

## Efficient algorithms for wave problems

Présentée le 24 septembre 2021

Faculté des sciences de base  
Chaire de mathématiques computationnelles et science de la simulation  
Programme doctoral en mathématiques

pour l'obtention du grade de Docteur ès Sciences

par

### Boris BONEV

Acceptée sur proposition du jury

Prof. F. Eisenbrand, président du jury  
Prof. J. S. Hesthaven, directeur de thèse  
Prof. P.-G. Martinsson, rapporteur  
Prof. R. Abgrall, rapporteur  
Prof. D. Kressner, rapporteur

**Disclaimer**

Doctoral thesis submitted the 21. of July 2021 in partial fulfillment of the requirements for the degree of Doctor of Philosophy at the Department of Mathematics at EPFL.

**Copyright notice**

 Except otherwise noted, this work is licensed under the Creative Commons Attribution-NonCommercial-NoDerivs 2.0 Generic License.

To view a copy of the CC BY-NC-ND code, visit:

<http://creativecommons.org/licenses/by-nc-nd/2.0/>

**Colophon**

This document was typeset with the help of KOMA-Script and L<sup>A</sup>T<sub>E</sub>X using the kaobook class.

**Publisher**

Published by EPFL, August 2021.

To my grandfather.



# Acknowledgements

First and foremost, I would like to thank my thesis supervisor, Prof. Jan S. Hesthaven, for this invaluable opportunity, his patience, and guidance throughout these years. Thanks to him, I have learned how to approach problems, both in science and in life. He has taught me to persevere and have confidence that there is light at the end of the tunnel. He has taught me to pick my battles and focus on what matters. I am fortunate to be able to say that I have learned from the best.

Much of this work would not have been possible without the work and help of Prof. Frank Giraldo, Prof. Michal Kopera, Dr. Mahya Hajihassanpour, and Zilan Cheng. I am very grateful for having had the opportunity of working and learning with them.

Special thanks are due to the jury members, Prof. Rémi Abgrall, Prof. Per-Gunnar Martinsson, Prof. Daniel Kressner, and Prof Friedrich Eisenbrand, who have taken their time to read and review this dissertation and serve as jury members. Their questions and comments have helped me in improving upon the original manuscript.

I would also like to thank the anonymous reviewers of past publications for reading my manuscripts and providing me with their helpful comments. It forced me to rethink ideas and concepts and better understand my work.

I thank my colleagues, and in particular, I would like to thank Dr. Stefano Massei, Prof. Daniel Kressner, and Dr. Fabian Mönkeberg, for always having an open door and the patience to answer my questions.

I would like to acknowledge the Swiss National Science Foundation (SNSF) which has supported this work under grant 513966.

I am fortunate to have some amazing people in my life. In no particular order, I would like to thank:

Martin and Mehdi for helping me proofread this thesis and giving me their suggestions on how to improve it.

Clara for feeding me, listening to me talk about boring matrices and distracting me when I needed it.

My friends Greg, Iordan, Mehdi, Rami, Martin, Martin, Fernando, Fabian, Ludger, Youssef, Youssef, Léo, Emile, Paride, Léo, Hugo, Alessandro, Chiara, Ivan, Benoit, Elena, Tomislav.

My family, for always being there when I needed them.

# Abstract

Wave phenomena manifest in nature as electromagnetic waves, acoustic waves, and gravitational waves among others. Their descriptions as partial differential equations in electromagnetics, acoustics, and fluid dynamics are ubiquitous in science and engineering. Having numerical methods to solve these problems efficiently is therefore of great importance and value to domains such as aerospace engineering, geophysics, and civil engineering. Wave problems are characterized by the finite speeds at which waves propagate and present a series of challenges for the numerical methods aimed at solving them. This dissertation is concerned with the development and analysis of numerical algorithms for solving wave problems efficiently using a computer. It contains two parts:

The first part is concerned with sparse linear systems which stem from discretizations of such problems. An approximate direct solver is developed, which can be computed and applied in quasilinear complexity. As such, it can also be used as a preconditioner to accelerate the computation of solutions using iterative methods. This direct solver is based on structured Gaussian elimination, using a nested dissection reordering and the compression of dense, intermediate matrices using rank structured matrix formats. We motivate the use of these formats and demonstrate their usefulness in our algorithm. The viability of the method is then verified using a variety of numerical experiments. These confirm the quasilinear complexity and the applicability of the method.

The second part focuses on the solution of the shallow water equations using the discontinuous Galerkin method. These equations are used to model tsunamis, storm surges, and weather phenomena. We aim to model large-scale tsunami events, as would be required for the development of an early-warning system. This necessitates the development of a well-balanced numerical scheme, which is efficient, flexible, and robust. We analyze the well-balanced property in the context of discontinuous Galerkin methods and how it can be obtained. Another problem that arises with the shallow water equations is the presence of dry areas. We introduce methods to handle these in a well-balanced, and physically consistent manner. The resulting method is validated using tests in one dimension, as well as simulations on the surface of the Earth. The latter are compared to real-world data obtained from buoys and satellites, which demonstrate the applicability and accuracy of our method.

**Keywords:** partial differential equations, sparse linear system, hierarchical matrices, low-rank approximation, HSS matrices, nested dissection, structured elimination, Gaussian elimination, multifrontal method, iterative solver, preconditioner, Poisson problem, Helmholtz problem, elastic wave equation, shallow water equations, discontinuous Galerkin method, wetting/drying, well-balanced schemes, tsunami simulation

# Zusammenfassung

Wellenphänomene kommen in der Natur unter anderem als elektromagnetische Wellen, akustische Wellen und Gravitationswellen vor. Ihre Beschreibung als partielle Differentialgleichungen in den Bereichen des Elektromagnetismus, Akustik und der Fluidodynamik sind daher allgegenwärtig in den Natur- und Ingenieurwissenschaften. Numerische Methoden zur effizienten Lösung dieser Probleme sind daher unschätzbar wertvoll für Domänen wie z.B. die Luft- und Raumfahrttechnik, die Geophysik und das Bauingenieurwesen. Eine wichtige Eigenschaft dieser Wellenphänomene ist die begrenzte Geschwindigkeit, mit der sich die Wellen ausbreiten. Wellenprobleme stellen eine Reihe an Herausforderungen an die assoziierten numerischen Methoden. Diese Dissertation beschäftigt sich mit der Entwicklung und Analyse numerischer Methoden und Algorithmen zur effizienten Lösung von partiellen Differentialgleichungen, die Wellenphänomene beschreiben, mithilfe eines Computers. Sie besteht aus zwei Teilen:

Der erste Teil beschäftigt sich mit dünnbesetzten Gleichungssystemen, wie sie bei der Diskretisierung von solchen Problemen vorkommen. Es wird ein approximativer Löser vorgestellt, welcher in quasi-linearer Zeit berechnet und angewendet werden kann. Der Löser kann daher auch, als Präkonditionierer angewandt werden, um die Konvergenzraten iterativer Löser zu beschleunigen. Unser Löser basiert auf strukturierten GauSS-Eliminationsverfahren anhand von verschachtelter Unterteilung und der Kompression von dichtbesetzten Matrizen anhand von hierarchischen Matrizen. Wir motivieren und zeigen die Nützlichkeit von hierarchischen Matrizen in der Entwicklung effizienter Lösungsverfahren. Anschließend demonstrieren wir die Anwendbarkeit und die Komplexität unseres Lözers mithilfe von numerischen Experimenten.

Im zweiten Teil stellen wir diskontinuierliche Galerkin Methoden zur Lösung der Flachwassergleichungen vor. Diese Gleichungen werden zur Modellierung von Tsunamis, Sturmfluten und anderer Wetterphänomene verwendet. Wir entwickeln ein Modell zur Simulation von Tsunami-Phänomenen, wie es für die Entwicklung eines Frühwarnsystems notwendig ist. Dies benötigt die Entwicklung eines numerischen Verfahrens, welches gewisse stationäre Gleichgewichte exakt erhält. Diese Eigenschaft ist wichtig, um die Plausibilität der Lösungen zu gewährleisten. Wir analysieren diskontinuierliche Galerkin Verfahren unter diesem Aspekt und entwickeln darüber hinaus Methoden zur Behandlung der Phasengrenzen. Im Anschluss verifizieren wir unsere Methoden anhand von Beispielen in einer Dimension und Simulationen auf der Oberfläche der Erde, welche wir mit Satellitendaten und Bojendaten vergleichen.

**Schlüsselwörter:** partielle Differentialgleichungen, dünnbesetzte Gleichungssysteme, hierarchische Matrizen, niedrig-Rang Approximation, hierarchisch semi-separable Matrizen, verschachtelte Unterteilung, strukturierte GauSS-Elimination, Iterative Löser, Präkonditionierer, Poisson-Gleichung, Helmholtz-Gleichung, elastische Wellengleichung, Flachwassergleichung, diskontinuierliche Galerkin Verfahren, Phasengrenzen, wohlbalancierte Methoden, Tsunami Simulation

# Résumé

La physique des ondes apparaît dans la Nature entre autres sous forme d'ondes électromagnétiques, d'ondes sonores ou encore d'ondes gravitationnelles. Les équations aux dérivées partielles correspondantes à ces phénomènes sont omniprésentes en sciences et dans leurs applications industrielles. Pour cette raison, il est très utile d'avoir des méthodes numériques permettant résoudre ces problèmes efficacement pour des applications allant de l'ingénierie aérospatiale, à la géophysique en passant par l'ingénierie civile. Une des caractéristiques de ces problèmes est la nature finie de la vitesse de propagation des ondes. Les méthodes numériques dédiées à leur résolution sont aussi confrontés à une multitude de difficultés. Cette thèse a pour but le développement et l'analyse d'algorithmes numériques permettant de résoudre efficacement certains problèmes de propagation d'ondes au moyen d'ordinateurs. Elle se compose de deux parties :

La première partie considère les systèmes linéaires creux qui sont obtenus suite à la discréttisation de certains problèmes mentionnés ci-dessus. Une méthode consistant à résoudre une approximation du système linéaire avec un solveur direct est développée. Elle possède une complexité quasi-linéaire. En tant que telle, elle peut aussi servir de préconditionneur pour réduire le temps de calcul de méthodes itératives. Le solveur direct approché repose sur une méthode structurée du Pivot de Gauss, en utilisant une méthode de dissection emboîtée qui réordonne et compresse des matrices intermédiaires denses selon une structure dépendante de leur rang. Nous justifions l'utilisation de ces structures particulières. Nous montrons leur utilité dans notre algorithme. La réussite de la méthode est ensuite vérifiée au travers de plusieurs expériences numériques. Elles confirment la complexité quasi-linéaire et l'efficacité de la méthode.

La seconde partie étudie les solutions des équations de Barré de Saint-Venant ou équations des écoulements en eau peu profonde en utilisant une méthode de Galerkin discontinue. Ces équations sont utilisées pour modéliser les tsunamis, les ondes de tempêtes et des phénomènes météorologiques similaires. Nous avons pour but de modéliser l'apparition de tsunamis à large échelle, ce qui est nécessaire pour le développement d'un système d'alarme prévisionnel. Pour cela, il est nécessaire de développer une méthode numérique flexible, efficace, robuste qui, en particulier, préserve la stabilité lorsque le système est dans un état stationnaire (schéma équilibré). Nous analysons cette propriété de stabilité spécifiquement dans le contexte de méthodes de Galerkin discontinues et nous montrons comment celle-ci peut être obtenue systématiquement. Un autre problème découlant des équations en eau peu profonde est la présence de régions sèches. Nous introduisons des méthodes pour traiter ces régions de manière conservative et cohérente du point de vue physique. Le modèle est validé en une dimension, puis avec des simulations sur la surface terrestre. Ces dernières sont comparées avec des données expérimentales obtenues par des bouées et des satellites. Ces résultats montrent l'efficacité et la précision de cette méthode.

