that $A = LU$.*

Specifically one can also talk about the structure of L and U.

Proposition 2.7. *Under the assumptions of Theorem 2.6 consider the factorization $A = LU$. Then U is the upper triangular matrix created in the first pass of Gaussian elimination. Furthermore, the k-th column of L contains the multipliers arising in the k-th step of the first pass the elimination.*

In the context of LU factorization and sparse matrices, the following definition will be important to characterize performance.

Definition 2.8 (Fill-in)**.** Consider a matrix $A \in \mathbb{R}^{n\times n}$ under the assumptions of Theorem 2.6, and let $A = LU$ be the unique LU factorization. The *fill-in* is defined as the number of non-zero elements in $|L| + |U|$ minus the number of non-zero elements in A, where $|\cdot|$ denotes the element-wise absolute value.

The reason for the element-wise absolute value in Definition 2.8 is because we are not interested in the sum $L + U$, but rather a measure of how many more non-zero elements we have after factorization. The terminology *a fill-in* is also used to describe an element which is zero in A but non-zero in $|L| + |U|$. A low fill-in is desired since that means a smaller amount of memory and computation power is required to treat the problem. Note that a fill-in in one step of the algorithm may be cancelled in a later step. However, this seldom happens in practice. The fill-in can be determined without knowledge of the exact values of the matrix, but only of the non-zero pattern. Moreover, the fill-in depends on the order of rows and columns. Namely, fill-in can be reduced by properly permuting the rows and columns of the matrix A. This is illustrated in the following example.

Example 2.9 (The arrow matrix [140, s. 11.1.3][237, ex. 3.3])**.** Consider the two matrices

$$A = \begin{pmatrix} a & b & c & d & e \\ b & f & 0 & 0 & 0 \\ c & 0 & g & 0 & 0 \\ d & 0 & 0 & h & 0 \\ e & 0 & 0 & 0 & i \end{pmatrix} \qquad \text{and} \qquad \hat{A} = \begin{pmatrix} i & 0 & 0 & 0 & e \\ 0 & f & 0 & 0 & b \\ 0 & 0 & g & 0 & c \\ 0 & 0 & 0 & h & d \\ e & b & c & d & a \end{pmatrix}.$$

The matrices are similar, even permutation-similar. However, if one applies Gaussian elimination to A, the resulting factors L and U are full matrices. Even

worse, after only one step in the elimination, the remaining matrix is full. On the contrary, for the matrix $\hat{A}$ there is very little computation needed to complete the Gaussian elimination and the result will have no fill-in. Hence there is a lot of extra storage and computation power required to do an LU factorization of A compared to one of $\hat{A}$, although they are equivalent representations of the same linear system.

Motivated by the enormous differences in Example 2.9, the problem at hand is this to find a permutation matrix P that minimizes the fill-in in the LU factorization of PAP^T. However, this is a combinatorial optimization problem and thus expensive to solve. Hence we resort to heuristics based on graph theory. From now we assume that the matrix A is an M-matrix, see Definition 2.5.

Sparse matrices and graphs

In order to get to the main points, we start by introducing some common concepts from graph theory, and naturally we start with the graph itself.

Definition 2.10 (Graph, nodes, and edges)**.** Consider a finite set of *nodes*, $V = \{v_1, v_2, \ldots, v_n\}$. An *edge*, $e = (v_i, v_j)$, defines a relation between the two nodes $v_i, v_j \in V$. A *graph* is defined as the pair $G = (V, E)$.

We denote by $|V|$ the cardinality of V, i.e., the number of elements in V. The set V can contain any objects, and we will think of them as labels of the nodes in the graph. Without loss of generality we can assume $V \subset \mathbb{N}$ and for our situation we will think of it as $V = \{1, 2, \ldots, n\}$ for some n. Also note that $E \subseteq V \times V$. On a higher level one can distinguish two main types of graphs, directed and undirected.

Definition 2.11 (Undirected graph)**.** A graph is called *undirected* if $(v_i, v_j) \in E$ implies that $(v_j, v_i) \in E$ for all $v_i, v_j \in V$.

Definition 2.12 (Directed graph)**.** A graph is called *directed* if it is not undirected. A directed graph is also known as a *digraph*.

The undirected graph can be described as the relation depicted by E. The set E has a symmetry property, i.e., the pair $e = (v_i, v_j) \in E$ is unordered. Whereas the directed graph represents a non-symmetric relation. For a directed graph there is hence a difference between an *outgoing edge*, leaving the node, and an *incoming edge*, entering the node. It is possible to define multiple matrices related to a graph, we will make use of the adjacency matrix.

Definition 2.13 (Adjacency matrix)**.** Consider a graph $G = (V, E)$ with $|V| = n$, its *adjacency matrix* $H \in \mathbb{R}^{n \times n}$ G is defined as

$$a_{ij} = \begin{cases} 0 & (v_i, v_j) \notin E \\ \text{non-zero} & (v_i, v_j) \in E. \end{cases}$$

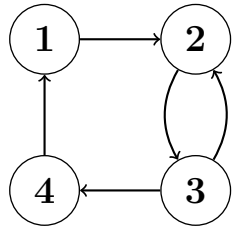


Figure 2.1: The directed graph from Example 2.14. What does the adjacency matrix look like?

In some cases the non-zero elements are taken to be all 1:s in which case we have a *binary adjacency matrix*, and in other they are representing some weight of the corresponding edge. Note that an undirected graph gives rise to a symmetric adjacency matrix, and a directed graph to a non-symmetric. From the definition we can see that an outgoing edge from node i is represented by a non-zero element in row i and an incoming edge to node i is represented by a non-zero element in column i.

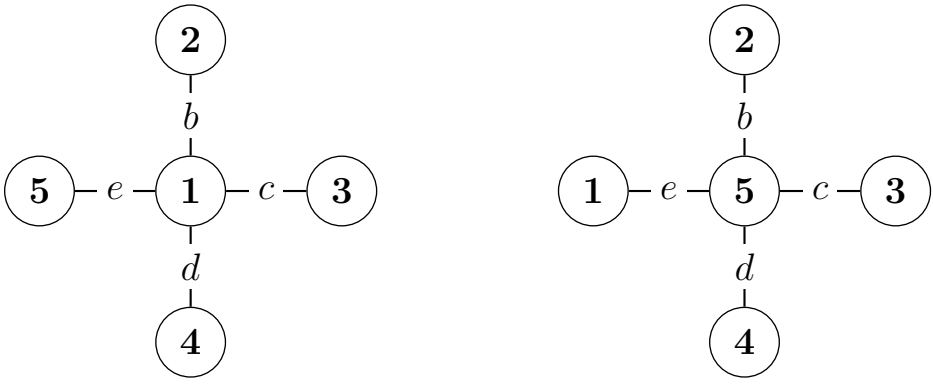
Example 2.14. Consider the graph in Figure 2.1. We have that $V = \{1, 2, 3, 4\}$, and $E = \{(1, 2), (2, 3), (3, 2), (3, 4), (4, 1)\}$. The graph is directed and the adjacency matrix H is given by

$$H = \begin{pmatrix} * & * & 0 & 0 \\ 0 & * & * & 0 \\ 0 & * & * & * \\ * & 0 & 0 & * \end{pmatrix}.$$

In analogy with the adjacency matrix, one can also define the graph of a matrix.

Definition 2.15 (Graph of a matrix)**.** Consider a matrix $A \in \mathbb{R}^{n \times n}$. The graph $G = (V, E)$ associated with the matrix A is defined by $V = \{1, 2, \ldots, n\}$ and $(i, j) \in E$ if and only if the element $a_{ij} \neq 0$.

Since we are interested in M-matrices, we have matrices with non-zero diagonals. Technically this means that in the graph we should have edges $(v_i, v_i) \in E$ for all $v_i \in V$. However, these are typically not drawn explicitly but implicitly assumed to be there. Intuitively we can understand that a sparse matrix has a corresponding graph with few edges. It is thus of interest to preserve this sparsity among edges during an elimination, and in the next section we will get back to how elimination in a matrix affects the graph. First we discuss topics related to connectivity of a graph.



(a) Graph of matrix A. (b) Graph of matrix $\hat{A}$.

Figure 2.2: Graphs of the matrices A and $\hat{A}$ from Example 2.9.

Definition 2.16 (Path)**.** A *path* in a graph is a sequence of nodes in the graph, $v_{j_1}, v_{j_2}, \ldots, v_{j_{k+1}} \in V$, such that $(v_{j_i}, v_{j_{i+1}}) \in E$ for $i = 1, 2, 3, \ldots, k$, and all the nodes are unique, i.e., $v_{j_i} \neq v_{j_\ell}$ for $i \neq \ell$. If the path consists of $k+1$ nodes, i.e., passes k edges, it is said to be of length k.

Definition 2.17 (Distance)**.** The distance between v_i and v_j is defined as

$$d(v_i, v_j) = \min \{k \text{ such that it exists a path of length } k \text{ between } v_i \text{ and } v_j.\}$$

With these definition we then say that graph is *connected* if there is a path between any pair of nodes and *disconnected* if it is not connected. Specifically and undirected graph is disconnected if and only if the corresponding adjacency matrix is permutation-similar to a block-diagonal matrix. Another interesting property is described in the following proposition, from which the intuition will be useful when discussing level-of-fill in the context of ILU in later sections.

Proposition 2.18. *Let H be a binary adjacency matrix. Then element (i, j) of H^k is non-zero if and only if there exists at least one path of length k between node i and node j.*

Proof. See Problem 2.14. □

Example 2.19 (Graph of the arrow matrix)**.** Consider the arrow matrix A from Example 2.9, as well as the permuted matrix $\hat{A}$. The graph of A is given in Figure 2.2(a) and the graph of $\hat{A}$ in Figure 2.2(b). From Example 2.9 we know that A and $\hat{A}$ are permutation similar, and comparing the graphs in Figure 2.2(a) and 2.2(b) we can see that these are very similar.

In the example above we can note the following feature. Reordering rows and columns in the matrix is equivalent to renumber the nodes in the graph. This idea leads to the following proposition, which will have a key role in the reordering techniques in the next section.

Proposition 2.20 (Symmetric permutation and graphs)**.** *Let H be an adjacency matrix and P a permutation matrix. Then $B = PHP^T$ has the same graph as H but with the labels reordered.*

Proof. See Problem 2.15. □

Reordering techniques

Given a matrix A and a permutation P we have seen that A and PAP^T can have vastly different fill-in, and that the corresponding graphs are the same but with relabelling of the nodes. We now connect the dots with how Gaussian elimination affects the graph of a matrix and how heuristics based on relabelling techniques in graphs can be used to construct efficient permutation matrices P. We start with the former.