**Mots-clés :** équations aux dérivées partielles, système linéaire creux, matrices hiérarchiques, approximation de bas rang, matrices hiérarchiques semi-séparables, dissection emboîtée, élimination structurée, pivot de Gauss, méthode multifrontale, solveur itératif, préconditionneur, problème de Poisson, équation de Helmholtz, équation d'onde élastique, équations de Barré de Saint-Venant, méthode de Galerkin discontinue, mouillage/séchage, schéma de conservation, simulation de tsunami.

# Preface

Numerical analysis is the study of algorithms  
for the problems of continuous mathematics  
- Lloyd N. Trefethen [1]

This thesis contains two parts - one on hierarchical preconditioners for large linear systems and one on discontinuous Galerkin methods for the shallow water equations. They are connected - the continuous problems that we design algorithms for are wave problems. Apart from that, I have decided to keep them as separate as possible, such that both parts can stand on their own. This is intended to aid the reader who is more interested in one topic over the other.

The first part focuses on efficient preconditioners, as well as sparse direct solvers for wave problems and elliptic problems in particular. It introduces some fundamental concepts on finite element approximation, low-rank approximation, sparse direct solvers, iterative solvers and hierarchical matrices, before explaining our work on hierarchical solvers and preconditioners for elliptic problems. The second part on the other hand, focuses on the shallow water equations and the discontinuous Galerkin method. We discuss some important notions on well-balanced methods and wetting-drying, before we move on to design a method intended for large-scale ocean, tsunami, and storm surge modeling.

The work as presented here has either been published or submitted for publication in the following articles:

- [2] Boris Bonev, Jan S. Hesthaven, Francis X. Giraldo, and Michal A. Kopera. 'Discontinuous Galerkin scheme for the spherical shallow water equations with applications to tsunami modeling and prediction'. In: *Journal of Computational Physics* (2018)
- [3] Mahya Hajihassanpour, Boris Bonev, and Jan S. Hesthaven. 'A comparative study of earthquake source models in high-order accurate tsunami simulations'. In: *Ocean Modelling* 141. August (Sept. 2019)
- [4] Boris Bonev and Jan S. Hesthaven. 'A hierarchical preconditioner for wave problems in quasilinear complexity'. In: 513966 (May 2021). arXiv: [2105.07791](https://arxiv.org/abs/2105.07791)

To skip directly to our contributions on preconditioning for wave problems, I suggest reading Chapter 7 and Chapter 8. For our contributions to numerical schemes for the schallow water equations, I suggest reading Chapter 11, Chapter 12 and Chapter 13.

Research should be reproducible wherever possible. Moreover, the best way of understanding algorithms is often to implement and play with them. Most of the codes related to this thesis have been made available at [github.com/bonevbs](https://github.com/bonevbs). I have made an effort to reference relevant sections to the codes wherever possible. The reader is encouraged to use the hyperlinks provided on the side to check them out.

Boris Bonev

# Contents

Preface	ix
Contents	x
<b>HIERARCHICAL PRECONDITIONERS</b>	<b>1</b>
<b>1 Motivation</b>	<b>2</b>
1.1 Some related problems . . . . .	2
1.2 Finite element approximation . . . . .	4
1.3 Green's function . . . . .	6
<b>2 Low-rank approximation</b>	<b>8</b>
2.1 Linear algebra basics . . . . .	8
2.2 Sparse matrices . . . . .	12
2.3 Low-rank matrices . . . . .	14
2.4 Rank-revealing QR . . . . .	15
2.5 Random sampling . . . . .	16
<b>3 Sparse direct solvers</b>	<b>20</b>
3.1 Graph elimination . . . . .	20
3.2 LDR Factorization . . . . .	21
3.3 Fill-in and reorderings . . . . .	23
3.4 Structured elimination . . . . .	26
<b>4 Iterative solvers</b>	<b>29</b>
4.1 Krylov spaces . . . . .	29
4.2 The Arnoldi iteration . . . . .	29
4.3 GMRES . . . . .	31
4.4 Convergence of GMRES . . . . .	32
4.5 Preconditioning . . . . .	34
<b>5 Hierarchical matrices</b>	<b>37</b>
5.1 Approximate separability . . . . .	37
5.2 Block cluster trees . . . . .	43
5.3 Hierarchical matrices . . . . .	45
5.4 Nested bases . . . . .	47
<b>6 Algorithms for hierarchical matrices</b>	<b>51</b>
6.1 HODLR arithmetic . . . . .	51
6.2 HSS arithmetic . . . . .	53
6.3 HSS compression . . . . .	57
6.4 HssMatrices.jl . . . . .	63
<b>7 Hierarchical approximate solvers</b>	<b>66</b>
7.1 Compressing the fill-in . . . . .	66
7.2 Existing methods . . . . .	69
7.3 Approximate factorization . . . . .	70

7.4	Complexity of the algorithm . . . . .	79
<b>8</b>	<b>Numerical Experiments</b>	<b>81</b>
8.1	Parameters . . . . .	81
8.2	Poisson problem . . . . .	82
8.3	Helmholtz problem . . . . .	84
8.4	Scaling and performance . . . . .	92
8.5	Codes for reproducibility . . . . .	95
8.6	Concluding remarks . . . . .	96
<b>DISCONTINUOUS GALERKIN METHODS FOR THE SHALLOW WATER EQUATIONS</b>		<b>97</b>
<b>9</b>	<b>Motivation</b>	<b>98</b>
9.1	The shallow water equations . . . . .	99
9.2	A simple scheme . . . . .	102
<b>10</b>	<b>The discontinuous Galerkin method</b>	<b>106</b>
10.1	In one dimension . . . . .	106
10.2	On the Sphere . . . . .	109
10.3	A few words on meshes . . . . .	113
10.4	Time integration . . . . .	115
<b>11</b>	<b>Well-balanced schemes</b>	<b>118</b>
11.1	The well-balanced property . . . . .	118
11.2	Hydrostatic reconstruction . . . . .	119
11.3	Well-balanced DG schemes . . . . .	121
11.4	Non-conforming meshes . . . . .	122
<b>12</b>	<b>Wet/dry transitions</b>	<b>125</b>
12.1	A survey of existing methods . . . . .	125
12.2	Maintaining positivity . . . . .	126
12.3	Flux discretization . . . . .	129
12.4	A few notes on stability . . . . .	130
<b>13</b>	<b>Numerical Results</b>	<b>132</b>
13.1	Results in one dimension . . . . .	132
13.2	Results on the sphere . . . . .	138
13.3	Dynamic source models . . . . .	143
13.4	Concluding remarks . . . . .	148
<b>Bibliography</b>		<b>150</b>

# List of Figures

1.1	Numerical solutions of the Helmholtz problem using finite elements . . . . .	3
1.2	Triangular mesh on a guitar shaped domain. . . . .	5
1.3	Sparsity pattern of a Helmholtz problem . . . . .	6
2.1	Illustration of the compressed sparse column format. . . . .	13
2.2	Low-rank approximation of the EPFL logo. . . . .	14
2.3	Comparison of rank estimators. . . . .	18
3.1	Sparsity pattern and corresponding adjacency graph. . . . .	20
3.2	Equivalence of sparse Gaussian elimination and graph elimination. . . . .	21
3.3	Elimination and fill-in in the arrowhead matrix. . . . .	23
3.4	Sparsity pattern of the $L$ factor for the Helmholtz problem . . . . .	23
3.5	Sparsity pattern of the Galerkin matrix reordered using Reverse Cuthill-McKee. .	24
3.6	Nested dissection of a finite element mesh. . . . .	25
3.7	Nested dissection and its associated elimination tree. . . . .	25
3.8	Sparsity pattern of the Galerkin matrix after nested dissection reordering. .	26
3.9	Elimination of a single supernode. . . . .	26
4.1	Influence of the spectrum on the convergence rate of GMRES. . . . .	34
4.2	Convergence and spectrum of the preconditioned Helmholtz equation. . . . .	35
5.1	Illustration of the admissibility condition. . . . .	40
5.2	Example of a cluster tree . . . . .	42
5.3	Example of a cluster tree . . . . .	44
5.4	Block partitioning for an admissibility condition in one dimension. . . . .	44
5.5	HODLR block partitioning. . . . .	46
5.7	Illustration of a HSS block row and a block column. . . . .	48
6.1	Illustration of the ULV factorization algorithm for HSS matrices. . . . .	55
6.2	Illustration of numbering in the HSS hierarchy. . . . .	58
6.3	Visualization of HSS rank structure using HssMatrices.jl. . . . .	63
6.4	Timings and memory requirements of various HSS algorithms in HssMatrices.jl.	65
7.1	Compressibility of the stiffness matrix. . . . .	68
7.2	Rank structure of the top-level Schur complement in 2D. . . . .	68
7.3	$\mathcal{H}$ -matrix rank structure of the top-level Schur complement in 3D. . . . .	69
7.4	HSS rank structure of the top-level Schur complement in 3D. . . . .	69
7.5	Illustration of well-separated nodes. . . . .	72
7.6	HSS block structure of the Schur complement. . . . .	73
7.7	Illustration of the connectivity between $\mu$ and $\nu$ for finite element discretizations. . . . .	74
7.8	Elimination tree with switching level. . . . .	78
8.1	Relative residual for each iteration of the preconditioned GMRES applied to the Poisson problem. . . . .	82
8.2	Preconditioner performance under h-refinement for the Poisson problem. . .	83
8.3	Preconditioner performance under p-refinement for the Poisson problem. .	84

8.5	Influence of the preconditioner hierarchy on GMRES convergence for a compression tolerance of $10^{-6}$ . . . . .	85
8.6	Influence of the preconditioner hierarchy on GMRES convergence for a compression tolerance of $10^{-4}$ . . . . .	85
8.7	Preconditioner performance for the Helmholtz problem at various wave numbers. . . . .	86
8.8	Preconditioner performance for the Helmholtz problem under h-refinement. . . . .	86
8.4	Top-level Schur complements for the Helmholtz problem. . . . .	86
8.9	Preconditioner performance for the Helmholtz problem under p-refinement. . . . .	87
8.10	Solution of the Helmholtz problem on a guitar shaped domain. . . . .	87
8.11	Relative residual for various wave numbers on the guitar shaped domain with a compression tolerance of $10^{-6}$ . . . . .	88
8.12	Relative residual for various wave numbers on the guitar shaped domain with a compression tolerance of $10^{-5}$ . . . . .	88
8.13	Randomly generated heterogeneous zones. . . . .	88
8.14	Solution of the Helmholtz problem with heterogeneous material coefficients. . . . .	89
8.15	Preconditioner performance for the elastic wave problem under h-refinement. . . . .	90
8.16	Preconditioner performance for the elastic wave problem under p-refinement. . . . .	90
8.17	Material distributions in the Marmousi II test case. . . . .	91
8.18	Solution of the frequency-domain elastic wave equations for the Marmousi II test case. . . . .	91
8.19	Preconditioner performance for the Marmousi II test case. . . . .	92
8.20	Timings and memory requirements under $h$ -refinement. . . . .	93
8.21	Timings and memory requirements for the Helmholtz problem under $\kappa h$ -refinement. . . . .	94
9.1	Hokusai's Great Wave off Kamigawa. . . . .	98
9.2	Illustration of the assumptions for the shallow water equations in one dimension. . . . .	100
10.1	Comparison of the finite volume scheme to the discontinuous Galerkin scheme for a smooth solution. . . . .	109
10.2	Transformation into the reference element. . . . .	110
10.3	Generation of icosahedral meshes on the sphere. . . . .	113
10.4	Illustration of an adaptively refined non-conforming mesh. . . . .	114
10.5	Illustration of a mesh hierarchy as a forest. . . . .	114
10.6	Treatment of the hanging node in non-conforming discretizations. . . . .	114
10.7	Linear stability regions of the explicit Euler and SSPRK(3,3) methods . . . . .	116
12.1	The challenge of preserving the lake at rest solution in partly dry elements. . . . .	129
13.1	Convergence behavior of our wetting/drying scheme for a standing wave. . . . .	133
13.2	Comparison of regular DG method and our method on the lake at rest solution on a sloping beach. . . . .	133
13.3	Comparison of conservation errors for the lake at rest solution. . . . .	134
13.4	Dam break on a dry bed at different times. . . . .	135
13.5	Convergence behavior for the dam break solution on a dry bed. . . . .	136
13.6	Oscillating lake in a parabolic bed. . . . .	137
13.7	Convergence behavior for the oscillating lake in a parabolic channel. . . . .	138
13.8	Depiction of an adaptive tsunami simulation on the sphere. . . . .	140
13.9	Locally refined mesh for the simulation of the Tohoku tsunami. . . . .	140
13.10	High-fidelity simulation of the Tohoku tsunami. . . . .	141

13.11	Comparison of simulation data to real-world buoy data, recorded during the Tohoku tsunami. . . . .	142
13.12	Comparison of static and dynamic source models using wave signals from DART buoy 21418 for the Tohoku tsunami. . . . .	145
13.13	Discontinuous Galerkin simulation of the Sumatra-Andaman tsunami. . . .	146
13.14	Subfault locations and satellite data for the Sumatra-Andaman tsunami. . .	146
13.15	Comparison of the simulation results to satellite measurement and results obtained by others. . . . .	147
13.16	Comparison of static and dynamic source models to the signals extracted from satellite data during the Sumatra tsunami event. . . . .	148

## List of Tables

1.1	Green's functions of various elliptic operators . . . . .	7
2.1	Complexity of matrix factorizations using dense matrices . . . . .	12
2.2	Computational complexity of common operations using low-rank matrices. . . .	14
6.1	Computational complexity of arithmetic using HODLR matrices. . . . .	53
6.2	Computational complexity of arithmetic using HSS matrices. . . . .	53
7.1	Computational cost of operations arising in the factorization and application of the preconditioner. . . . .	79
8.1	Comparison of both dissection methods for the heterogeneous Helmholtz problem. . . . .	89
8.2	Factorization and application times for the Poisson problem under $h$ -refinement. . . . .	92
8.3	Factorization and application times for the Helmholtz problem under $h$ -refinement. . . . .	93
8.4	Factorization and application times with a direct solver for the Helmholtz problem under $h$ -refinement. . . . .	93
8.5	Factorization and application times for the Helmholtz problem under $\kappa h$ -refinement with $\kappa h = 1/4$ . . . . .	94
8.6	Factorization and application times for the Helmholtz problem under $\kappa h$ -refinement with $\kappa h = 1/2$ . . . . .	94
8.7	Factorization and application times for the three-dimensional Poisson problem under $h$ -refinement. . . . .	95
10.1	Butcher Tableaus for a general, explicit Runge-Kutta scheme. . . . .	116
10.2	Butcher Tableau for the strong-stability preserving Runge-Kutta 3,3 scheme. .	117
13.1	Relative errors for the lake at rest solution on the sphere. . . . .	139
13.2	Comparison of forecasted and recorded tsunami arrival times. . . . .	143
13.3	Subfault parameters for the Sumatra-Andaman tsunami. . . . .	147

# **HIERARCHICAL PRECONDITIONERS**

# 1

## Motivation

The continuous problems of physics that we are concerned with are *wave problems*, which are modelled within the more general field of *partial differential equations* (PDEs). Wave problems play an important role in physics and engineering and present a wealth of interesting phenomena such as scattering, dispersion and shock waves. The most basic mathematical description of wave phenomena is the (linear) wave equation

$$u_{tt} - c^2 \nabla^2 u = f, \quad (1.1)$$

subject to suitable initial and boundary conditions. Here,  $u : \Omega \times [0, \infty) \rightarrow \mathbb{R}$  is the solution, defined for a coordinate  $x$  in the  $d$ -dimensional spatial domain  $\Omega \subseteq \mathbb{R}^d$  and a time  $t$  in the domain  $[0, \infty)$ .  $u_{tt}$  denotes the second partial derivative with respect to time  $\partial^2 u / \partial t^2$  and the Laplacian  $\nabla^2$  is taken with respect to the spatial coordinates  $x$ . The constant  $c$  denotes the wave propagation speed and  $f$  is a function that acts as forcing term for the equation. In one dimension, for instance, the wave equation (1.1) describes the motion of a vibrating guitar string. In two dimensions, it describes the movement of fluid surfaces, e.g. water waves.

### 1.1 Some related problems

A classical approach to solve the wave equation is to exploit its linearity and use the Fourier transform

$$u(t) = \frac{1}{2\pi} \int_{-\infty}^{\infty} \tilde{u}(\omega) \exp(i\omega t) d\omega, \quad (1.2)$$

which yields the frequency-domain version of (1.1)

$$-\omega^2 \tilde{u}^2 - c^2 \nabla^2 \tilde{u} = \tilde{f}.$$

Here,  $\omega$  is the angular frequency, and  $\tilde{u}$ ,  $\tilde{f}$  denote the Fourier transforms (in time,  $x$  is kept constant) of  $u$ ,  $f$ . We drop the tilde  $\sim$  and introduce the wavenumber  $\kappa = \omega/c$  for simplicity. This yields our first elliptic PDE, the Helmholtz problem:

**Problem 1.1.1** (Helmholtz problem) Let  $\Omega \subset \mathbb{R}^d$  be an open, bounded Lipschitz domain and  $\Gamma_D, \Gamma_N \subseteq \partial\Omega$  such that  $\Gamma_D \cup \Gamma_N = \partial\Omega$  and  $\Gamma_D \cap \Gamma_N = \emptyset$ . Find  $u : \Omega \times [0, \infty) \rightarrow \mathbb{R}$ , such that

$$-\nabla^2 u - \kappa^2 u = f \quad \text{in } \Omega, \quad (1.3a)$$

$$u = g_D \quad \text{on } \Gamma_D, \quad (1.3b)$$

$$\nabla u \cdot \hat{n} = g_N \quad \text{on } \Gamma_N, \quad (1.3c)$$

subject to suitable boundary data  $g_D : \Gamma_D \rightarrow \mathbb{R}$  and  $g_N : \Gamma_N \rightarrow \mathbb{R}$ .

By setting  $\kappa = 0$ , we recover the related Poisson problem:

1.1 Some related problems . . . . .	2
1.2 Finite element approximation . . . . .	4
1.3 Green's function . . . . .	6

**Problem 1.1.2** (Poisson problem) Let  $\Omega, \Gamma_D, \Gamma_N$  be as in Problem 1.1.1. Then, find  $u : \Omega \times [0, \infty) \rightarrow \mathbb{R}$ , such that

$$-\nabla^2 u = f \quad \text{in } \Omega, \quad (1.4a)$$

$$u = g_D \quad \text{on } \Gamma_D, \quad (1.4b)$$

$$\hat{n} \cdot \nabla u = g_N \quad \text{on } \Gamma_N, \quad (1.4c)$$

subject to suitable boundary data  $g_D, g_N$  on  $\Gamma_D, \Gamma_N$ .

Both problems are related in the sense that the homogeneous Helmholtz problem 1.1.1 is the eigenvalue problem of the Poisson problem, where  $\kappa^2$  corresponds to the eigenvalue of the Laplacian. Therefore, if a wavenumber coincides with the square-root of an eigenvalue of the Laplacian, the Helmholtz problem 1.1.1 becomes singular. Figure 1.1 depicts some numerically computed solutions to the two-dimensional Helmholtz problem on the square domain  $\Omega = [-1, 1]^2$  with  $f = 1$  and homogeneous boundary conditions.

We observe that with increasing wavenumber, the solutions become more and more oscillatory. It is often useful to refer to the wavelength of the problem, which is  $\lambda = 2\pi/\kappa$ .

Another way to understand the Poisson problem 1.1.2 is as the steady-state solution to the wave problem (1.1). Therefore, it can describe e.g. the constant displacement of a string under tension. This bring us to the elastic wave equations in an inhomogeneous but isotropic medium

$$u_{tt} - \nabla \cdot \sigma(u) = f, \quad (1.5)$$

to which suitable boundary conditions must be specified. Now,  $u, f : \Omega \times [0, \infty) \rightarrow \mathbb{R}^d$  are vector-valued functions and the symbol  $\sigma(\cdot)$  denotes the  $d$ -by- $d$  Cauchy stress tensor given by

$$\sigma(u) = \lambda(\nabla \cdot u)\mathbf{I} + \mu(\nabla u + (\nabla u)^T), \quad (1.6)$$

where  $\lambda, \mu : \Omega \rightarrow \mathbb{R}$  are the Lamé parameters of the material and  $\mathbf{I}$  denotes the  $d$ -by- $d$  identity matrix. These equations can be understood as one possible generalization of the wave equation (1.1). The characteristic property of the elastic wave equation is that it permits both pressure and shear waves and are therefore able to describe waves in both solids and fluids.<sup>1</sup>

As for the wave equation (1.1), we recover the associated elliptic problem by considering the steady-state problem.

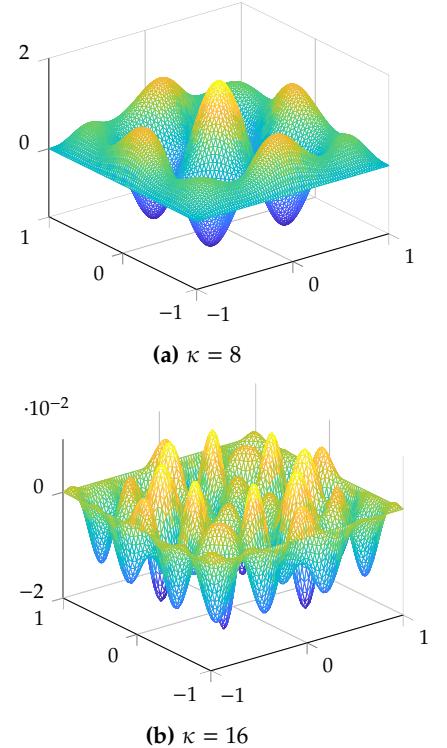
**Problem 1.1.3** (Steady-state elastic waves) Let  $\Omega, \Gamma_D, \Gamma_N$  be as in Problem 1.1.1. Then, find  $u : \Omega \times [0, \infty) \rightarrow \mathbb{R}$ , such that

$$-\nabla \cdot \sigma(u) = f \quad \text{in } \Omega, \quad (1.7a)$$

$$u = g_D \quad \text{on } \Gamma_D, \quad (1.7b)$$

$$\sigma(u) \cdot \hat{n} = g_N \quad \text{on } \Gamma_N, \quad (1.7c)$$

subject to suitable boundary data  $g_D, g_N$  on  $\Gamma_D, \Gamma_N$ .



**Figure 1.1:** Numerical solutions to the Helmholtz equation on  $\Omega = [-1, 1]^2$  with  $f = 1$ . We chose a finite element discretization with  $p = 1$  and  $h = 1/80$ .

1: Pressure waves are characterized by the displacement being aligned with the direction of wave propagation. The displacement in shear waves is perpendicular to the propagation of the wave.

Or, once again, we can use the Fourier transform (1.2) and recover the frequency-domain elastic wave equation.

**Problem 1.1.4** (Frequency-domain elastic waves) Let  $\Omega, \Gamma_D, \Gamma_N$  be as in Problem 1.1.1. Then, find  $u : \Omega \times [0, \infty) \rightarrow \mathbb{R}$ , such that

$$-\nabla \cdot \sigma(u) - \rho \omega^2 u = f \quad \text{in } \Omega, \quad (1.8a)$$

$$u = g_D \quad \text{on } \Gamma_D, \quad (1.8b)$$

$$\sigma(u) \cdot \hat{n} = g_N \quad \text{on } \Gamma_N, \quad (1.8c)$$

subject to suitable boundary data  $g_D, g_N$  on  $\Gamma_D, \Gamma_N$ .

These equations have their applications in fields ranging from geophysics to civil engineering. They are frequently used to model seismic waves and are the basis for seismic imaging techniques such as full waveform inversion [5].

## 1.2 Finite element approximation

One of the most ubiquitous methods in science and engineering for the solution of partial differential equations is the finite element method (FEM) [6].