Gaussian elimination starts from lower numbered nodes, and works towards higher numbered nodes. We observe the process after $k-1$ steps and are interested in the remaining matrix of size $(n-k) \times (n-k)$, and we want it to be sparse. Consider an index $\ell > k$. In the elimination step, row ℓ will keep all its non-zero elements (no cancellation) and the non-zero elements of row k will be introduced. In terms of fill-in, one can thus say that fill-in in created when row k eliminates row ℓ, and there are non-zero elements in row k that do not exist in row ℓ (observe that we always assume that the diagonal elements are non-zero). Hence, considering matrix A from Example 2.9, one step of the LU factorization will give a lot of fill-in since the first row has many non-zero elements of which most are zero in the rows below. Equivalently, from a graph perspective, we are interested in the remaining graph where the first $k-1$ nodes has been removed and edges modified accordingly, and we want this to have few edges, i.e., in some sense be sparse. In the elimination step, node ℓ will keep all its previous neighbours (no cancellation) and get all the neighbours of node k. Hence the fill-in introduced corresponds to node k giving new neighbours to node ℓ. Thus in the context of Example 2.19 we get that A will give a lot of fill-in since in the graph the node labelled "1" has many neighbours which the following nodes do not have.

Minimum degree ordering

The *minimum degree ordering* is a greedy algorithm. It tries to minimize the fill-in in each step of Gaussian elimination by simply choosing the node with smallest (out-)*degree*, i.e. node with fewest (outgoing) edges. This node is moved to the top, the elimination, or a simulation thereof, is done, and the algorithm proceeds.

An equivalent motivation of the heuristics is that in each step the pivot is chosen as the row with fewest non-zero elements. This method can be implemented either as a pre-processor to find a permutation before the elimination, or it can be used on-line during the elimination.

Reverse Cuthill–McKee ordering

The reverse Cuthill–McKee ordering is based on the Cuthill–McKee ordering and the observation that reversing the order suggested by the latter typically results in less fill-in. The Cuthill–McKee ordering is a heuristic to minimize the *profile* of the matrix.

Definition 2.21 (Profile)**.** For a symmetric matrix $A \in \mathbb{R}^{n\times n}$ the *profile*, $f(A)$, is defined as

$$f(A) := n + \sum_{i=1}^{n}(i - f_i(A)),$$

where $f_i(A) := \min\{j : 1 \leq j \leq i, a_{ij} \neq 0\}$.

Example 2.22. Consider the matrices A and $\hat{A}$ from Example 2.9. These have profiles $f(A) = 5 + (1-1) + (2-1) + (3-1) + (4-1) + (5-1) = 15$, and $f(\hat{A}) = 5 + (1-1) + (2-2) + (3-3) + (4-4) + (5-1) = 9$. Hence although the matrices have the same bandwidth, $\hat{A}$ has the non-zero elements further down, and hence also further to the right, which results in a smaller profile.

Essentially the algorithm for computing the Cuthill–McKee ordering is a breadth-first search of the graph, given some initial starting node. The difference is in how the starting node is chosen and the exact order in which the nodes are visited. The breadth-first search will implicitly construct sets of nodes, called *level sets*, based on the shortest path from the starting node to the corresponding nodes. Specifically, in the Cuthill–McKee algorithm these level sets are explicitly constructed and used to achieve the ordering. The algorithm is presented as Algorithm 5.

Different versions of the algorithm have different ways of choosing the starting node in Step 1. Some ways to choose a starting node are:

- Greedy choice: start with the node of smallest (out-)degree
- Brute force: test all possible starting nodes and use the one which gives the smallest profile
- Greedy/Brute: test the $m \ll n$ nodes with smallest (out-)degrees and choose the one which gives the smallest profile

Algorithm 5: Cuthill–McKee

input : A graph $G = (V, E)$
output : A list with the new ordering

Choose a starting node v and label it "1"
Set $S_0 = \{v\}$, $i = 0$, $k = 2$
while $S_i \neq \emptyset$ **do**
    $S_{i+1} = \emptyset$
    Sort S_i in descending order based on out-degree
    for $v \in S_i$ **do**
        Label v with the lowest available number k
        Increment $k = k + 1$
        $S_{i+1} = S_{i+1} \cup \{\text{Neighbours of } v \text{ not previously labelled}\}$
    end
    $i = i + 1$
end
return A list with the mapping from old label to new label

- Pseudoperipheral Node: A *peripheral node* is a node who's maximum distance to another node in the graph is equal to the *diameter*[1] of the graph. A *pseudoperipheral node* has a maximum distance to another node which is close to the diameter. Figuratively speaking these nodes are "far away" from the center of the graph. One way of finding such nodes is to start from a random node, do a breadth-first-search to find the node furthest away, choose this node and repeat until the maximum distance found is no longer increasing. For further reference, see, e.g., the paper [136].

2.3 Incomplete LU factorization

As the name suggests, the *incomplete LU factorization*, also known as *ILU*, is an inexact factorization of a matrix $A \in \mathbb{R}^{n \times n}$ in the sense that $A = LU - R$, where R is a non-zero residual matrix. The goal is to, by removing steps and computations from the Gaussian elimination, create a cheaply computable, yet hopefully efficient, preconditioner.

The simplest ILU factorization is known as ILU(0) and it can be loosely described as not allowing any fill-in. More precisely, the ILU(0) is computed using Gaussian elimination with the extra criteria that if $a_{ij} = 0$ then all operations involving the index pair (i, j) are skipped. That is to say, none of the fill-in generated in the full Gaussian elimination is regarded at any step of the computation of ILU(0). In more general terms it can also be phrased as ILU(0)

[1] The diameter is the maximum distance between two nodes.

and A has the same *zero pattern*. In relation to incomplete Gaussian elimination of A we define a zero pattern $\mathcal{P}$ as a set of index pairs (i, j) such that; if $(i, j) \in \mathcal{P}$, then all operations involving the index pair (i, j) are skipped in the Gaussian elimination. In this context we define the $\text{ILU}_{\mathcal{P}}$ as the ILU produced using the zero pattern $\mathcal{P}$. This class of factorizations is broad, and it is worth to point out that some factorizations might not be well defined, since the Gaussian elimination may break down, e.g., if $(i, i) \in \mathcal{P}$ for some i. However, we do know the following.

Theorem 2.23 ([237, Theorem 10.2])**.** *Let $A \in \mathbb{R}^{n\times n}$ be an M-matrix, and let $\mathcal{P}$ be a zero pattern such that $\mathcal{P} \subset \{(i, j)|i \neq j, 1 \leq i, j \leq n\}$. Then the $ILU_{\mathcal{P}}$ factorization $A = LU - R$ is well defined and is a regular splitting of A.*

Hence, as long as the zero pattern does not involve the diagonal, the $\text{ILU}_{\mathcal{P}}$ factorization can be used as a preconditioner. However, what is a good choice of zero pattern? One choice is the ILU(p) factorizations, where $p = 0$ generates the ILU(0), and where higher p means a more accurate factorization such that for high enough $p < \infty$ we get back the full LU factorization. To define these ILU(p) factorizations we go back to the concept of fill-in produced in Gaussian elimination.

Definition 2.24 (Level of fill)**.** Given a matrix $A \in \mathbb{R}^{n\times n}$, the *level of fill* is an $n \times n$ integer matrix defined recursively as

$$\text{lev}_{ij} := \begin{cases} 0 & a_{ij} \neq 0 \text{ or } i = j \\ \infty & \text{otherwise,} \end{cases}$$

and each time the element is touched in the Gaussian elimination it is updated as

$$\text{lev}_{ij} = \min\{\text{lev}_{ij}, \text{lev}_{ik} + \text{lev}_{kj} + 1\}.$$

In the book of Saad [237] the level of fill is motivated by the fact that the size of the elements should correspond to the level of fill, since the updates in the Gaussian elimination process is of the type $\alpha_{ij} = \alpha_{ij} - \alpha_{ik}\alpha_{kj}$, which we schematically can think of as $\alpha^{\text{lev}_{ij}} = \alpha^{\text{lev}_{ij}} - \alpha^{\text{lev}_{ik}}\alpha^{\text{lev}_{kj}}$, with $\alpha < 1$ from diagonal dominance of the M-matrices. For further intuition, the level of fill can be though of in a shortest-path sense, cf. triangle operation and the Floyd–Warshall algorithm in, e.g., [216]. The further we are, in some sense, form an original element, the smaller the fill-in will be. The following example gives a similar characterization.

Example 2.25. Given a matrix $A \in \mathbb{R}^{n\times n}$ assumed to have a unique LU factorization $A = LU$, and let $\hat{A} = |L| + |U|$. Then we have the following interpretation of lev_{ij} for A.

- $\text{lev}_{ij} = 0$ if and only if the corresponding element a_{ij} is non-zero
- $\text{lev}_{ij} = \infty$ if and only if $\hat{a}_{i,j} = 0$, i.e., no fill-in occurred at this position

- Assuming no cancellation, then level of fill is $0 < \text{lev}_{ij} < \infty$ if and only if $a_{ij} = 0$ and $\hat{a}_{i,j} \neq 0$, i.e., fill in has occurred. More specifically:
  - If $\text{lev}_{ij} = 1$, then the fill-in was generated directly by a non-zero element in A
  - If $\text{lev}_{ij} = 2$, then the fill-in was generated indirectly, by an element which is itself a fill-in, i.e., with $\text{lev}_{ik} = 1$

Note that, the level of fill does not take into account how many non-zero elements which affect the current element. Neither does it reflect if any other element with higher level of fill is modifying the given element. Nevertheless, the idea about the level of fill presented in Example 2.25 can in general be characterized using the following concept of a fill-path.

Definition 2.26 (Fill-path)**.** Given a graph with nodes $V = \{1, 2, \ldots, n\}$. A *fill-path* is a path between two nodes i and j such that all intermediate nodes k fulfil $k < \min\{i, j\}$.

Theorem 2.27 ([237, Theorem 10.6 and 10.7])**.** *Given a matrix and its corresponding graph. Fill-in occurs during Gaussian elimination at element (i, j) if and only if there exists a fill-path between i and j. Moreover, $lev_{ij} = p < \infty$ if and only if there exists a fill-path between i and j and the shortest such fill-path is of length $p + 1$.*

With the level of fill defined we now define the ILU(p) as an ILU$_{\mathcal{P}}$ with the specific zero pattern $\mathcal{P} = \{(i, j) | \text{lev}_{ij} > p\}$. In other words, the Gaussian elimination only considers computations with elements that are, in the sense of level of fill, close to an original element of the matrix A.