We are concerned with finding the numerical solution of second-order elliptic problems of the form

$$\mathcal{L}u = f \quad \text{in } \Omega, \quad (1.9a)$$

$$u = g_D \quad \text{on } \Gamma_D, \quad (1.9b)$$

$$\nabla u \cdot \hat{n} = g_N \quad \text{on } \Gamma_N, \quad (1.9c)$$

on open, bounded Lipschitz domains  $\Omega \subset \mathbb{R}^d$  subject to suitable boundary data  $g_D$  and  $g_N$  on the Dirichlet and Neumann boundaries  $\Gamma_D, \Gamma_N$ , respectively, with  $\Gamma_D \cup \Gamma_N = \partial\Omega$  and  $\Gamma_D \cap \Gamma_N = \emptyset$ . The unit outward normal vector on the boundary  $\partial\Omega$  is denoted by  $\hat{n}$ . We are particularly interested in second-order elliptic operators of the form

$$\mathcal{L}u = -\nabla \cdot (\mathbf{A}\nabla u + \mathbf{b}u) + \mathbf{c} \cdot \nabla u + du, \quad (1.10)$$

where  $\mathbf{A}, \mathbf{b}, \mathbf{c}, d$  are coefficients with suitable dimensions and variable in space.

To formulate a finite element method, we consider a variational version of the problem in some Hilbert space  $V$ .<sup>2</sup> <sup>3</sup> Then, the variational formulation is: Find  $u \in V$ , such that

$$\forall v \in V : \quad a(u, v) = l(v), \quad (1.11)$$

where

$$a(u, v) = \langle v, \mathcal{L}u \rangle = \int_{\Omega} v \mathcal{L}u \, dx \quad (1.12)$$

is a bilinear form on  $V \times V$  and

$$l(v) = \langle v, f \rangle = \int_{\Omega} v f \, dx \quad (1.13)$$

2: Discussion on the correct choice of function spaces  $V$  and the existence and uniqueness of solutions therein can be found in [7].

3: Under certain assumptions, one can show that this formulation is equivalent to the differential formulation. This again depends mainly on the choice of solution space  $V$ . For details see [6–9].

a linear form on  $V$ . An important requirement here is that the solution space  $V$  must satisfy the boundary conditions or the problem may not be well-defined (or equivalent to (1.9)). Let us restrict ourselves to the Helmholtz problem (1.3) with homogeneous boundary Dirichlet data  $g_D = 0$  everywhere ( $\Gamma_D = \partial\Omega$ ). For this special case, the bilinear form (1.12) is

$$a(u, v) = \int_{\Omega} \nabla v \cdot \nabla u - \kappa^2 v \cdot u \, dx. \quad (1.14)$$

To discretize the problem, we choose a finite sub-vector space  $V_h \subset V$  which is a good approximation of  $V$ . Here, it is common to use the subscript letter “ $h$ ” to denote the approximating space.  $h$  usually refers to the smallest lengthscale, that is resolved by  $V_h$ . What does it mean to have a “good” approximating space? It usually means that we can control the approximation error  $\|v - v_h\|_V$ , by choosing an appropriate  $h$ :

$$\inf_{v_h \in V_h} \|v - v_h\|_V \rightarrow 0 \quad \text{for } h \rightarrow 0. \quad (1.15)$$

We then proceed to approximate the variational problem (1.11) to

**Problem 1.2.1** (Galerkin formulation) Find  $u_h \in V_h$ , such that

$$\forall v_h \in V_h : \quad a(u_h, v_h) = l(v_h). \quad (1.16)$$

This is recognized as the *Galerkin projection*, requiring that the residual  $u - u_h$  is orthogonal to  $V_h$ , i.e.

$$a(u - u_h, v_h) = a(u, v_h) - a(u_h, v_h) = l(v_h) - l(v_h) = 0.$$

In other words, the error is minimized within  $V_h$  and therefore, getting a good approximation is fundamentally determined by the choice of  $V_h$ .

A common strategy for constructing  $V_h$  is to discretize the computational domain by forming a mesh and then constructing a basis for  $V_h$  on that mesh. In that setting,  $h$  typically denotes the maximum diameter of an element in the mesh. Figure 1.2 depicts such a triangulation of a guitar shaped domain.

To construct a basis for  $V_h = \text{span}\{\varphi_1, \varphi_2, \dots, \varphi_n\}$ , basis functions are typically constructed on a element-to-element basis.<sup>4</sup> We skip the details of this step and instead note that there are two major approaches here. The first one is to construct basis functions locally which result in a discontinuous approximation space. These methods are therefore called *discontinuous Galerkin* (DG) methods. The alternative, and the classical approach for elliptic problems, is to enforce continuity up to a certain degree at the element interfaces. Therefore, such methods are called *continuous Galerkin* (CG) methods.

Thus, we assume that a basis  $\{\varphi_1, \varphi_2, \dots, \varphi_n\}$  of polynomial order  $p$  has been constructed. We can then assume that the solution  $u_h$  can be represented in this basis, i.e.

$$u_h = \sum_{j=1}^n u_j \varphi_j. \quad (1.17)$$

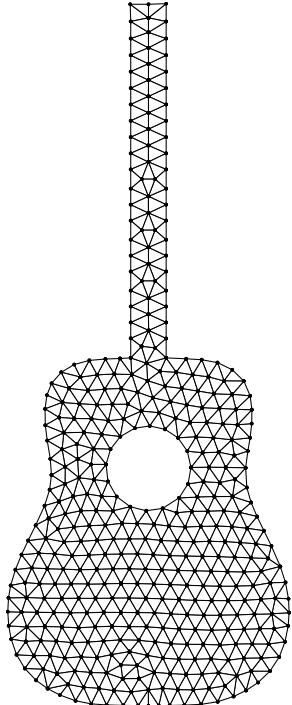


Figure 1.2: Triangular mesh on a guitar shaped domain.

4: If the basis is instead constructed globally, the methods are typically called spectral element methods (SEM). Another important family of methods are boundary element methods (BEM), which instead formulate the problem on the boundary, so that instead of the domain  $\Omega$ , the boundary of the domain  $\partial\Omega$  is discretized.

Here, we use  $\mathbf{u} = [u_1, u_2, \dots, u_n]^\top$  to denote the coefficients of  $u_h$  in the basis of  $\{\varphi_j\}_j^n$ . Inserting this and the basis for  $V_h$  into the Galerkin formulation (1.16) then yields

$$\forall i \in 1, 2, \dots, n : \sum_{j=1}^n u_j a(\varphi_j, \varphi_i) = l(\varphi_i). \quad (1.18)$$

For the Helmholtz problem with constant wavenumber and homogeneous Dirichlet boundaries this yields the linear system

$$S\mathbf{u} - \kappa^2 M\mathbf{u} = M\mathbf{f}, \quad (1.19)$$

with the mass matrix

$$M_{ij} = \int_{\Omega_h} \varphi_i \varphi_j \, dx \quad (1.20)$$

and the stiffness matrix

$$S_{ij} = \int_{\Omega_h} \nabla \varphi_i \nabla \varphi_j \, dx. \quad (1.21)$$

The right-hand side vector  $\mathbf{f}$  contains the coefficients for a representation of  $f$  in  $V_h$ . Thus, the finite element approximation is a systematic approach to convert continuously formulated problems into discrete systems of linear equations of the form

$$A\mathbf{x} = \mathbf{b}, \quad (1.22)$$

which makes them computationally tractable. For our example we have  $A = S - \kappa^2 M$ , a matrix of order  $n$ ,  $\mathbf{x} = \mathbf{u}$  is the vector of unknowns and  $\mathbf{b} = M\mathbf{f}$  is the right-hand side.

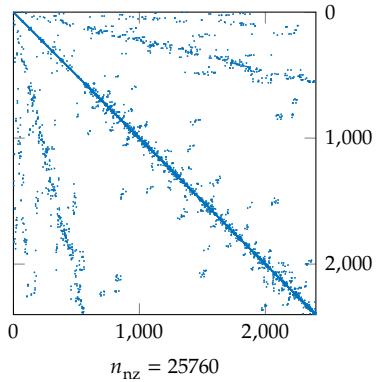
The Galerkin matrix  $A$  represents a discrete representative of our original, continuous operator  $\mathcal{L}$ . Because the chosen basis functions are compactly supported, we can expect the matrix  $A$  to be sparse, however, its bandwidth is of order  $n^{1-1/d}$ . Figure 1.3 depicts the sparsity pattern of such a matrix.<sup>5</sup> Many relevant engineering and scientific computing problems are solved using finite element approximation, and therefore result in a sparse linear system of the form (1.22). These problems are often large, which requires efficient algorithms to solve them. It is therefore crucial to design efficient algorithms, which can tackle the challenges posed by these problems.

### 1.3 Green's function

An alternative way of solving (1.9) is through the method of fundamental solutions. Loosely speaking, we say that the problem has a fundamental solution if there exists a map  $\mathcal{G}$ , which maps any suitable right-hand side to the solution  $u$ :

$$u(\mathbf{x}) = \mathcal{G}\mathbf{f} = \int_{\Omega} g(\mathbf{x}, \mathbf{y}) f(\mathbf{y}) \, d\mathbf{y}. \quad (1.23)$$

Then, the kernel function  $g(\mathbf{x}, \mathbf{y})$  is called the Green's function of the associated problem. What does the Green's function look like? Inserting



**Figure 1.3:** Sparsity pattern of a finite element discretization (discontinuous Galerkin) of the Helmholtz problem on the square domain  $\Omega = [-1, 1]^2$  with  $p = 1$  and  $h = 1/20$ . This results in a matrix of order 2401 with 25760 non-zero entries.

5: For a definition of the bandwidth of a matrix see Definition 3.3.1.

(1.23) into the problem (1.9) yields

$$\begin{aligned}\mathcal{L}\mathcal{G}f &= \mathcal{L} \int_{\Omega} g(x, y)f(y) dy \\ &= \int_{\Omega} \mathcal{L}g(x, y)f(y) dy \stackrel{!}{=} f(x).\end{aligned}$$

This property is satisfied by the Dirac delta  $\delta(x)$ <sup>6</sup> and we therefore require

$$\mathcal{L}g(x, y) = \delta(x - y). \quad (1.24)$$

A more detailed and rigorous introduction to fundamental solutions and Green's functions can be found in [7, 8, 10]. It is important to note that, much like any other solution of the PDE, the Green's function will depend on the boundary conditions that are chosen. This renders the method of fundamental solutions impractical for applications involving complex geometries and boundary conditions.

However, it is intuitive that the operator  $\mathcal{G}$  must share some properties with the inverse of the Galerkin matrix  $\mathbf{A}^{-1}$ . This is one of the central ideas of the methods presented in Chapter 5, and it is therefore useful to consider the properties of the Green's function, if it is known for the problem at hand. Table 1.1 provides an overview over some of the Green's functions that are relevant for our applications.

6: The Dirac delta is defined as the function (in the distributional sense), that satisfies

$$\delta(\varphi) = \int_{\Omega} \delta(y)\varphi(y)dy = \varphi(0),$$

for any test function  $\varphi \in C_0^{\infty}(\Omega)$ .

**Table 1.1:** Overview of some Green's functions  $g(x, y) = G(x - y)$  of various elliptic operators in various domains. Vanishing boundary conditions were used for the Laplace equations and radiating boundary conditions for the Helmholtz problems [10].

operator $\mathcal{L}$	$G(\mathbf{r})$
$-\nabla^2$ in $\mathbb{R}^2$	$\frac{1}{2\pi} \ln \ \mathbf{r}\ $
$-\nabla^2$ in $\mathbb{R}^3$	$\frac{1}{4\pi \ \mathbf{r}\ }$
$-\nabla^2 - \kappa^2$ in $\mathbb{R}^2$	$\frac{i}{4} H_0^{(1)}(\kappa \ \mathbf{r}\ )$ <sup>7</sup>
$-\nabla^2 - \kappa^2$ in $\mathbb{R}^3$	$\frac{\exp(i\kappa \ \mathbf{r}\ )}{4\pi \ \mathbf{r}\ }$

7:  $H_v^{(1)}$  denotes the Hankel function of the first kind.

# 2

## Low-rank approximation

A big portion of this work relies on low-rank approximation. To set the stage, we give a brief overview of linear algebra preliminaries. Most of the theory that is covered is a rough sketch and there are numerous excellent works on each of the subjects. These are referenced on the side. Section 2.1 is concerned with the basic notions of linear algebra, leading up to the subject of low-rank approximation, discussed in Section 2.3.

### 2.1 Linear algebra basics

Let us introduce our notation. Scalars are typeset in lowercase italic Roman or Greek letters ( $a, x, \beta, \dots$ ). Vectors and matrices are denoted with bold letters, where lowercase bold italic Roman and Greek letters ( $\mathbf{u}, \mathbf{v}, \omega, \dots$ ) indicate vectors and uppercase bold Roman or Greek letters ( $\mathbf{A}, \mathbf{B}, \Omega, \dots$ ) stand for matrices. The bold letter  $\mathbf{0}$  will denote matrices and vectors of all zeroes, whose dimensions, unless otherwise specified, will be evident from the context. We will use  $\mathbf{1}$  to denote the vector of all ones and the bold letter  $\mathbf{I}$  to denote the identity matrix, whose dimensions again will be evident from the context in which they appear. The transpose of a matrix is denoted by the superscript  $\top$ , and similarly, the conjugate transpose by the superscript  $*$ . The  $j$ -th basis vector of the Euclidian space is denoted as  $e_j$ , i.e.  $e_1 = [1, 0, 0, \dots]^\top, e_2 = [0, 1, 0, \dots]^\top$ , etc. Finally, we will often use both notations  $A(I, J)$  and  $A_{IJ}$  to denote submatrices corresponding to the index sets  $I$  and  $J$ . Sometimes it will be convenient to apply the MATLAB-style notation  $A(:, j)$ , to denote entire columns of  $A$ . For the sake of convenience, we may specify the index sets  $I$  in standard mathematical notation, i.e.  $I = \{i_1, i_2, \dots\}$ , but actually mean an index vector  $[i_1, i_2, \dots]^\top$ , so that the indices appear in the intended order.

#### Rank and nullity

We begin with some basic definitions related to matrices. Matrices generalize linear maps  $T : V \rightarrow W$  over finite vector spaces  $V, W$ . As such, it is convenient to introduce the notion of

**Definition 2.1.1 (Rank and Nullity)** *Let  $V, W$  be finite dimensional vector spaces and let  $T : V \rightarrow W$  be a linear transform. We define*

$$\text{rank } T = \dim \text{range } T \tag{2.1}$$

and

$$\text{nullity } T = \dim \text{null } T \tag{2.2}$$

With these notions in place, we proceed to the fundamental *rank-nullity theorem*:

2.1	Linear algebra basics . . . . .	8
	Rank and nullity . . . . .	8
	Important properties . . . . .	9
	Singular value decomposition	10
	Other matrix factorizations .	12
2.2	Sparse matrices . . . . .	12
2.3	Low-rank matrices . . . . .	14
2.4	Rank-revealing QR . . . . .	15
2.5	Random sampling . . . . .	16
	Range approximation . . . . .	16
	Rank estimation via sampling	17
	Randomized ID . . . . .	18

**Theorem 2.1.1** (Rank-nullity theorem) Let  $V, W$  be finite vector spaces and let  $T : V \rightarrow W$  be a linear transform. Then

$$\text{rank } T + \text{nullity } T = \dim V. \quad (2.3)$$

This is useful as it offers insight into how we might decompose the linear transform  $T$ . Imagine for instance a matrix  $A$  and the linear transform induced by  $A$ , which maps  $v \in V$  to  $Av \in W$ . Theorem 2.1.1 tells us that we can identify linear subspaces of  $V$  containing all vectors that do not get mapped to  $\mathbf{0}$  by  $A$ . This becomes important in the context of low-rank approximation and Theorem 2.1.3.

## Important properties of matrices

We introduce some important definitions related to matrices. The goal here is to give an overview and to fix notation. For more details, as well as proofs for the theorems we refer the reader to [1, 11–13].

**Definition 2.1.2** (Spectral norm) Let  $A \in \mathbb{C}^{m \times n}$ . We define the spectral norm

$$\|A\|_2 = \max_{x \in \mathbb{C}^n} \frac{\|Ax\|_2}{\|x\|_2}. \quad (2.4)$$

In general, if we do not specify a subscript and write  $\|\cdot\|$ , we mean the spectral or Euclidian norm.

**Definition 2.1.3** (Inner product) The inner product for matrices is

$$\langle A, B \rangle = \text{tr } AB^*, \quad (2.5)$$

where  $A, B \in \mathbb{C}^{m \times n}$ .

The inner-product induces the inner-product norm, which for matrices is the Frobenius norm.

**Definition 2.1.4** (Frobenius norm) The Frobenius norm is defined as

$$\|A\|_F^2 = \text{tr } A^* A = \text{tr } AA^* = \sum_{i,j} |A_{ij}|^2. \quad (2.6)$$

An important class of matrices are unitary/orthogonal matrices.

**Definition 2.1.5** (Unitary/orthogonal matrices) A matrix  $U \in \mathbb{C}^{n \times n}$  is called unitary iff  $U^* U = UU^* = I$ . Similarly, a matrix  $Q \in \mathbb{R}^{n \times n}$  is called orthogonal iff  $Q^\top Q = QQ^\top = I$ .

**Definition 2.1.6** (Unitarily invariant norm) We say a norm  $\|\cdot\|$  is unitarily invariant, if  $\|UA\| = \|A\|$  holds for any unitary matrix  $U$  and any  $A$ .

Both the Frobenius norm  $\|\cdot\|_F$  and the spectral norm  $\|\cdot\|_2$  are unitarily invariant.

Our main goal is to find solutions to the linear system (1.22). We introduce the notion of invertible matrices, which are the main focus of this work.

**Definition 2.1.7 (Invertible matrix)** Let  $A \in \mathbb{C}^{n \times n}$ . Then  $A$  is called invertible if there exists a matrix  $A^{-1} \in \mathbb{C}^{n \times n}$ , such that  $AA^{-1} = A^{-1}A = I$ .

A square matrix is invertible, iff it has full rank.

A useful indicator, that is related to the inverse of a matrix is the spectral condition number.<sup>1</sup>

**Definition 2.1.8 (Spectral condition number)** Given an invertible matrix  $A$ , the condition number is defined as

$$\kappa(A) = \|A^{-1}\|_2 \|A\|_2. \quad (2.7)$$

For the purpose of analyzing matrices, we will often use the spectral decomposition.

**Definition 2.1.9 (Diagonalizable matrix)** Let  $A \in \mathbb{C}^{n \times n}$  be a square matrix. Then,  $A$  is called diagonalizable, if there exists an invertible matrix  $P \in \mathbb{C}^{n \times n}$  and a diagonal matrix  $\Lambda = \text{diag}(\lambda_1, \lambda_2, \dots, \lambda_n) \in \mathbb{C}^{n \times n}$  such that

$$A = P\Lambda P^{-1}. \quad (2.8)$$

(2.8) is also the spectral- or eigendecomposition of  $A$ . The column vectors of  $P$  are then called the eigenvectors of  $A$ . Similarly,  $\lambda_i$  are called eigenvalues and the set  $\Lambda = \{\lambda_i\}_{i=1}^n$  is called the spectrum of  $A$ .

If a matrix is diagonalizable by a unitary matrix  $P$ , we call  $A$  unitarily diagonalizable.

**Definition 2.1.10 (Normal matrices)** The matrix  $A$  is called normal iff  $A^*A = AA^*$ .

**Proposition 2.1.2** A matrix  $A$  is normal iff there exists a unitary matrix  $U$  and a diagonal matrix  $\Lambda$ , such that  $A = U\Lambda U^*$ .

## Singular value decomposition

As we have mentioned earlier, we may want to decompose the matrix  $A$  and identify the subspace of  $V$ , which does not get mapped to  $0$ . This is achieved by the *singular value decomposition* (SVD), which has many useful applications beyond identifying the range and null space of a matrix:

**Theorem 2.1.3 (Singular Value Decomposition)** Let  $A \in \mathbb{C}^{m \times n}$  and  $m \geq n$ . Then there exist unitary matrices  $U \in \mathbb{C}^{m \times m}$  and  $V \in \mathbb{C}^{n \times n}$  such that

$$A = U\Sigma V^*, \text{ with } \Sigma = \begin{bmatrix} \sigma_1 & & \\ & \ddots & \\ & & \sigma_n \end{bmatrix} \in \mathbb{R}^{m \times n}, \quad (2.9)$$

where  $\sigma_1 \geq \sigma_2 \geq \dots \geq \sigma_n \geq 0$ .

1: The symbol  $\kappa$  is used interchangably for both the condition number of a matrix and the wavenumber. To distinguish between them, we write  $\kappa(\cdot)$  as a function of a matrix, whenever we refer to the condition number.

The diagonal entries  $\sigma_i = \Sigma_{ii}$  are called the singular values of  $A$ . If they are ordered as in Theorem 2.1.3, then  $\Sigma$  is uniquely defined. The column vectors of  $U$  and  $V$  are called the left- and right-singular vectors of  $A$ . In the case of a real matrix  $A \in \mathbb{R}^{m \times n}$ ,  $U$  and  $V$  are real-valued as well.

**Theorem 2.1.4**  $\|A\|_2 = \sigma_1$  and  $\|A\|_F = \sqrt{\sigma_1^2 + \sigma_2^2 + \dots + \sigma_n^2}$ .

**Theorem 2.1.5** The non-zero singular values of  $A$  are the square roots of the non-zero eigenvalues of  $AA^*$  and  $A^*A$ .

An important way of understanding the singular value decomposition is to understand it as a sum of rank-1 matrices

$$A = \sum_{j=1}^n \sigma_j u_j v_j^*, \quad (2.10)$$

where  $u_j$  and  $v_j$  are the columns of  $U$  and  $V$  respectively. We can identify the range and null space of  $A$  using the singular value decomposition:

**Theorem 2.1.6** The rank of  $A$  is the number of non-zero singular values.

**Theorem 2.1.7**  $\text{range } A = \text{span}\{u_1, u_2, \dots, u_k\}$  and  $\text{null } A = \text{span}\{v_{k+1}, v_{k+2}, \dots, v_n\}$ , where  $k = \text{rank } A$ .

An important Theorem connecting the singular values to the inner product, is Von Neumann's trace inequality:

**Theorem 2.1.8** (Von Neumann's trace inequality) For  $A, B \in \mathbb{R}^{m \times n}$  with  $m \geq n$  with singular values  $\sigma_1(A) \geq \sigma_2(A) \geq \dots \geq \sigma_n(A)$ , and  $\sigma_1(B) \geq \sigma_2(B) \geq \dots \geq \sigma_n(B)$ , we have

$$|\langle A, B \rangle| \leq \sigma_1(A)\sigma_1(B) + \sigma_2(A)\sigma_2(B) + \dots + \sigma_n(A)\sigma_n(B). \quad (2.11)$$

An important consequence is

$$\begin{aligned} \|A - B\|_F^2 &= |\langle A - B, A - B \rangle| = \|A\|_F^2 - 2\langle A, B \rangle + \|B\|_F^2 \\ &\geq \sum_{j=1}^n (\sigma_j(A) - \sigma_j(B))^2. \end{aligned} \quad (2.12)$$

**Corollary 2.1.9** (Interlacing property) Let  $A_k$  denote the matrix containing the first  $k$  columns of  $A \in \mathbb{C}^{m \times n}$ . Then

$$\sigma_j(A) \geq \sigma_j(A_k) \geq \sigma_{j+n-k}(A) \quad \text{for } j = 1, 2, \dots, k \quad (2.13)$$

The interlacing property follows from the interlacing properties of the eigenvalues of  $AA^*$  [12].