2.4 Problems

Problem 2.1. The iterations in Jacobi, Gauss–Seidel and SOR are often described as $Mx_{k+1} = Nx_k + b$.

(a) Describe how M and N relate to A for the different methods. What is required for convergence?

(b) The equation can be rewritten as a fixed-point iteration. How does it look like? How does the classical convergence criterion for fixed-point iterations relate to the one above?

(c) Fixed-point iterations of this type can be rewritten as a *Neumann series*, i.e., as $y_k = \sum_{i=0}^{k} z_i$, where $z_0 = M^{-1}b$ and $z_i = M^{-1}Nz_{i-1}$ for $i = 1, 2, \ldots$. Show that $y_k = x_k$ if the fixed-point iteration is started with $x_{-1} = 0$.

Problem 2.2. *Jacobi method for diagonally dominant matrices.*
A matrix A is called *diagonally dominant* if $|a_{i,i}| > \sum_{j=1 j\neq i}^{n} |a_{i,j}|$.

(a) Show that a sufficient condition for the convergence of the Jacobi method is the diagonally dominance of the matrix A.

(b) The above is a sufficient condition for convergence. Show that it is not a necessary condition, i.e., find an example where this condition is not satisfied but Jacobi still converges.

Problem 2.3.

(a) What is a regular spliting?

(b) What is an M-matrix?

(c) How are the concepts in (a) and (b) related?

Problem 2.4. Carry out the Jacobi and the Gauss–Seidel method for this problem

```
n=10000; e = ones(n,1);
A = spdiags([e e -5*e e e], -2:2, n, n);
b=sin(pi*(1:n)');
```

Test also an hybrid method which consists of alternating one iteration of Jacobi and one iteration of Gauss–Seidel. How many iterations are required to reach accuracy 10^{-5}? Provide the source code.

Problem 2.5. Given

$$B = \begin{pmatrix} 62/45 & 91/45 & -103/45 \\ 91/45 & 421/90 & -194/45 \\ -103/45 & -194/45 & 409/90 \end{pmatrix} \qquad a = \begin{pmatrix} 2/3 \\ 1/3 \\ 2/3 \end{pmatrix}$$

let us consider the following iteration given as MATALB code

```
v = a; v = v/norm(v)
for i = 1:8
  v = -B*v + a;
end
```

(a) Do the iterations converge? To what should it converge?

(b) What happens when 20 iterations are performed?

(c) What can we say in relation to Theorem 11.2.1 in [140]? Do we have a contradiction? Which criteria are fulfilled and which are not? Try a different starting vector. What happens? For which vectors does it work?

Hint: Consider the eigenvalues and eigenvectors of B. You can motivate your reasoning in (b) by for example run the iteration in symbolic (put a "`sym(...)`" around the definition of B and a)

Problem 2.6. Carry out the Jacobi and Gauss–Seidel method for the problem defined by the following MATLAB code

```
n=100;
e = ones(n,1);
A = spdiags([e e -5*e e e], -2:2, n, n)
b=sin(pi*(1:n)')
```

Plot the residual during the ierations. Test the following different choices for starting vector $x_0 = [1, \dots, 1]$ and $x_0 = b$ and discuss the results.

Problem 2.7. Consider the matrix A and the vector b generated by the MATLAB code

```
n=100
A=diag(100*ones(1,n))+diag(10*ones(1,n-1),1) ...
+diag(10*ones(1,n-1),-1)-randn(n);
b=rand(n,1);
```

and consider the fixed point iteration (2.8) applied to the two splitting given by the MATLAB code

```
M1=diag(diag(A)); N1=M1-A;
M2=diag(diag(A))+diag(diag(A,-1),-1)+diag(diag(A,1),1); N2=M2-A;
```

(a) To which method is equivalent the fixed point iteration (2.8) with the splitting $A = M_1 - N_1$?

(b) Can the convergence of fixed point iteration (2.8) with the splitting $A = M_1 - N_1$ be justified by a specific structure of the matrix A?

(c) Plot the converge of the fixed point iteration (2.8) for both splittings.

(d) Can the convergence of fixed point iteration (2.8) with the splitting $A = M_2 - N_2$ be justified by a specific structure of the matrix A?

Problem 2.8. The classic SOR method is defined as $x_{k+1} = (D - \omega E)^{-1}(\omega F + (1-\omega)D)x_k + \omega(D - \omega E)^{-1}b$. The *backward SOR* is instead defined as $x_{k+1} = (D - \omega F)^{-1}(\omega E + (1-\omega)D)x_k + \omega(D - \omega F)^{-1}b$. The Symmetric SOR (SSOR) is defined as a half-step with SOR and another half-step with backward SOR. Find the recurrence relation $x_{k+1} = G_\omega x_k + f_\omega$ and find the matrix M of the relation $Mx_{k+1} = Nx_k + b$ for the SSOR.

Problem 2.9. *The penalty method for solving PDEs with the matrix splitting.* A way to solve PDEs with complicated domains consists of imposing the boundary conditions with the penalty method. In this problem we will go through this application and we will show how this method naturally provides a matrix splitting.

Let us consider the following problem

$$\begin{cases} \Delta u(x,y) = 1 & (x,y) \in \Omega \\ u(x,y) = 0 & (x,y) \in \partial\Omega \end{cases}$$

Where Ω is a complicated domain such that $\Omega \subseteq [-1,1]^2$. An easy way to approximate the solution of this PDE is to introduce the function

$$k(x,y) = \begin{cases} \tilde{k} & (x,y) \notin \Omega \\ 0 & (x,y) \in \Omega \end{cases}$$

where $\tilde{k}$ is a large number, e.g., 10^6. Then the solution of the original PDE can be approximated with the solution of

$$\begin{cases} \Delta\tilde{u}(x,y) + k(x,y)\tilde{u}(x,y) = 1 & (x,y) \in [-1,1]^2 \\ \tilde{u}(x,y) = 0 & (x,y) \in \partial[-1,1]^2 \end{cases}$$

We will consider $\Omega = [-1,1] \setminus \left\{x^2 + y^2 \leq 1/10\right\}$

(a) Show that the finite difference discretization of the last PDE can be written in the form $Ax = b$ with $A = M - N$. Give an expression for M and N. A hint can be found in the next question.

(b) Complete the following MATLAB code for computing the matrices M,N and the vector b and solve the modified PDE.

```
n=100; e = ones(n,1);
xx=linspace(-1,1,n); yy=linspace(-1,1,n); h=xx(2)-xx(1);
D = ..............................;
I = speye(n);
M = kron(D,I)+kron(I,D);
kk=1e5; k=@(x,y) kk*(x.^2+y.^2<1/10);
f=@(x,y) ...........;
[XX,YY] = meshgrid(xx,yy); KK=k(XX,YY);
N = -spdiags(KK(:), 0, n^2, n^2);
F=.................;
u=(M-N)\F(:); uu=reshape(u,n,n);
imagesc(xx,yy,uu); colorbar
```

(c) The linear system is naturally defined as a splitting. Does the iterative method $Mx_{k+1} = Nx_k + b$ converges?

(d) Use the same method for the equation $(x+y)\Delta u(x,y) = 1$ with the same domain Ω. Does the iterative method $Mx_{k+1} = Nx_k + b$ converges? (You need to compute the new matrices M and N)

Observe that it is very fast to solve the linear system $My = z$ since the matrix M is very sparse and structured. Moreover, in a direct discretization of the original PDE, the computation of the discretized problem is more complicated.
A sharp answer is not required. It is fine to simply show some numerical experiments and presenting an argument that justify the results. Try to use different values of penalty parameter $\tilde{k}$ and different discretization parameter n (number of discretization points).

Problem 2.10. Consider the matrix A generated by the following script

```
n=1000; e = ones(n,1);
A = spdiags([e 5*e e], -1:1, n, n); A = A^2;
```

Consider the linear system $Ax = b$ with b random vector, compare the convergence of the iterative methods defined by the following matrix splittings:

- M_1 = diagonal part of A and $N_1 = M_1 - A$,
- M_2 = tridiagonal part of A and $N_2 = M_2 - A$.

Problem 2.11. *A naive sparse LU factorization method*
By using the ideas presented in Section 2.2, we can derive a naive algorithm for performing a sparse LU factorization. More precisely, given a matrix A, we permute the columns and rows with a permutation matrix P such that $PAP^T = LU$ with L and U sparse.

The algorithm we propose is based on the following idea. We follow the standard LU factorization method with a small variation. In the generic i-th iteration of the LU method, before doing the Gaussian elimination of the current sub-matrix, we sort the columns from the more sparse to the less sparse. More precisely, in the Algorithm 3.2.1 of the Golub and Van Loan book [140], we sort the columns of the matrix `A(i:n,i:n)` from the more sparse to the less sparse.

(a) You will need to implement a MATLAB function `S=sparsity_pattern(A)` such that, given the matrix `A`, the output of this function is the matrix `S` defined as

$$S(i,j) = \begin{cases} 1 & A(i,j) \neq 0 \\ 0 & A(i,j) = 0 \end{cases}.$$

A quick way to implement this with handle functions is the following

`sparsity_pattern=@(A) not(A==0).`

(b) Complete the following MATLAB function (or write your own) that implements the naive sparse LU factorization

```
function [L, U] = sparse_lu(A)
n = size(A, 1); L = eye(n);
for k = 1 : n
    AA=A(...........); % select the proper submatrix
    [~,idx]=sort(sum(sparsity_pattern(AA)),...............);
    AA=AA(idx,idx);
    A(k:n,k:n)=AA;
    L(k + 1 : n, k) = A(k + 1 : n, k) / A(k, k);
    for l = k + 1 : n
        A(l, :) = A(l, :) - L(l, k) * A(k, :);
    end
end
U = A;
end
```

(c) Compute the naive sparse LU factorization of a random sparse matrix. Compare with the standard LU implementation of MATLAB. How does the naive algorithm perform? You can run the following script in MATLAB for doing this test

```
n=200;
A = sprand(n,n,0.01)+speye(n); A=full(A);
[L, U] = sparse_lu(A);
subplot(1,3,1)
spy(A)
title("A")
subplot(1,3,2)
spy(abs(L)>1e-6)
title("sparse L")
subplot(1,3,3)
spy(abs(U)>1e-6)
title("sparse U")

figure
[L, U, P] = lu(A);
subplot(1,3,1)
spy(A)
title("A")
subplot(1,3,2)
spy(abs(L)>1e-6)
title("L")
subplot(1,3,3)
spy(abs(U)>1e-6)
title("U")
```

(d) Compute the naive sparse LU factorization of an arrow matrix with the same structure of the matrix A in Example 2.9. Compare with the standard LU implementation of MATLAB. How does the naive algorithm perform?