## Other matrix factorizations

We provide a short overview of some other important matrix factorizations, which will be useful.

**Theorem 2.1.10** (QR factorization) *Let  $A \in \mathbb{C}^{m \times n}$  and  $m \geq n$ . Then, there exists a unitary matrix  $Q \in \mathbb{C}^{m \times m}$ , such that*

$$A = QR \quad \text{with} \quad R = \begin{bmatrix} R_1 \\ \mathbf{0} \end{bmatrix} = \begin{bmatrix} \mathbf{R}_1 \\ \mathbf{0} \end{bmatrix}, \quad (2.14)$$

where  $R$  and  $R_1 \in \mathbb{C}^{n \times n}$  are upper triangular matrices.

From an algorithmic point of view, the QR decomposition is perhaps the most important matrix factorization as it is used just about anywhere. As such, we will be returning to the QR factorization in Section 2.3.

Another notable factorization which is the interpolative decomposition, which represents the original matrix using a subset of the columns of  $A$ :

**Theorem 2.1.11** (Interpolative decomposition) *Let  $A \in \mathbb{C}^{m \times n}$  be a matrix of rank  $k$ . Then, there exists an index set  $J = [j_1, j_2, \dots, j_k]$  and a matrix  $X \in \mathbb{C}^{k \times n}$ , such that*

$$A = A(:, J)X, \quad (2.15)$$

where  $X$  satisfies  $X(:, J) = I$  and  $\forall i, j : |X(i, j)| \leq 1$ .

The interpolative decomposition proves useful for computing low-rank representations of matrices using columns of the original matrix  $A$ . In practice, the selection of an optimal set  $J$  of columns is an NP-hard problem and can therefore not be computed in polynomial time. The condition on the bound on entries in  $X$  can be relaxed to 2 rather than 1, which allows such decompositions to be computed in polynomial time using rank-revealing QR factorizations (See Section 2.4).

Finally, we introduce the Schur decomposition:

**Theorem 2.1.12** (Schur decomposition) *Let  $A \in \mathbb{C}^{n \times n}$  be a square matrix. Then, there exists a unitary matrix  $Q \in \mathbb{C}^{n \times n}$ , such that*

$$A = QUQ^*, \quad (2.16)$$

where  $U \in \mathbb{C}^{n \times n}$  is an upper triangular matrix.

We have not yet specified algorithms to compute these factorizations. Introductions to numerical algorithms for the computation of such factorizations are presented in [1, 12]. An overview of the computational complexity of these algorithms is found in Table 2.1.

## 2.2 Sparse matrices

A powerful concept when dealing with large matrices is sparsity. A great amount of relevant problems in science and engineering have a high degree of sparsity.<sup>3</sup>

**Table 2.1:** Computational complexity of common matrix factorizations using dense matrices  $A \in \mathbb{C}^{m \times n}$  with  $m > n$  or  $m = n$ .

operation	complexity
$A = U\Sigma V^*$	$\mathcal{O}(mn^2)$
$A = P\Lambda P^{-1}$	$\mathcal{O}(n^3)$ <sup>2</sup>
$A = QR$	$\mathcal{O}(mn^2)$
$A = LU$	$\mathcal{O}(n^3)$
$A = QLUQ^*$	$\mathcal{O}(n^3)$

2: Due to the equivalence of finding the eigenvalues of a matrix and polynomial rootfinding, there cannot be an algorithm that will compute the spectral decomposition to arbitrary precision in a fixed number of steps for matrices of order  $n \leq 5$ . The iterative algorithms referred to, converge to the desired accuracy quickly such that the stated accuracy is observed in practice [1, pp. 194].

3: Sparsity can be defined as the number of zero entries in relation to the total number of entries in the matrix:

$$1 - \frac{\text{nnz } A}{mn}.$$

**Definition 2.2.1** (Sparse matrices) et  $A \in \mathbb{C}^{m \times n}$ . We call  $A$  a sparse matrix if  $A_{ij} = 0$  for most entries  $(i, j) \in \{1, \dots, m\} \times \{1, \dots, n\}$ . The collection of all non-zero indices  $\{(i, j) \in s.t. A_{ij} \neq 0\}$  is called the sparsity structure or sparsity pattern of  $A$ .  $\text{nnz } A$  denotes the total number of non-zero entries in  $A$ .

The naive way of storing a sparse matrix comes in the form of a coordinate list (COO format), in which we store the sparsity pattern as a list of indices  $(i, j)$  and a list for the entry values  $A_{ij}$ . It is easy to see that this format requires only  $\mathcal{O}(\text{nnz } A)$  storage to store the matrix. Similarly, computing a matrix-vector product  $x \rightarrow Ax$  only requires us to sum over the contributions of each non-zero entry and therefore costs only  $\mathcal{O}(\text{nnz } A)$  flops.

Other operations, however, have become considerably more difficult. One such operation is the extraction of a submatrix  $A(I, J)$ , where  $I = \{i_1, i_2, \dots, i_k\}$  and  $J = \{j_1, j_2, \dots, j_l\}$  are some index sets. This requires us to go over each entry in  $A$  and compare it to the indices in  $I$  and  $J$  which makes the complexity of accessing a submatrix  $\mathcal{O}(n_{\text{nz}} k l)$ .<sup>4</sup> Even if we were to organize  $I$  and  $J$  into binary-search trees (BST) [14], this operation would still require  $\mathcal{O}(n_{\text{nz}} k \log k l \log l)$  operations. This is an important point here, as many of the algorithms for sparse matrices involve search algorithms, as there is no a-priori way of knowing where to find the relevant entries.

A powerful concept in search algorithms is to keep datastructures sorted at all times, so that accessing them becomes highly efficient [14]. This idea is effectively realized in the compressed sparse column (CSC) and compressed sparse row (CSR) formats. Figure 2.1 illustrates the representation of a sparse matrix in CSC format. This representation further compresses the storage by storing a list of pointers `colptr`, which indicate where the information for each column is stored. The row indices and (non-zero) values are stored in the arrays `rowind` and `values`. If we were to access column  $j$ , we can directly “jump” to the relevant portion of `rowind` and `values`. This is done by retrieving the pointers `colptr[j]` and `colptr[j+1]`, which tell us which section of `rowind` and `values` contain the information belonging to column  $j$ . This improves performance for many algorithms, as it significantly reduces the number of entries over which we have to search.

colptr:	1	6	8	10	10	12	...					
rowind:	1	3	4	8	9	2	4	1	3	4	8	...
values:	$A_{11}$	$A_{31}$	$A_{41}$	$A_{81}$	$A_{91}$	$A_{22}$	$A_{42}$	$A_{13}$	$A_{33}$	$A_{45}$	$A_{85}$	...

4: When speaking about the complexity of algorithms for sparse matrices, it is often convenient to consider the maximum number of non-zero entries per column, which we denote with  $n_{\text{nz}}$ .

We leave the discussion of sparse matrices at that, as many of the algorithms either rely on the specific structure of  $A$  or the datastructure used to store it. Chapter 3 discusses methods for computing solutions of linear systems, where the matrix is sparse.

**Figure 2.1:** Illustration of sparse matrix representation in compressed sparse column (CSC) format.

## 2.3 Low-rank matrices

Unfortunately, not all problems allow for sparse representations of their matrices. However, in some cases, we may exploit that the matrix  $A \in \mathbb{C}^{m \times n}$  has a low rank  $k$  and therefore admits a representation

$$A = UV^*, \quad (2.17)$$

where the matrices  $U \in \mathbb{C}^{m \times k}$  and  $V \in \mathbb{C}^{n \times k}$  are called the generators of  $A$ . This representation follows directly from (2.10) and has many algorithmic advantages, i.e. the storage cost is  $\mathcal{O}(k(m + n))$ , as opposed to  $\mathcal{O}(mn)$  if the matrix is stored as a dense matrix. Depending on the rank  $k$ , this has the potential to yield significant speed-up in common arithmetic operations such as matrix-vector multiplications. Table 2.2 provides an overview of the computational cost of common arithmetic operations using this representation.

Thus, the question arises in which cases such a representation exists and how it may be computed. We introduce the notion of numerical rank.

**Definition 2.3.1** (Numerical rank) *For a given tolerance  $\epsilon \in \mathbb{R}_{>0}$ , the numerical rank of  $A$  is the smallest integer  $k_\epsilon$ , which permits a matrix  $\tilde{A}$  of rank  $k_\epsilon$  to approximate  $A$ , such that*

$$\|A - \tilde{A}\| \leq \epsilon. \quad (2.18)$$

The class of matrices that satisfy (2.18) are much more relevant in practice than matrices that are actually of low-rank up to machine precision.<sup>5</sup> Let us assume that we have somehow obtained a matrix  $A$ , which we know to be of low numerical rank. In this case how should we obtain the representation (2.17)? Theorem 2.3.1 provides the answer to this question.

**Definition 2.3.2** (Rank- $k$  truncation) *Consider the singular value decomposition (2.9) of a matrix  $A$ . For a positive integer  $k \leq n$ , we call*

$$A_k = U_k \Sigma_k V_k^* \quad (2.19)$$

*the rank- $k$  truncation of  $A$ , with  $U_k = [u_1, u_2, \dots, u_k]$ ,  $V_k = [v_1, v_2, \dots, v_k]$  and  $\Sigma_k = \text{diag}(\sigma_1, \sigma_2, \dots, \sigma_k)$ .*

**Theorem 2.3.1** (Eckart-Young-Mirsky) *For any unitarily invariant norm  $\|\cdot\|$ , the best rank- $k$  approximation of a matrix  $A \in \mathbb{C}^{m \times n}$  is its rank- $k$  truncation:*

$$\|A - A_k\| = \min_{\substack{\tilde{A} \in \mathbb{C}^{m \times n} \\ \text{rank}(\tilde{A})=k}} \|A - \tilde{A}\|. \quad (2.20)$$

The proof for the Frobenius norm  $\|\cdot\|_F$  follows by inserting the truncation into Equation 2.12. Similar proofs can be constructed for the general case by bounding the norm  $\|A - \tilde{A}\|$  from below [13].

**Table 2.2:** Computational complexity of common operations using low-rank matrices.  $A$  and  $B$  are  $n \times n$  low-rank matrices of rank  $k$  and  $x$  is a  $n$ -dimensional vector.

operation	complexity
$x \rightarrow Ax$	$\mathcal{O}(kn)$
$B \rightarrow A + B$	$\mathcal{O}(kn)$
$B \rightarrow AB$	$\mathcal{O}(kn)$

5: In most cases we will denote  $k_\epsilon$  with  $k$  to simplify notation.



**Figure 2.2:** Low-rank approximation of the EPFL logo with a rank of  $k = 10$ . Some smearing occurs around the letter "P", but otherwise the low-rank approximation does a good job of approximating the original picture.

## 2.4 Rank-revealing QR factorization

The question then arises, which algorithm should be used to compute low-rank representations of the form (2.17). The SVD will clearly give us the best answer. However, it is expensive to compute and we may wish to terminate the computation early if the numerical rank  $k$  is much smaller than the dimensions of the matrix.

Let us revisit the QR-factorization. However this time, we consider the pivoted version

$$A\Pi = QR = [Q_1 \ Q_2] \begin{bmatrix} R_{11} & R_{12} \\ \mathbf{0} & R_{22} \end{bmatrix}, \quad (2.21)$$

where  $\Pi$  denotes a permutation matrix for the columns of  $A$ . We have chosen to partition  $Q$  and  $R$  to expose the first  $k$  columns of  $Q$  and the corresponding upper-triangular block  $R_{11}$  of order  $k$ .

Now, let us imagine that  $A$  has low numerical rank, such that  $\sigma_k(A) \gg \sigma_{k+1}(A)$ . After computing the factorization (2.21) and detecting a significant drop from  $\sigma_{\min}(R_{11})$  to  $\sigma_{\max}(R_{22})$ , one wonders whether  $k$  is the numerical rank we seek.<sup>6</sup> By the interlacing property for singular values, we recover

$$\sigma_{\min}(R_{11}) \leq \sigma_k(A) \quad (2.22)$$

and

$$\sigma_{\max}(R_{22}) \geq \sigma_{k+1}(A). \quad (2.23)$$

The first inequality tells us that for a general permutation  $\Pi$ , we may do a bad job at separating the row space of  $A$  from its nullspace. Moreover, considering the second inequality, we may overestimate the rank if we base our rank-estimation on  $\sigma_{\max}(R_{22})$ .

**Definition 2.4.1** (Rank-revealing QR factorization) *Let  $A\Pi = QR$  be the QR factorization of  $A\Pi$  as in (2.21) with*

$$R = \begin{bmatrix} R_{11} & R_{12} \\ \mathbf{0} & R_{22} \end{bmatrix},$$

*where  $R_{11}$  is the upper triangular block of order  $k$ . We call the QR factorization a rank-revealing QR (RRQR) factorization if it satisfies*

$$\sigma_{\min}(R_{11}) \geq \frac{\sigma_k(A)}{p(k, n)}, \quad (2.24a)$$

$$\sigma_{\max}(R_{22}) \leq \sigma_{k+1}(A)p(k, n), \quad (2.24b)$$

*where  $p(k, n)$  is a function that can be bounded by a low-order polynomial in  $k$  and  $n$ .*

In other words, this property guarantees that if  $\sigma_k(A) \gg \sigma_{k+1}(A)$ , it will be reflected in  $\sigma_{\min}(R_{11})$  and  $\sigma_{\max}(R_{22})$  [15].<sup>7</sup>

Algorithms that compute rank-revealing QR approximations have their use in rank-estimation, subspace selection and other relevant problems. In the context of low-rank approximation, they are not only useful for revealing the numerical rank of  $A$  but also to compute a low-rank representation of the form (2.17).<sup>8</sup> After computing a rank-revealing

6:  $\sigma_{\min}(A)$  and  $\sigma_{\max}(A)$  denote the minimum and maximum singular values of  $A$  respectively.

7: The terms “rank-revealing” is usually used to refer to algorithms that produce QR factorizations that satisfy (2.24).

8: One of the main practical issues with algorithms for low-rank approximations is that the ranks of matrices tend to increase with each operation. For instance, if we add two matrices  $C = A + B$ , we have  $\text{rank}(C) \leq \text{rank}(A) + \text{rank}(B)$ . In the worst case, the resulting rank of  $C$  may be fairly large. However, in practice it may be closer to  $C \approx \max(\text{rank}(A), \text{rank}(B))$ . This is, however, not reflected by the computed representation. As a consequence, one has to frequently use recompression.

QR factorization  $A\Pi = QR$ , we approximate

$$A\Pi \approx \tilde{A}\Pi = Q_1 [R_{11} \quad R_{12}], \quad (2.25)$$

which is a factorization of the form (2.17) with approximation error  $\|A - \tilde{A}\| = \|R_{22}\|$ . By the properties of the rank-revealing QR factorization, we know that the error is bounded by the best approximation error times the function  $p(k, n)$ , i.e.  $\|R_{22}\|_2 \leq \sigma_{k+1}(A)p(k, n)$ .

There are various algorithms for computing rank-revealing QR factorizations [15, 16]. Most of these algorithms share the core algorithms of either using Householder reflectors or Givens rotations to compute the factorization [12]. The main difference lies in the strategy for pivoting and computing  $\Pi$ . Due to the rank-revealing property, these algorithms allow to be terminated early. This yields a typical cost of  $\mathcal{O}(kmn)$  operations and a worst-case complexity of  $\mathcal{O}(mn^2)$  operations, where  $k$  is the rank at which the algorithm halts.

## 2.5 Random sampling

One of the main problems with the methods presented so far is their computational cost. While the arithmetic with low-rank matrices is basically linear, (assuming constant rank  $k$ ), the compression into low-rank format is not. In many applications, we may be presented with a matrix  $A$  without having direct access to its entries. Instead we may have routines to compute matrix-vector products  $x \rightarrow Ax$  and  $x \rightarrow A^*x$  efficiently. Perhaps we also know that  $A$  is of low numerical rank and therefore admits a low-rank representation (2.17). The question arises of how we can efficiently compute such representations.

### Range approximation

Randomized methods provide a solution to this problem [17]. The core idea is to use random-sampling to extract the dominant column space  $Q$  of  $A$ , such that

$$\|A - QQ^*A\| \leq \epsilon \quad (2.26)$$

for some tolerance  $\epsilon$  and some norm  $\|\cdot\|$ . This is done by forming the sample matrix  $S = A\Omega$ , where  $\Omega$  is a  $n \times k + p$  standard Gaussian matrix, and  $p$  is a small positive integer.  $p$  is called the oversampling parameter and is used to control the quality of the approximation.

**Definition 2.5.1** (Standard random Gaussian matrix) We call  $\Omega \in \mathbb{R}^{m \times n}$  a random Gaussian matrix if each entry  $\Omega_{ij}$  is drawn independently from the same standard Gaussian distribution  $\mathcal{N}(0, 1)$ . The columns of  $\Omega$  are called standard Gaussian vectors.

Then, if the singular values of  $A$  decay rapidly,  $S$  will be a good approximation to the dominant column space of  $A$ , and we can extract it by forming a QR factorization of  $S$ .

We consider the simple case where  $A$  has exact rank  $k$  that is known a priori. A simple algorithm, that will help us understand randomized

methods is presented in Algorithm 2.1. One may ask whether  $\mathbf{Q}$  reliably

```

procedure RANDOM RANGE FINDER( $A, k$ )
    Draw an  $n \times k + p$  Gaussian random matrix  $\Omega$ 
    Sample  $A$  by forming  $S = A\Omega$ 
    Form the QR factorization  $S = QR$ 
    return  $Q$ 
end procedure

```

provides a good approximation of the column space. This is indeed the case, as it can be shown that if  $\mathbf{Q}$  is computed using Algorithm 2.1 with target rank  $k \geq 2$  and oversampling parameter  $p \geq 2$ , such that  $k + p \leq \min\{m, n\}$ , then

$$\|A - \mathbf{Q}\mathbf{Q}^*A\| \leq (1 + 9\sqrt{k + p}\sqrt{\min\{m, n\}})\sigma_{k+1}(A) \quad (2.27)$$

is satisfied with a probability of at least  $1 - 3p^{-p}$  under mild assumptions on  $p$  [17].

## Rank estimation via sampling

So far, we have assumed that  $A$  is exactly of rank  $k$  and that it is known a priori. In practice, this is rarely the case and we would instead seek to extract a column space such that (2.26) is met. To do so, we adapt Algorithm 2.1 to automatically detect the numerical rank of  $A$ .

**Lemma 2.5.1** Let  $B \in \mathbb{R}^{m \times n}$  with  $m \geq n$  and  $r$  a positive integer, as well as  $\alpha \in \mathbb{R}_{>0}$ . Then let  $\{\omega_1, \omega_2, \dots, \omega_r\}$  denote a collection of independently drawn standard Gaussian vectors. Then

$$\|B\| \leq \alpha \sqrt{\frac{2}{\pi}} \max_{i=1,2,\dots,r} \|B\omega_i\| \quad (2.28)$$

with a probability of at least  $1 - \alpha^{-r}$ .

*Proof.* Because standard Gaussian vectors are invariant with respect to rotation, without loss of generality, we can assume  $B = \text{diag}(\sigma_1, \sigma_2, \dots, \sigma_n)$ , where  $\sigma_i = \sigma_i(B)$  denote the singular values of  $B$ . As a consequence, for a single standard Gaussian vector  $\omega$ , we have

$$\begin{aligned} \mathbb{P}[\alpha\|\mathbf{B}\omega\| \leq \|\mathbf{B}\|] &= \mathbb{P}[\alpha^2\|\mathbf{B}\omega\|^2 \leq \|\mathbf{B}\|^2] = \mathbb{P}\left[\alpha^2 \sum_{i=1}^n \sigma_i^2 \omega_i^2 \leq \sigma_1^2\right] \\ &\leq \mathbb{P}[\alpha^2 \omega_1^2 \leq 1] = \frac{1}{\sqrt{2\pi}} \int_{-\alpha/\sqrt{2}}^{\alpha/\sqrt{2}} \exp\left(-\frac{t^2}{2}\right) dt \\ &\leq \sqrt{\frac{2}{\pi}} \alpha^{-1}. \end{aligned}$$

Then, if we repeat this experiment  $r$  times, we find that the probability of (2.28) is bounded from below by  $1 - \alpha^{-r}$ , which concludes the proof.  $\square$

Lemma 2.5.1 offers insight into when our approximation satisfies a certain tolerance with a high probability. On the other hand, it does not tell us

**Algorithm 2.1:** Randomized range finder for a matrix  $A \in m \times n$  and a fixed rank  $k$ .  $p$  is the oversampling parameter [17].

how accurate this estimate may be. Moreover, there is a good chance that it will overestimate the rank as it is not a sharp estimate and aims to bound the norm from above with high probability. This is illustrated in Figure 2.3, where we depict the estimated ranks of 100 random rank-16 matrices.

Let us again consider the matrix  $B \in \mathbb{R}^{m \times n}$  with singular value decomposition  $B = U\Sigma V^*$  and standard Gaussian matrix  $\Omega \in \mathbb{R}^{n \times r}$  with columns  $\omega_i$ . Then, the estimator

$$\frac{1}{r} \|B\Omega\|_F^2 = \frac{1}{r} \sum_{i=1}^r \omega_i^*(B^*B)\omega_i = \sum_{i=1}^r \tilde{\omega}_i^*(\Sigma^*\Sigma)\tilde{\omega}_i \quad (2.29)$$

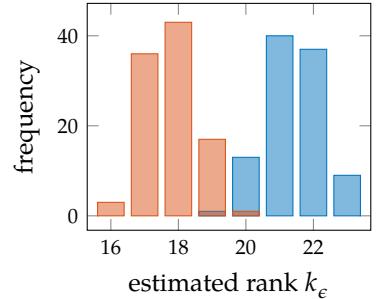
is an unbiased estimator of the Frobenius norm [18]. Here  $\tilde{\omega}_i = V\omega_i$  denote rotated standard random vectors, which are also standard Gaussian vectors. We can verify that the expected value is indeed the square of the Frobenius norm of  $B$ :

$$\mathbb{E} \left[ \frac{1}{r} \|B\Omega\|_F^2 \right] = \|B\|_F^2. \quad (2.30)$$

With these estimates, we can return to the random range finder (Algorithm 2.1) and modify it to adaptively choose the rank  $k$ , based on a supplied tolerance (Algorithm 2.2).

```

procedure ADAPTIVE RANGE FINDER( $A, \epsilon, r$ )
    Set  $k \leftarrow 0$ 
    do
        Set  $k \leftarrow k + k_{\text{step}}$ 
        Draw a  $n \times k$  Gaussian random matrix  $\Omega_1$ 
        Draw a  $n \times r$  Gaussian random matrix  $\Omega_2$ 
        Sample  $A$  by forming  $[S_1 S_2] = A [\Omega_1 \quad \Omega_2]$ 
        Form the QR factorization  $S_1 = QR$ 
        Form  $S_2 \leftarrow S_2 - QQ^*S_2$  to compute  $(A - QQ^*A)\Omega_2$ 
        Use  $S_2$  to compute the estimator of choice est
    while est  $> \epsilon$ 
    return  $Q, k$ 
end procedure
```



**Figure 2.3:** Comparison of rank estimators on 100 samples of random matrices of order 100 with  $\sigma_i = 2^{-i}$ , for  $\epsilon = 2^{-16}$ . The red histogram depicts the frequency of ranks estimated using the Frobenius norm estimator (2.29). The blue histogram depicts the frequency of ranks estimated using the spectral norm estimator (2.28)

**Algorithm 2.2:** Adaptive range finder. The estimator **est** can be selected to estimate the norm in either the spectral or the Frobenius norm.