(e) (Optional) Notice, in the current implementation of `sparse_lu`, we do not provide the permutation matrix P. Modify the function in a way that P is now given as output.

Problem 2.12.

(a) Describe the concept of *level of fill*, what is it? How is it connected to the graph of a matrix?

(b) Plot the LU-factorization of the five-point finite difference stencil

```
n=10; e = ones(n,1);
A = spdiags([e -2*e e], -1:1, n, n); I = speye(n,n);
A = kron(I,A) + kron(A,I);
```

Plot the ILU(0) factorization of it.

(c) Write a script that computes the level of fill for the elements of this matrix. Compare with the results in (b).

Problem 2.13. Consider the following MATLAB code:

```
B = rand(4); B = B+B';
B = B-diag(diag(B))+eye(4); M = kron(eye(3),B);
```

The matrix M denotes the adjacency matrix of graph with 12 nodes. If we run the Cuthill–McKee ordering on M as described in Saad [237], Algorithm 3.2, it gets stuck in an infinite loop? Why?
Hint: Structure of M.

Problem 2.14. Prove Proposition 2.18.
Hint: Use induction: What do we know about $k = 1$?.

Problem 2.15. Prove Proposition 2.20.
Hint: Any permutation matrix can be constructed from a sequence of transpositions, see [159, s. 0.9.5].

Problem 2.16. Use the notion of graph separators to prove that the Cuthill–McKee ordering gives a block-tridiagonal matrix.

Definition 2.28 (Graph separator)**.** A *graph separator* is a set of nodes, such that if these are removed the graph becomes disconnected with exactly two connected components.

Hint: The reordering gives a new graph and a new adjacency matrix. What does it mean to be a graph separator and how is it connected to the level sets?

Problem 2.17. *A more challenging PDE: is ILU a valid preconditioner?*
We consider the linear systems obtained by discretizing a certain PDE in a more challenging domain. This problem is generated with FeNICS, with the following Python script

```
from __future__ import print_function
from fenics import *
import matplotlib.pyplot as plt
from mshr import *
import scipy.io
from scipy.io import savemat
parameters['linear_algebra_backend'] = "Eigen"
parameters['reorder_dofs_serial'] = False

# Create mesh and define function space
D1 = Rectangle(Point(0, 0), Point(1.0, 1.0))
D2 = Rectangle(Point(0.1, 0.1), Point(0.2, 0.2))
D3 = Circle(Point(0.5, 0.5), 0.05)
D4 = Circle(Point(0.7, 0.7), 0.04)
D5 = Circle(Point(0.1, 0.7), 0.04)
domain = D1 - D2 - D3 - D4 - D5
mesh = generate_mesh(domain, 40)

V = FunctionSpace(mesh, 'P', 2)
```

```
# Define boundary condition
u_D = 0

def boundary(x, on_boundary):
    return on_boundary

bc = DirichletBC(V, u_D, boundary)

# Define variational problem
u = TrialFunction(V)
v = TestFunction(V)
f = Constant(-6.0)
a = dot(grad(u), grad(v))*dx
L = f*v*dx

# Compute solution
u = Function(V)
solve(a == L, u, bc)

# Plot solution and mesh (uncomment to see the plot)
#plot(u)
#plot(mesh)
#plt.show()

# Assemble matrix
A, b = assemble_system(a, L, bc);
rows,cols,vals = as_backend_type(A).data() # Get CSR data
A = as_backend_type(A).sparray()
savemat('pde_gen_mat.mat', {'A': A,'b':b.array()})
```

The script produce a matrix which can be loaded in MATLAB. In the Figure 2.3 is showed the solution, the domain and the grid.

(a) By reading the python script describe the problem we are solving and the various settings we used.

(b) By running the python script it is generated a MATLAB file called pde_gen_mat that contains the matrix A and the vector b defining the linear systems associated to this PDE. Is this linear system sparse? Will the LU factorization be sparse?

(c) Use the sparse reverse Cuthill–McKee ordering (implemented in MATLAB). See the function symrcm. How does it look this matrix after the reordering? Will the LU factorization be sparse?

(d) Solve, in MATLAB, the linear systems $Ax = b$ with the following steps:

- Perform a sparse reverse Cuthill–McKee ordering.
- Generate an ILU factorization with the following settings

```
setup.type = 'crout';
setup.milu = 'row';
setup.droptol = 0.1;
```

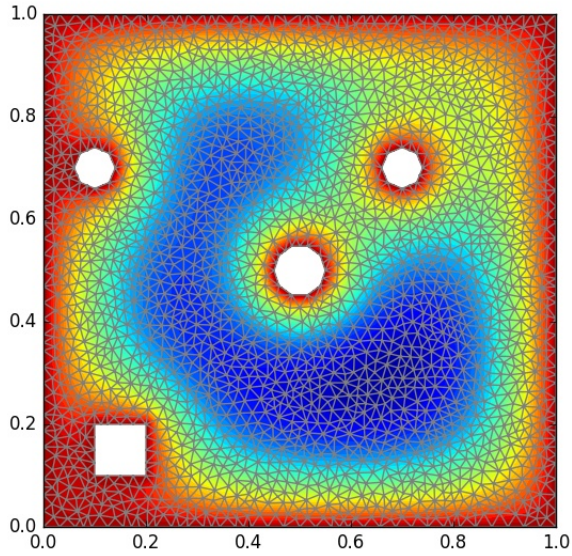


Figure 2.3: Numerical solution to the PDE described in Problem 2.17

– Use GMRES with the ILU preconditioner.

Did you menage to solve the linear system? Compare the convergence of GMRES with and without the ILU preconditioner.

Problem 2.18. Consider the matrix A with the structure

$$A = \begin{pmatrix} \times & & \times & \times & & \times \\ & \times & & \times & & \\ \times & & \times & & \times & \\ \times & \times & & \times & & \times \\ & & \times & & \times & \times \\ \times & & & \times & \times & \times \end{pmatrix}.$$

Suppose LU-factorization without pivoting is used to factorize A. Predict in which positions there will be fill-in.

Problem 2.19. Incomplete LU-factorization is often described as $A = LU - R$. Describe what L, U, and R are. How do they look for ILU(0) and what is the difference for ILU(1)?

Problem 2.20. Roughly describe modified ILU (MILU), what are the general ideas? The MILU in implemented in MATALB as variation of the ILU. What

is the difference of performing MILU versus ILU(0)? Explain this by giving an example. You may for instance consider the matrix A generated by the code below but you are very welcome to use a matrix with a structure that you find more interesting.

```
n=4; e = ones(n,1); A = spdiags([e -2*e e], -1:1, n, n);
I = speye(n,n); A = kron(I,A) + kron(A,I);
```

Chapter 3

Preconditioning for Helmholtz equation

The Helmholtz equation is a fundamental equation in physics and engineering, since it characterizes the propagation of waves, e.g., acoustic waves. This chapter summarizes preconditioners specifically designed for matrices stemming from the discretization of the Helmholtz equation. In particular we describe the techniques based on shifting the problem, which can lead to a damped variation of the equation still approximating important properties such that it can be used as a preconditioner. We also describe sweeping preconditioners and absorbing boundary conditions.

Literature

✎ Shifted Laplacian:

- Overview: damping, eigenvalues, etc [123]
- Shifted Laplacian for FEM discretization: [134]
- Two grid preconditioning with eigenvectors: [204]

✎ Sweeping preconditioner:

- LDL^T factorization: section 4.1 of [140]
- Perfectly Matched Layers (PML) and Absorbing Boundary Conditions (ABC): [228] [168]
- Sweeping Preconditioner and hierarchical matrices: [122]

Further reading

Hierarchical matrices relevant for Helmholtz equation are presented in an extended way in the books [29] [153]. Software is also available, e.g., the C library HLib. An analysis of the weak formulation of the Helmholtz equation is presented in [202]. H-matrices are discussed also in [16, 264]. More preconditioners for the Helmholtz equation are presented in [59, 193, 194].

3.1 Damping and oscillations

We consider the following instance of the Helmholtz equation

$$\begin{cases} \Delta u + \kappa^2(1+\varepsilon i)u = f \\ u\big|_{\partial\Omega} = g \end{cases} \tag{3.1}$$

where $\Omega \subseteq \mathbb{C}^d$ is a compact set and $\kappa \in \mathbb{C}$. The equation (3.1) is used to model the propagation of waves through a medium. The constant κ, which is referred to as *wavenumber*, and the constant ε depend on the material properties of the medium. For reason which will be clear in the next section, this equation is difficult to solve when κ is large and ε is small (or zero). We will now review some of the properties of the solution of equation (3.1). In order to simplify the exposition, some of the properties are illustrated for mono-dimensional examples.

Oscillation of the solutions

The larger is κ, the more the solution to (3.1) oscillates. We show this feature in the homogeneous mono-dimensional case, in particular we set $\Omega = [0,1] \subset \mathbb{R}$ and $f = 0$. The equation (3.1) becomes

$$\begin{cases} u''(x) + \kappa^2(1+\varepsilon i)u(x) = 0 \\ u(0) = u_0 \\ u(1) = u_1 \end{cases} \tag{3.2}$$

The solution to this problem is

$$u(x) = Ae^{i\kappa\sqrt{1+\varepsilon i}x} + Be^{-i\kappa\sqrt{1+\varepsilon i}x} \tag{3.3}$$

Let us now assume that $\varepsilon = 0$; then we can express

$$u(x) = A\sin(\kappa x) + B\cos(\kappa x) \tag{3.4}$$

The values of the constants A and B are obtained by imposing the boundary conditions. In Figure 3.1 is illustrated the typical solution to (3.2) for $\varepsilon = 0$.

Damping of the solution to the Helmholtz equation with complex wavenumber

We now use the concept of *damping*. We say that an oscillating function $f(x)$ is damping, if the amplitude of the oscillations $f(x)$ tends to reduce. When the wavenumber κ is complex, the solution to the Helmholtz equation is damping in

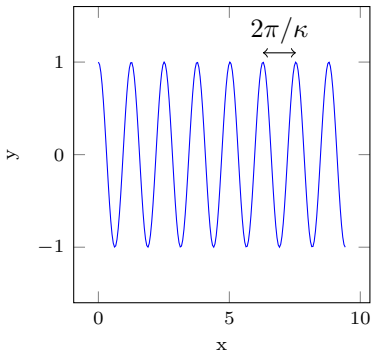


Figure 3.1: Solution 1D Helmholtz equation (periodic)

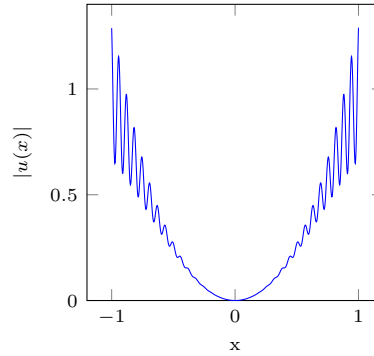


Figure 3.2: Solution 1D Helmholtz equation (damping)

certain regions of the domain. We illustrated this with in the mono-dimensional case. Let us consider

$$\begin{cases} u''(x) + \kappa^2(1+\varepsilon i)u(x) = 0 \\ u(0) = 1 \\ \lim\limits_{x\to\infty} u(x) = 0. \end{cases} \tag{3.5}$$

The solution to (3.5) is damping at infinity, namely the amplitude of the oscillations reduces going to infinity, see Problem 3.2 for more details. In general, if Ω is a compact domain, the solution is damping in the middle of the domain. See Figure 3.2 where the domain of is $\Omega = [-1, 1]$ and the boundary values are equal to one.

Methods and discretization techniques used for solving the Helmholtz equation should be based on the known features of the solution which we discussed above. An overview of these aspects is given below.

- If κ is large and ε small, the solution oscillates with frequency $2\pi/\kappa$. A numerical method based on domain discretization has to be able to capture these oscillations. For example, in a finite difference approach, the discretization step has to fulfill $h < 2\pi/\kappa$, which is very small if κ is large.
- If ε is complex, the solution is more regular thanks to the damping effect. In particular, the solution is almost constant in the middle of the domain. There are methods that can exploit such regularity, e.g. multigrid methods. In particular, the larger is ε, the easier is to solve the problem.
- The larger is κ the higher is the frequency of the oscillations, the larger is ε the faster is the damping. Therefore, a favorable situation is κ small and ε large.

3.2 The two grids method

We now present an approach inspired by the ideas discussed in [204]. We consider the Helmholtz equation in one dimension

$$\begin{cases} u''(x) + \kappa^2(1+\varepsilon i)u(x) = f(x) \\ u(0) = A \\ u(1) = B \end{cases} \tag{3.6}$$

We approximate the solution to this equation by the finite difference method. Let $0 = x_1 < x_2 < \cdots < x_N = 1$ be an uniform (fine) grid such that $x_{i+1} - x_i = h$ for $i = 1, \ldots, N-1$. We use the finite difference discretization of the second derivative given by

$$u''(x_j) \approx \frac{u(x_{j+1}) - 2u(x_j) + u(x_{j-1})}{h^2}.$$

The finite difference discretization results in the linear system $[D+\kappa^2(1+\varepsilon i)E]u = f$ where

$$D := \begin{pmatrix} 1 & & & & & & \\ -1/h^2 & 2/h^2 & -1/h^2 & & & & \\ & -1/h^2 & 2/h^2 & -1/h^2 & & & \\ & & \ddots & \ddots & \ddots & & \\ & & & -1/h^2 & 2/h^2 & -1/h^2 & \\ & & & & & 1 \end{pmatrix}, \quad f := \begin{pmatrix} A \\ f(x_2) \\ f(x_3) \\ \vdots \\ f(x_{N-1}) \\ B \end{pmatrix},$$

$$E := \begin{pmatrix} 0 & & & & & \\ & 1 & & & & \\ & & 1 & & & \\ & & & \ddots & & \\ & & & & 1 & \\ & & & & & 0 \end{pmatrix}, \qquad u := \begin{pmatrix} u_1 \\ u_2 \\ u_3 \\ \vdots \\ u_{N-1} \\ u_N \end{pmatrix}.$$

Let us denote by $Au = f$ such linear system. The idea behind the two grids method is to consider a coarse grid for approximating the solution. In particular, let us consider the coarse grid

$$0 = \tilde{x}_1 < \tilde{x}_2 < \cdots < \tilde{x}_{\tilde{N}} = 1 \qquad \text{with} \qquad \tilde{h} = \tilde{x}_{i+1} - \tilde{x}_i$$

where $\tilde{N} < N$. Let $\tilde{A}\tilde{u} = \tilde{f}$ be the corresponding linear system. Then u, solution of the original linear system, can be approximated by interpolating $\tilde{u}$ in the fine grid. Clearly, we obtain a good approximation only if the solution to (3.6) does not oscillate too much, namely κ is not large. This approach is then used as preconditioner. More precisely, given the linear system $Au = b$, then we consider the preconditioner which consists of the following procedure

- compute $\tilde{b}$ by interpolating b on the coarse grid (implicit assumption $b_j \approx b(x_j)$),
- solve $\tilde{A}\tilde{u} = \tilde{b}$,
- compute an approximation to u by interpolating $\tilde{u}$ on the fine grid.

3.3 Shifted-Laplacian preconditioner

We now consider the Helmholtz equation with real wavenumber

$$\begin{cases} \Delta u + \kappa^2 u = f \\ u\big|_{\partial\Omega} = g \end{cases} \tag{3.7}$$

as preconditioner we use

$$\begin{cases} \Delta u + \kappa^2 u(1 + \varepsilon i) = f \\ u\big|_{\partial\Omega} = g \end{cases}$$

for $0 < \varepsilon < 1$. This preconditioner has two main features:

- The larger is ε the faster is the computation of the action of preconditioner.
- The smaller is ε the better is the approximation of the solution provided by the preconditioner.

In practice, small values of ε lead to a better preconditioner which will let the method we are using, e.g., GMRES, converge faster. On the other hand, small values of ε makes the process of computing the action of the preconditioner more expensive. Therefore, the value of ε is a trade-off between these two scenarios. The quality of this preconditioner is described by the following theorem. For a more formal formulation see [134, Theorem 1.4].

Theorem 3.1. *Under the technical hypothesis in [134, Theorem 1.4], we consider the discretized problem obtained with FEM by using elements that have radius such that $h\kappa < 1$. Let A the matrix associated with the linear system and let A_ε the matrix of the preconditioner, then*

$$\|I - A_\varepsilon^{-1} A\|_2 \leq C\varepsilon \tag{3.8}$$

where C is a problem-dependent constant, which depends on Ω and κ.

Note that the hypothesis $h\kappa < 1$ guarantees that all the oscillations of the solutions are resolved. In the mono-dimensional case we analyzed before, with a finite difference discretization, we obtained $h\kappa < 2\pi$.

The proof of the Theorem 3.1 is based on FEM arguments. Rather then proving this result, which is very general, we verify that similar results can be obtained in a simplified case with FD. We consider the mono-dimensional problem,

$$\begin{cases} u''(x) + \kappa u(x) = f(x) \\ u(0) = 0 \\ u(1) = 0 \end{cases}$$

and let $Au = f$ be the linear system obtained by a FD discretization. The eigenvalues have a closed expression given in [123, ss. 3.4–3.5], which is

$$\lambda_k = \kappa^2 - \frac{4}{h^2} \sin\left(\frac{j\pi}{2(n+1)}\right) \qquad j = 1, \ldots, n \tag{3.9}$$

whereas for the preconditioned system

$$\lambda_k^\varepsilon = \kappa^2(1+\varepsilon i) - \frac{4}{h^2} \sin\left(\frac{j\pi}{2(n+1)}\right) \qquad j = 1, \ldots, n \tag{3.10}$$

We can now study the spectrum for this simplified problem in order to understand the basic concepts. In particular we get

$$\|I - A_\varepsilon^{-1} A\|_2 \approx 1 - \lambda_{\max}\left(A_\varepsilon^{-1} A\right). \tag{3.11}$$

and by using (3.9) and (3.10) we get, for $\varepsilon < 1$, the following first order approximation

$$\frac{\lambda_j}{\lambda_j^\varepsilon} \approx \frac{\kappa^2}{\kappa^2(1+\varepsilon i)} = 1 - \varepsilon + \mathcal{O}(\varepsilon^2) \tag{3.12}$$

Observation 3.2. *If we choose $\varepsilon = 1$, the eigenvalues of $A_\varepsilon^{-1} A$ are in the circle centered in $1/2$ with radius $1/2$. This means that GMRES converges slowly, see Corollary 1.4.*

3.4 Perfectly Matched Layers

In this section we consider the Helmholtz equation (3.13) in the unbounded domain $\mathbb{R}^d$, $d = 2, 3$, with source f, and radiation condition, also called *Sommerfeld condition*, given by

$$\Delta u(x) + \frac{\omega^2}{c^2(x)} u(x) = f(x) \quad \text{in } \mathbb{R}^d \tag{3.13}$$

$$\lim_{|x|\to\infty} |x|^{\frac{d-1}{2}} \left(\frac{\partial}{\partial |x|} - i\omega\right) u = 0\,. \tag{3.14}$$

The Sommerfeld condition (3.14) implies that the wave propagates from the source outwards. This condition is sufficient in order to have uniqueness of the solution. We assume that $\text{supp}(f) \subset D := [0,1]^d$, i.e., the source is limited to the unit square D, and we are only interested in the solution in this domain. A way to transform (3.13) in an equivalent PDE with bounded domain is by replacing the boundary conditions (3.14) at infinity with *absorbing boundary conditions* (ABCs) in the boundary of D. This can be done with the method of perfectly matched layers (PML) which we briefly recall.

Computational framework of PML

The idea of PML consists of performing a change of variables in a way such that, with respect to the new variables, the solution to (3.13) decays exponentially to zero for $|x| \to \infty$. Dirichlet homogeneous boundary conditions can therefore efficiently be used to set the problem with respect the new variables. We present the PML for a bi-dimensional problem, i.e., $d = 2$ in (3.13).
We denote by (z_1, z_2) the new variables, where $z_j := x_j + ig(x_j)$ for $j = 1, 2$ and where g is a given function specified later. Observe that we extended the field of the variables to the complex numbers. In order to perform the change of variable in (3.13) we observe that

$$\frac{\partial u}{\partial z_j} = \left(1 + i\frac{\mathrm{d}g}{\mathrm{d}x_j}\right)^{-1} \frac{\partial u}{\partial x_j} \tag{3.15}$$

Motivated by this expression, the function g is typically selected such that

$$\frac{\mathrm{d}g}{\mathrm{d}x} = \frac{\sigma(x)}{\omega},$$

where $\sigma(x)$ is a function which is zero in the domain of interest for (3.13), and positive in a small band until the boundary. One way to select this function is the following

$$\sigma(t) = \begin{cases} \frac{C}{\eta}\left(\frac{t-\eta}{\eta}\right)^2 & 0 \le t \le \eta \\ \frac{C}{\eta}\left(\frac{t-1+\eta}{\eta}\right)^2 & 1-\eta \le t \le 1 \\ 0 & \text{otherwise} \end{cases}$$

where η is a small number, e.g. 10^{-6}, which is application dependent. With this reasoning, equation (3.15) can be written as

$$\frac{\partial u}{\partial z_j} = \left(1 + i\frac{\sigma(x_j)}{\omega}\right)^{-1} \frac{\partial u}{\partial x} = s(x_j)\frac{\partial u}{\partial x_j}$$

where $s(x_j)$ is defined accordingly. We can therefore write (3.13), with respect to the new variables as

$$\left[\left(s_1\frac{\partial}{\partial z_1}\right)\left(s_1\frac{\partial}{\partial z_1}\right)u(z)+\left(s_2\frac{\partial}{\partial z_2}\right)\left(s_2\frac{\partial}{\partial z_2}\right)u(z)\right]+\frac{\omega^2}{c^2(x)}u(z)=f(z) \tag{3.16}$$

It is possible to show that the solution to (3.16) decays exponentially to zero for z outsize of $\mathrm{supp}(f)$. Therefore, in order to have a well defined problem which can be numerically solved, we impose homogeneous Dirichlet boundary conditions on (3.16). In conclusion, the PML gives the following problem

$$\left[\left(s_1\frac{\partial}{\partial z_1}\right)\left(s_1\frac{\partial}{\partial z_1}\right)u(z)+\left(s_2\frac{\partial}{\partial z_2}\right)\left(s_2\frac{\partial}{\partial z_2}\right)u(z)\right]+\frac{\omega^2}{c^2(x)}u(z)=f(z) \tag{3.17}$$

$$u(z)=0 \quad \text{for} \quad z\in\partial D \tag{3.18}$$

Remark 3.3. *The solution to* (3.18) *is a an approximation to the solution to the original problem* (3.13) *with boundary conditions* (3.14) *in the set* $\mathrm{supp}(f)$. *This is justified by two main facts:*

- *the solution to* (3.16) *decays to zero exponentially outside of* $\mathrm{supp}(f)$,
- $g(x)=0$ *for* $x\in\mathrm{supp}(f)$, *namely the new variables correspond with the old ones in the domain of interest.*

In the rest of this section we will only consider the PML reformulation (3.18) but we will denote the variables with x (and not z which was used in the derivation). We will also use the following nomenclature: we will call *layers* of the PML the regions where $\{x\in D\wedge\sigma(x)>0\}$; we will also assume that $\mathrm{supp}\, f(x)$ does not intersect such regions.

3.5 Sweeping factorization

Symmetrization and discretization

The equation (3.16) is not symmetric, but we can make it symmetric by dividing by $s_1(x)s_2(x)$

$$\left(\frac{\partial}{\partial x_1}\left(\frac{s_1}{s_2}\frac{\partial}{\partial x_1}\right)u(x)+\frac{\partial}{\partial x_2}\left(\frac{s_2}{s_1}\frac{\partial}{\partial x_2}\right)u(x)\right)+\frac{\omega^2}{s_1(x)s_2(x)c^2(x)}u(x)=f(x) \tag{3.19}$$

We now discretize this equation with finite differences. More precisely, by using the five point stencil for the second derivative, we get

$$\begin{aligned}
&\frac{1}{h^2}\left(\frac{s_1}{s_2}\right)_{i+\frac{1}{2},j} u_{i+1,j} + \frac{1}{h^2}\left(\frac{s_2}{s_1}\right)_{i,j+\frac{1}{2}} u_{i,j+1} + \\
&\frac{1}{h^2}\left(\frac{s_1}{s_2}\right)_{i-\frac{1}{2},j} u_{i-1,j} + \frac{1}{h^2}\left(\frac{s_2}{s_1}\right)_{i,j-\frac{1}{2}} u_{i,j-1} - \\
\Bigg[&\frac{1}{h^2}\left(\frac{s_1}{s_2}\right)_{i+\frac{1}{2},j} + \frac{1}{h^2}\left(\frac{s_1}{s_2}\right)_{i-\frac{1}{2},j} + \frac{1}{h^2}\left(\frac{s_2}{s_1}\right)_{i,j+\frac{1}{2}} + \\
&\frac{1}{h^2}\left(\frac{s_2}{s_1}\right)_{i,j-\frac{1}{2}} - \frac{\omega^2}{(s_1 s_2)_{i,j} c_{i,j}^2}\Bigg] u_{i,j} = f_{i,j}
\end{aligned}$$

The discretization points are $n+2$ in each direction, by taking $h = \frac{1}{n+1}$, but we only consider the n internal ones, which consequently have coordinates (ih, jh), $i, j = 1, \ldots, n$. We order the unknowns $u_{i,j}$, and also the known function f, by rows, from bottom to top:

$$\tilde{u}_m = (u_{1,m}, u_{2,m}, \ldots, u_{n,m})^T \ , \quad \tilde{f}_m = (f_{1,m}, f_{2,m}, \ldots, f_{n,m})^T$$

$$u = \begin{pmatrix} \tilde{u}_1 \\ \tilde{u}_2 \\ \vdots \\ \tilde{u}_n \end{pmatrix} \ , \qquad f = \begin{pmatrix} \tilde{f}_1 \\ \tilde{f}_2 \\ \vdots \\ \tilde{f}_n \end{pmatrix}$$

and we get the system

$$Au = f$$

where

$$A = \begin{pmatrix} A_{1,1} & A_{1,2} & & \\ A_{2,1} & A_{2,2} & \ddots & \\ & \ddots & \ddots & A_{n-1,n} \\ & & A_{n,n-1} & A_{n,n} \end{pmatrix} \tag{3.20}$$

and $A_{m,m}$ is tridiagonal and $A_{m,m-1} = A_{m-1,m}^T$ is diagonal.

Matrix factorizations

The sweeping factorization is a block LDL^T factorization, and to understand what it means let's take a step back.

Theorem 3.4 ([140, Theorem 4.1.3]). *If $A \in \mathbb{R}^{n\times n}$ is symmetric and the principal submatrix $A(1:k,1:k)$ is nonsingular for $k = 1:n-1$, then there exists a unit lower triangular matrix L and a diagonal matrix $D := diag(d_1, d_2, \ldots, d_n)$ such that $A = LDL^T$, and the factorization is unique.*

Proof. From the LU-factorization $A = LU$, the matrix $L^{-1}AL^{-T} = UL^{-T}$ is symmetric and upper triangular, so it must be diagonal. Then by setting $D = UL^{-T}$ the factorization is proved, and the uniqueness comes from the uniqueness of the LU factorization. □

Definition 3.5 (Schur complement). Given a $(p+q)\times(p+q)$ block matrix

$$M = \begin{pmatrix} A & B \\ C & D \end{pmatrix}$$

Suppose D invertible; then the Schur complement of the block D of the matrix M is the $p \times p$ matrix

$$M/D := A - BD^{-1}C$$

Suppose A invertible; then the Schur complement of the block A of the matrix M is the $q \times q$ matrix

$$M/A := D - CA^{-1}B$$

Definition 3.6 (Block LDL^T decomposition). If A, B and D are symmetric,

$$\begin{aligned}\begin{pmatrix} A & B \\ B & D \end{pmatrix} &= \begin{pmatrix} I & 0 \\ BA^{-1} & I \end{pmatrix}\begin{pmatrix} A & 0 \\ 0 & D - BA^{-1}B \end{pmatrix}\begin{pmatrix} I & (BA^{-1})^T \\ 0 & I \end{pmatrix} \\ &= \begin{pmatrix} I & 0 \\ BA^{-1} & I \end{pmatrix}\begin{pmatrix} A & 0 \\ 0 & D - BA^{-1}B \end{pmatrix}\begin{pmatrix} I & 0 \\ BA^{-1} & I \end{pmatrix}^T\end{aligned}$$

Let's notice that $D - BA^{-1}B$ is the Schur complement of the block A.

Sweeping factorization

We now consider a specific block LDL^T factorization which we refer to as *sweeping factorization*. This factorization is defined algorithmically. We start from the first row block row

$$A = L_1 \begin{pmatrix} S_1 & 0 & & \\ 0 & S_2 & A_{2,3} & \\ & A_{3,2} & \ddots & \ddots \\ & & \ddots & \ddots \end{pmatrix} L_1^T$$

where $S_1 = A_{1,1}$ and $S_2 = A_{2,2} - A_{2,1}S_1^{-1}A_{1,2}$ is the Schur complement of the first block $A_{1,1} = S_1$, and

$$L_1 = \begin{pmatrix} I & 0 & & \\ A_{2,1}S_1^{-1} & I & 0 & \\ & 0 & I & \ddots \\ & & \ddots & \ddots \end{pmatrix}$$

Repeating the process iteratively, we get

$$A = L_1 L_2 \cdots L_{n-1} \begin{pmatrix} S_1 & & & \\ & S_2 & & \\ & & \ddots & \\ & & & S_n \end{pmatrix} L_{n-1}^T L_{n-2}^T \cdots L_2^T L_1^T$$

where $S_m = A_{m,m} - A_{m,m-1}S_{m-1}^{-1}A_{m-1,m}$, $m = 2, 3, \ldots, n$, and L_m is such that the block $(m+1, m)$ is $A_{m+1}S_m^{-1}$, the diagonal blocks (k, k) are I, and the others are zeros. Observe now that the solution to the linear system defined by the matrix (3.20) after the block LDL^T factorization can be computed as

$$u = \left(L_1^T\right)^{-1} \left(L_2^T\right)^{-1} \cdots \left(L_{n-1}^T\right)^{-1} \begin{pmatrix} S_1^{-1} & & & \\ & S_2^{-1} & & \\ & & \ddots & \\ & & & S_n^{-1} \end{pmatrix} L_{n-1}^{-1} L_{n-2}^{-1} \cdots L_1^{-1} f$$

In Algorithm 6 is summarized the procedure for computing the sweeping factorization whereas in Algorithm 7 is summarized the procedure for solving the linear system given the sweeping factorization.

The complexity for computing the sweeping factorization is $\mathcal{O}(n^4) = \mathcal{O}(N^2)$ and the complexity for solving the linear system (given the sweeping factorization) is $\mathcal{O}(n^3) = \mathcal{O}(N^{\frac{3}{2}})$.

By following this reasoning, a first approach for deriving a preconditioner is the following. We approximate the sweeping factorization $A = LDL^T$ by approximating the Schur complements S_m by $\tilde{S}_m$ in a way that the computation of $\tilde{S}_m$ has lower complexity (see Problem 3.19). Let $A \approx \tilde{L}\tilde{D}\tilde{L}^T$ be the approximate sweeping factorization, we consider the preconditioner $P = \tilde{L}^{-1}\tilde{D}^{-1}\tilde{L}^{-T}$. The action of the preconditioner can be efficiently computed by using Algorithm 7. The quality of the preconditioner depends on how close the approximated sweeping factorization is to the exact sweeping factorization.

Decay of the rank of off-diagonal blocks

The diagonal blocks $A_{1,1}$ to $A_{m,m}$ of the matrix (3.20) are the discretization of Helmholtz operator, with zero boundary conditions, with respect to the variable

Algorithm 6: Construction of the sweeping factorization (without hierarchical matrix representation)

input : A matrix $A \in \mathbb{R}^{n^2 \times n^2} = \mathbb{R}^{N \times N}$
output : Block-diagonal LDL^T factorization components S_m and their inverse T_m

$S_1 = A_{1,1}$
$T_1 = S_1^{-1}$
for $m = 2, \ldots, n$ **do**
  $S_m = A_{m,m} - A_{m,m-1} T_{m-1} A_{m-1,m}$
  $T_m = S_m^{-1}$
end
return S_m and T_m for $m = 1, \ldots, n$

x_2. The inverse of the matrix A corresponds to the discretization of the Green's function of the Helmholtz operator with zero boundary conditions. This matrix has a very special property which is described by the following theorem.

Theorem 3.7. *Let $G(x, y)$ be the Green's function of the Helmholtz operator with zero boundary conditions. Let*

$$I = (i_1, \ldots, i_k) \qquad J = (j_1, \ldots, j_k) \qquad \text{such that}$$
$$i_s, j_s \in \{1, \ldots, n\} \text{ for } s = 1, \ldots, k \qquad \text{and } I \cap J = \emptyset$$

Then the matrix $\tilde{G} = \{G(x_{i_t}, y_{j_s})\}_{t,s=1}^{k}$ can be approximated by a low rank matrix. More precisely, $\forall \varepsilon > 0 \;\; \exists R \in \mathbb{N}, \;\; V, W \in \mathbb{R}^{k \times R}$ such that $\|G - VW^T\| < \varepsilon$.

Let $G = \{G(x_i, y_j)\}_{i,j=1}^{n}$ be the matrix obtained by discretizing the Green's function, then the above theorem states that the submatrices of G that do not cross the diagonal can be approximated by low rank matrices.

A direct implication is that, even if the matrices $T_m = S_m^{-1}$ are dense, the off-diagonal blocks have numerically low rank (see Problem 3.16) and can be approximated in the hierarchical matrix framework as discussed in the next section.

Sweeping factorization in the Hierarchical matrix framework

Hierarchical matrices are used as data-sparse approximations/representation of non-sparse matrices. A matrix is hierarchical if all the submatrices that do not contain elements from the diagonal are low rank. These matrices can be efficiently stored, namely it is not need to store all the entries of the matrix but only the low-rank terms. There are several different ways to store hierarchical matrices. Operations

Algorithm 7: Computation of $u = A^{-1}f$ using the sweeping factorization (without hierarchical matrix representation)

input : Sweeping factorization of A and right hand side f
output : Solution $u = A^{-1}f$
for $m = 1, \ldots, n$ **do**
1 $\quad u_m = f_m$
end
for $m = 1, \ldots, n-1$ **do**
2 $\quad u_{m+1} = u_{m+1} - A_{m+1,m}(T_m u_m)$
end
for $m = 1, \ldots, n$ **do**
3 $\quad u_m = T_m u_m$
end
for $m = n-1, \ldots, 1$ **do**
4 $\quad u_m = u_m - T_m(A_{m,m+1} u_{m+1})$
end
5 **return** u_m for $m = 1, \ldots, n$

between hierarchical matrices, such as matrix-matrix addition/subtraction, matrix-vector operations etc., can be efficiently performed.

The final computational complexity of the preconditioner calculation becomes, in the framework of hierarchical matrices, $\mathcal{O}(R^2 N \log N)$ for the building and $\mathcal{O}(RN \log N)$ for the computation. This is much less expensive than the direct methods when R is small, both in terms of time and storage.

Let $\tilde{S}_m$ and $\tilde{T}_m$ be approximations of S_m and T_m in the hierachical matrix representation framework. The modified algorithms to construct the sweeping factorization and construct the solution u are shown in the following paragraph. The special operations used in the framework are the following: `hadd` and `hsub` are addition and subtraction respectively, their complexity is $\mathcal{O}(R^2 n \log n)$; `hmatvec` is matrix-vector multiplication, $\mathcal{O}(Rn \log n)$; `hmul` is matrix multiplication, $O(R^2 n \log^2 n)$; `hdiagmul` is multiplication between a matrix and a block diagonal matrix, $\mathcal{O}(Rn \log n)$; `hinv` is matrix inversion, $\mathcal{O}(R^2 n \log^2 n)$.

The cost of algorithm 8 is $\mathcal{O}(R^2 n^2 \log^2 n) = \mathcal{O}(R^2 N \log^2 N)$, and the cost of the computation of u, algorithm 9, is $\mathcal{O}(Rn^2 \log n) = \mathcal{O}(RN \log N)$. Algorithm 9 defines an operator which approximates the inverse of the discrete Helmholtz operator A; even if its approximation is not accurate, it can be used as a preconditioner, especially when considering R small, when the cost is much smaller compared to direct methods.

Algorithm 8: Construction of the sweeping factorization (with hierarchical matrix representation)

input : A matrix $A \in \mathbb{R}^{n^2 \times n^2} = \mathbb{R}^{N \times N}$
output : Approximation in the hierarchical representation of block-diagonal LDL^T factorization components, $\tilde{S}_m$ and their inverse $\tilde{T}_m$

$\tilde{S}_1 = A_{1,1}$
$\tilde{T}_1 = \texttt{hinv}(\tilde{S}_1)$
for $m = 2, \ldots, n$ **do**
  $\tilde{S}_m = \texttt{hsub}(A_{m,m}, \texttt{hdiagmul}(A_{m,m-1}, \texttt{hdiagmul}(\tilde{T}_{m-1}, A_{m-1,m})))$
  $\tilde{T}_m = \texttt{hinv}(\tilde{S}_m)$
end
return $\tilde{S}_m$ and $\tilde{T}_m$ for $m = 1, \ldots, n$

Algorithm 9: Computation of $u \approx A^{-1} f$ using the sweeping factorization (with hierarchical matrix representation)

input : Sweeping factorization of A and right hand side f
output : Solution $u = A^{-1} f$
for $m = 1, \ldots, n$ **do**
  $u_m = f_m$
end
for $m = 1, \ldots, n-1$ **do**
  $u_{m+1} = u_{m+1} - A_{m+1,m} \texttt{hmatvec}(\tilde{T}_m, u_m)$
end
for $m = 1, \ldots, n$ **do**
  $u_m = \texttt{hmatvec}(\tilde{T}_m, u_m)$
end
for $m = n-1, \ldots, 1$ **do**
  $u_m = u_m - \texttt{hmatvec}(\tilde{T}_m, A_{m,m+1} u_{m+1})$
end
return u_m for $m = 1, \ldots, n$

3.6 Problems

Problem 3.1. *Properties of the solution of the 1d-Helmholtz equation*
Consider the mono-dimensional homogeneous Helmholtz equation

$$\begin{cases} u''(x) + \kappa^2 u(x) = 0 \\ u(0) = u_0 \\ u(1) = u_1 \end{cases} \tag{3.21}$$

where $\kappa \in \mathbb{R}$.

(a) Compute the analytic solutions and show that they are oscillating functions. More precisely the larger is κ the more the solution oscillates.

(b) Solve this problem in MATLAB and illustrate with plots for different values of κ your argument in (a).

(c) What happens if $\kappa \in \mathbb{C}$? What happens if κ is purely imaginary?

Problem 3.2. *Damping effect*
Consider the mono-dimensional Helmholtz equation

$$\begin{cases} u''(x) + \kappa^2 (1 + \varepsilon i)^2 u(x) = 0 \\ u(0) = 1 \\ \lim\limits_{x \to \infty} u(x) = 0 \end{cases}$$

where $\kappa \in \mathbb{R}$.

(a) Show that for $\varepsilon = 0$ the solution is an oscillatory function such that the larger is κ the more the function oscillates.

(b) Show that for $\varepsilon > 0$ the solution is damping. More precisely show that the solution can be written as $u(x) = f(x)g_\varepsilon(x)$ where $f(x)$ is a periodic function and $\lim\limits_{x \to \infty} g_\varepsilon(x) = 0$.

(c) Show that the larger is ε the faster is the damping. More precisely show that if $\varepsilon_1 < \varepsilon_2$ then for a fixed x in the domain we have $|u_{\varepsilon_1}(x)| < |u_{\varepsilon_2}(x)|$.

Problem 3.3. *Two grid preconditioner*
Consider the mono-dimensional problem

$$\begin{cases} u''(x) + \kappa^2 (1 + \varepsilon i) u(x) = 0 \\ u(0) = 10 \\ u(1) = 10 \end{cases} \tag{3.22}$$

By discretizing the problem with FD with n discretization points, we obtain the linear systems $A_n u_n = b_n$. We want to solve this linear system with GMRES combined with the preconditioner that is described by the following procedure:

- Discretize the problem by using half of the discretization points
- Solve $A_{n/2}u_{n/2} = b_{n/2}$
- Approximate u_n by interpolating $u_{n/2}$

This procedure is implemented in the following script

```
n=10000; xx=linspace(0,1,n); h=xx(2)-xx(1);
e = ones(n,1);  I=speye(n);
kk=100000;  ee=0.5;
kk=kk+ee*kk*1i;
D = spdiags([e -2*e e], -1:1, n, n);
A = D./(h^2)+kk*I;
A(1,:)=0;        A(1,1)=1;
A(end,:)=0;      A(end,end)=1;
b=ones(n,1);     b(1)=10; b(end)=10;

P=@(u) precond(u,n/2,xx,kk); TOL=1e-12; MAXIT=100;
[X,FLAG,RELRES,ITER,RESVEC]=gmres(A,b,[],TOL,MAXIT,P);

semilogy(RESVEC./n)

function uu2=precond(b,n,xx2,kk)
    xx1=linspace(0,1,n); h=xx1(2)-xx1(1);
    e = ones(n,1); I=speye(n);
    D = spdiags([e -2*e e], -1:1, n, n);
    A = D./(h^2)+kk*I;
    A(1,:)=0;        A(1,1)=1;
    A(end,:)=0;      A(end,end)=1;
    b = interp1(xx2,b,xx1).';
    uu=A\b;
    uu2 = interp1(xx1,uu,xx2).';
end
```

(a) Run the script and generate plots for different values of the parameter `kk` in the script (that corresponds to κ^2) with $\varepsilon = 1/2$. How does the convergence of GMRES depend on the magnitude of `kk`?

(b) Analyze the results obtained in the previous task with the theory presented in this chapter. Why does this preconditioner work?

(c) Try to use different values for `ee` in the script (that corresponds to ε). When does the preconditioner work better? Why?

Problem 3.4. Solve in MATLAB (or FEniCS) the following Helmholtz equation

$$\begin{aligned} \Delta u(x,y) + (1+\varepsilon i)\kappa^2 u(x,y) &= 1 && (x,y) \in [0,1]^2 \\ u(x,y) &= 0 && (x,y) \in \partial[0,1]^2 \end{aligned}$$

(a) Fix $\varepsilon = 0$ and plot the solution for different values of κ. How do the oscillations of $u(x,y)$ depend on κ?

(b) Fix κ large enough and plot the solution for different values of $0 < \varepsilon < 1$. Is ε introducing a damping? Notice that the solution is a complex function. You can plot in in MATLAB as `surf(abs(U))`.

(c) Analyze the results and relate them to the mono-dimensional case.

Problem 3.5. The following Python script uses FEniCS to discretize the Helmholtz equation.

```
from __future__ import print_function
from FEniCS import *
from scipy.io import savemat
import matplotlib.pyplot as plt

parameters['linear_algebra_backend'] = "Eigen"
parameters['reorder_dofs_serial'] = False

# Create mesh and define function space
mesh = UnitSquareMesh(40, 40)
V = FunctionSpace(mesh, 'P', 1)

# Define boundary condition
u_D = Expression('1', degree=1)

def boundary(x, on_boundary):
    return on_boundary

bc = DirichletBC(V, u_D, boundary)

# Define variational problem
k = 20;
u = TrialFunction(V)
v = TestFunction(V)
f = Constant(-6.0)
a = -dot(grad(u), grad(v))*dx+k**2*dot(u,v)*dx
L = f*v*dx

# Compute solution
u = Function(V)
solve(a == L, u, bc)

# Uncomment the following code to plot solution and the mesh
#plot(u)
#plot(mesh)
#plt.show()

# Assemble matrix
A, b = assemble_system(a, L, bc);
rows,cols,vals = as_backend_type(A).data() # Get CSR data
A = as_backend_type(A).sparray()
b = as_backend_type(b).get_local()
savemat('Helmholtz.mat', {'A': A,'b':b})
```

(a) Which PDE is this script discretizing? Specify domain, right-hand side, boundary conditions, etc.

(b) Run the script. A MATLAB file will be generated (matrix A and vector u). Solve the linear systems in MATLAB and plot the solution. You need to reshape the solution.

Problem 3.6. *Helmholtz equation and Galerkin optimality conditions.*
Scheme the article [202] and try to understand why, and in which sense, the Galerkin method for the Helmholtz equation, with real wavenumber, is not optimal. Does the situation change if the wavenumber is complex? You can follow the lead of the following questions in order to find key words in the paper.

(a) What is the weak formulation of the Helmholtz equation?

(b) What does it means that a bilinear form $a(u, v)$ is coercive? Is the bilinear form involved in the Helmholtz equation coercive?

(c) What does it mean that a Galerkin method is quasi-optimal? Under which conditions a Galerkin method (like FEM) is quasi-optimal?

(d) What is the weak formulation of the Helmholtz equation if we introduce a complex shift? Is now the $a_\varepsilon(u, v)$ coercive? Is the Galerkin method quasi-optimal?

Problem 3.7. Consider the FD discretization of the following eigenvalue problem

$$\begin{aligned} &u''(x) + k^2(1 + \varepsilon i)u(x) = \lambda u(x) \\ &u(0) = 0, u(1) = 0 \end{aligned}$$

Prove (or show with MATLAB) that for $\varepsilon = 1$, the eigenvalues lie in the circle centered in $1/2$ with radius $1/2$.

Problem 3.8. *Laplace preconditioner for the Helmholtz equation.*
Consider Problem 3.5, where the Python script, by using FEniCS, generates a FEM-discretization of a Helmholtz equation on the unit square. We consider the following preconditioner. With a regular FD, the Laplace equation with homogeneous Dirichlet boundary conditions can be written as a Lyapunov equation, i.e. $DU + UD = B$, where D is the FD matrix that discretizes the second derivative. Lyapunov equations can be efficiently solved with specialized methods, which are implemented in MATLAB in the function `lyap`. Compare the convergence of GMRES with and without preconditioner and test different values of κ (change it in the Python script).

Problem 3.9. *Is a fine mesh needed for solving with FEM the Helmholtz equation with large κ?*

Let us consider the 1D Helmholtz equation

$$u''(x) + \kappa^2 u(x) = 0, \tag{3.23}$$
$$u(0) = 1, u(1) = 0. \tag{3.24}$$

Write a Python script that solves this problem using FEniCS. Fix $\kappa = 100$ (highly oscillatory solution) but use only $n = 10$ nodes. Test different degrees for the basis function.[1]

Problem 3.10. Consider the mono-dimensional Helmholtz equation

$$\begin{cases} u''(x) + \kappa^2 u(x) = 0 \\ u(0) = u_0 \\ u(1) = u_1 \end{cases}$$

We know that the larger is $\operatorname{Re}\kappa$ the more the solution is oscillating, the larger is $\operatorname{Im}\kappa$, the faster the solution is damping. Consider the limit case $\operatorname{Re}\kappa = 0$, which is tipically not fulfilled in the applications. Discuss the features of the analytic solution of this PDE and explain which iterative solvers are efficient for solving the discretized problem. Are the classical methods, e.g. Jacobi, a good option?

Problem 3.11. The PML is applied in such a way to make the problem symmetric with respect to the variables x_1 and x_2.

(a) Does the symmetry with respect to the variables guarantee the symmetry of the discretized system?

(b) Is the symmetry of A necessary in the sweeping factorization?

Problem 3.12. Prove that the $A = LDM^T$ decomposition becomes $A = LDL^T$ if A is symmetric.

Problem 3.13. Complete the following code for the implementation of the naive sweeping factorization. Discuss the computational complexity, and explain what could be exploited to make it faster, and show and example in which this exploit would work (not necessarily in full generality, you can take further assumptions).

```
S=A(1:n,1:n);
Tm=...;
for m=2:n
    S=A((1:n)+(m-1)*n,(1:n)+(m-1)*n)-...*Tm*...;
    Tm=...;
    T((1:n)+(m-1)*n,(1:n)+(m-1)*n)=Tm;
end
```

[1] In FEM there are two paramenters: h is the size of the elements and p is the degree of the polynomials which form the basis functions. Typically one chooses a small value for h in order to catch all the oscillations of the solution. In this problem we increase p instead.

Problem 3.14. Complete the following code for the computation of the solution $Au = b$ using the sweeping factorization, i.e. assuming is available the factorization $A = LDL^T$. Then compute the complexity, and propose changes to avoid redundant computations in the loops.

```
for m=1:n
    u((1:n)+(m-1)*n)=f((1:n)+(m-1)*n);
end
for m=1:n-1
    u((1:n)+(m)*n)=u((1:n)+(m)*n)-...
end
for m=1:n
    u((1:n)+(m-1)*n)=...*u((1:n)+(m-1)*n);
end
for m=...
    u((1:n)+(...)*n)=u((1:n)+(m-1)*n)-...
        T((1:n)+(m-1)*n,(1:n)+(m-1)*n)*...
end
```

Problem 3.15. Explain the principle behind the sweeping factorization, and why is theoretically the ordering important in the successful speed-up of the algorithm.

Problem 3.16. Using the naive sweeping factorization building algorithm, see Problem 3.13, check that the off-diagonal blocks of the matrices T_m have numerically low rank.

Problem 3.17. Hierarchical matrices are the key of the sweeping preconditioning approaches. We now consider another class of matrices, often referred to as *semi-separable* matrices, and show that similar properties are fulfilled.

(a) Let us consider

$$M = \begin{pmatrix} A & uv^T \\ uv^T & B \end{pmatrix}$$

where $u, v \in \mathbb{R}^n$ and $A \in \mathbb{R}^{n\times n}$. Compute a block LDL^T factorization. Compute the complexity of the block LDL^T factorization assuming that A is sparse and solving a linear system with A has complexity $\mathcal{O}(n)$. We keep the assumption regarding the sparsity of A for all the following problems.

(b) Let consider the matrix

$$M = \begin{pmatrix} A & UV^T \\ UV^T & B \end{pmatrix}$$

where $U, V \in \mathbb{R}^{n\times k}$. Compute a block LDL^T factorization. Compute the complexity of the block LDL^T factorization.

(c) Consider the matrix

$$M = \begin{pmatrix} A & E \\ E & B \end{pmatrix}$$

where $E \in \mathbb{R}^{n \times n}$. Compute a block LDL^T factorization. Assume that the singular values of E have a fast decay, i.e. $\sigma_i(E) < 2^{-i}$. Compute an approximation $\tilde{L}\tilde{D}\tilde{L}^T$ of the block LDL^T factorization such that $\|A - \tilde{L}\tilde{D}\tilde{L}^T\| < 10^{-16}$. Derive the complexity for computing the approximation $\tilde{L}\tilde{D}\tilde{L}^T$.

Problem 3.18. Give the definition and few examples of hierarchical matrices. Explain the advantages that this structure gives.

Problem 3.19. In the sweeping preconditioner approach it is needed to solve linear systems with the matrices S_j in order to compute the Schur complement. See Step 4 in Algorithm 6. Use the code in Problem 3.13 to construct the matrix S_5. Study the structure of the matrix S_5 and test the peformance of the preconditioner P which contains the diagonal part of S_5.

Problem 3.20. Hierarchical matrices arise in discretization of PDEs and operators (see Theorem 3.7). Give an example of function $G(x, y)$ such that its discretization matrix, defined as $A = [G(x_i, y_j)]_{i,j=1}^n$, is hierarchical, where $\{x_i, y_j\}$ is a uniform discretization of a subset of the support of G.