## Randomized interpolative decomposition

The interpolative decomposition (2.15) has a useful property in the context of random sampling. If we apply the interpolative decomposition to  $S^* = S^*(:, I)X$  we observe that

$$A\Omega = S = X^*S(I,:) = X^*A(I,:)\Omega. \quad (2.31)$$

In other words, we obtain the matrix  $X^*$  and the index set  $I$ , which are also a valid interpolative decomposition of the original matrix  $A$ . We can compute the interpolative decomposition by using a pivoted QR-decomposition  $S^*\Pi = QR$ , which yields

$$S^* = Q [R_1 \quad R_2] \Pi^* = QR_1 \underbrace{[I \quad R_1^{-1}R_2]}_{=X} \Pi = S^*(:, I)X. \quad (2.32)$$

The question arises, of how good such a representation might be in relation to the approximation  $QQ^*A$ , where  $Q$  is the unitary column space extracted from the sample matrix  $S$ .<sup>9</sup> Thus, assuming  $\text{span}(Q) = \text{span}(S)$  yields

$$\begin{aligned} S &= QQ^*S = X^*S(I,:) = X^*Q(I,: )Q^*S \\ \iff Q &= X^*Q(I,: ). \end{aligned}$$

Using this identity, we have

$$\begin{aligned} \|A - X^*A(I,: )\| &\leq \|A - X^*Q(I,: )Q^*A\| + \|X^*Q(I,: )Q^*A - X^*A(I,: )\| \\ &\leq \|A - QQ^*A\| + \|X\|\|Q(I,: )Q^*A - A(I,: )\| \\ &\leq (1 + \|X\|)\|A - QQ^*A\|. \end{aligned}$$

If we fix  $\|\cdot\|$  to be the Frobenius norm and assume that  $S$  is a  $m \times k$  matrix (i.e.  $p = 0$ , no oversampling), we have

$$\|A - X^*A(I,: )\|_F \leq (1 + \sqrt{1 + 4k(m - k)})\|A - QQ^*A\|_F, \quad (2.33)$$

where we have used the properties of  $X$  to bound its norm [17].

<sup>9</sup>: The unitary matrix  $Q$  does not denote the matrix from (2.32), but rather the matrix computed using Algorithm 2.1.

# Sparse direct solvers

Our main focus lies on solving the linear system

$$Ax = b, \quad (1.22)$$

where  $A \in \mathbb{R}^{n \times n}$  is an invertible sparse matrix and  $b \in \mathbb{R}^n$  is a right-hand side. This chapter is dedicated to direct methods, which are usually, in some form or another, equivalent to Gaussian elimination. Gauss did not have computers at his disposal and so we need to adapt Gaussian elimination to the setting of modern-day computing. To do so, we exploit the sparsity and structure of  $A$  to construct methods adapted to this specific problem.

## 3.1 Graph elimination

The most well-known algorithm in computational linear algebra is likely to be Gaussian elimination. Applying Gaussian elimination to sparse linear systems will prove educational in our quest to better understand the challenges of efficiently solving (1.22).

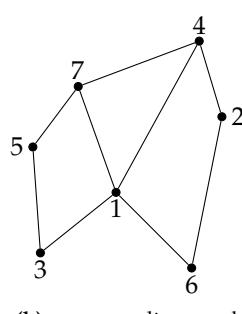
A sparse matrix  $A$  can be understood as the connectivity matrix of some weighted graph  $\mathcal{G}(A)$ , where two nodes  $i, j \in \mathcal{G}$  are connected iff  $A(i, j) \neq 0$ .

**Definition 3.1.1** *The adjacency graph  $\mathcal{G}(A)$  of a matrix  $A$  of dimensions  $n \times n$  is a graph  $(V, E)$  such that  $V$  is a set of  $n$  vertices, where vertex  $i$  is associated with the  $i$ -th degree of freedom. There is an edge  $(i, j) \in E$  iff  $A(i, j) \neq 0$  and  $i \neq j$ .*

The value  $A(i, j)$  is then associated with the weight of the corresponding edge. We ignore these values for now and concentrate on the sparsity pattern of  $A$ . Figure 3.1 depicts a sparse matrix with its corresponding graph. We observe that for every entry in the matrix, there is an edge in the graph as we expected. As  $A$  has a symmetric sparsity pattern,  $\mathcal{G}$  can be represented by an undirected graph, as every entry  $A(i, j)$ , implies the existence of the reverse edge  $A(j, i)$ .

	1	2	3	4	5	6	7
1	$\times$		$\times$	$\times$		$\times$	$\times$
2		$\times$		$\times$			$\times$
3	$\times$		$\times$		$\times$		
4	$\times$	$\times$		$\times$			$\times$
5		$\times$		$\times$		$\times$	
6	$\times$	$\times$			$\times$		
7	$\times$		$\times$	$\times$			$\times$

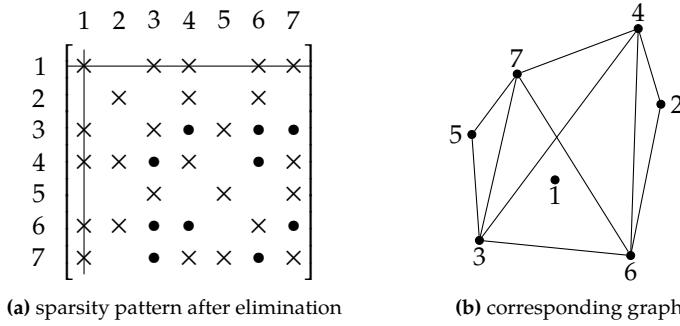
(a) sparsity pattern of  $A$



(b) corresponding graph

3.1 Graph elimination . . . . .	20
3.2 LDR Factorization . . . . .	21
3.3 Fill-in and reorderings . . . . .	23
Bandwidth reduction . . . . .	23
Nested dissection . . . . .	25
3.4 Structured elimination . . . . .	26

**Figure 3.1:** We can understand sparse matrices as the connectivity matrix of its adjacency graph. Here the matrix is symmetric, so the graph is not directed.



**Figure 3.2:** The elimination of degrees of freedom can be understood as a graph elimination procedure. The elimination of node 1 connects nodes 3, 4, 6 and 7 with each other. This leads to fill-in, as seen on the right.

We can now identify Gaussian elimination with the elimination of graph nodes in  $\mathcal{G}$ . To illustrate this, let us perform one step of Gaussian elimination to the matrix  $A$ . The elimination of the first degree of freedom is depicted in Figure 3.2. As node 1 is connected to nodes 3, 4, 6 and 7, we will add a multiple of the first row to these rows. This will eliminate the first entries in those rows, thus decoupling node 1 from its neighbors 3, 4, 6 and 7. However, this also adds new entries to these rows, which are depicted as black dots in Figure 3.2. This effect is called fill-in and is one of the major challenges for the development of sparse direct solvers.

Let us now have a look at the graph of the newly formed matrix. Figure 3.2 depicts the situation after the first elimination step. We see that node 1 is now decoupled from the remaining graph. On the other hand, the fill-in created new edges, connecting nodes 3, 4, 6 and 7 with each other. Therefore, we can regard Gaussian elimination for sparse matrices as a graph elimination algorithm, where nodes are subsequently isolated from the graph. Whenever we eliminate a node, we have to connect other nodes, previously connected to the eliminated node, with each other. With this example, we begin to understand the problem of fill-in on a graphical level. At each elimination step, we potentially create many new edges, depending on how many nodes were connected to the eliminated node.

## 3.2 LDR Factorization

Let us formalize the Gaussian elimination procedure. The elimination of the first degree of freedom can be written as

$$\begin{aligned} A &= \begin{bmatrix} a_{11} & a_{12} \\ a_{21} & A_{22} \end{bmatrix} \\ &= \begin{bmatrix} 1 & 0 \\ a_{21}/a_{11} & I \end{bmatrix} \begin{bmatrix} a_{11} & 0 \\ 0 & A_{22} - a_{21}a_{11}^{-1}a_{12} \end{bmatrix} \begin{bmatrix} 1 & a_{12}/a_{11} \\ 0 & I \end{bmatrix} = \tilde{L}^{(1)} \tilde{A}^{(1)} \tilde{R}^{(1)}, \end{aligned}$$

where  $I$  are identity matrices of corresponding size. Instead of eliminating nodes individually, we can group them into so-called *supernodes* and

eliminate entire blocks. The above elimination step then takes the form

$$\begin{aligned} \mathbf{A} &= \begin{bmatrix} \mathbf{A}_{11} & \mathbf{A}_{12} \\ \mathbf{A}_{21} & \mathbf{A}_{22} \end{bmatrix} \\ &= \underbrace{\begin{bmatrix} \mathbf{I} & \mathbf{0} \\ \mathbf{A}_{21}\mathbf{A}_{11}^{-1} & \mathbf{I} \end{bmatrix}}_{=\tilde{\mathbf{L}}^{(1)}} \underbrace{\begin{bmatrix} \mathbf{D}^{(1)} & \mathbf{0} \\ \mathbf{0} & \mathbf{S}^{(1)} \end{bmatrix}}_{=\tilde{\mathbf{A}}^{(1)}} \underbrace{\begin{bmatrix} \mathbf{I} & \mathbf{A}_{11}^{-1}\mathbf{A}_{12} \\ \mathbf{0} & \mathbf{I} \end{bmatrix}}_{=\tilde{\mathbf{R}}^{(1)}} \end{aligned} \quad (3.1)$$

with  $\mathbf{D}^{(1)} = \mathbf{A}_{11}$  and  $\mathbf{S}^{(1)} = \mathbf{A}_{22} - \mathbf{A}_{21}\mathbf{A}_{11}^{-1}\mathbf{A}_{12}$ , where  $\mathbf{S}^{(1)}$  is called the Schur complement. The matrices  $\tilde{\mathbf{L}}^{(1)}$  and  $\tilde{\mathbf{R}}^{(1)}$  are called the left and right transforms of the factorization and are lower and upper-triangular matrices. Due to their special structure, they can be trivially inverted by changing the sign of the off-diagonal block

$$\begin{bmatrix} \mathbf{I} & \mathbf{0} \\ \mathbf{A}_{21}\mathbf{A}_{11}^{-1} & \mathbf{I} \end{bmatrix}^{-1} = \begin{bmatrix} \mathbf{I} & \mathbf{0} \\ -\mathbf{A}_{21}\mathbf{A}_{11}^{-1} & \mathbf{I} \end{bmatrix}.$$

We proceed the factorization recursively by applying (3.1) to the Schur complements  $\mathbf{S}^{(i)}$ . At the  $i$ -th iteration, the matrix  $\tilde{\mathbf{A}}^{(i)}$  has the form

$$\tilde{\mathbf{A}}^{(i)} = \begin{bmatrix} \mathbf{D}^{(i)} & \mathbf{0} \\ \mathbf{0} & \mathbf{S}^{(i)} \end{bmatrix}.$$

Factoring the Schur complement as

$$\mathbf{S}^{(i)} = \tilde{\mathbf{L}}^{(i+1)} \begin{bmatrix} \mathbf{S}_{11}^{(i)} & \mathbf{0} \\ \mathbf{0} & \mathbf{S}^{(i+1)} \end{bmatrix} \tilde{\mathbf{R}}^{(i+1)}$$

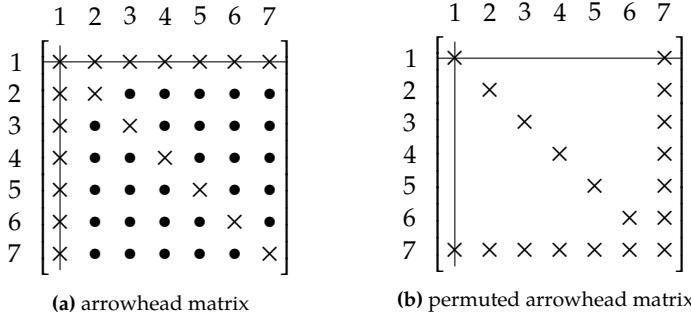
yields the intermediate matrix  $\tilde{\mathbf{A}}^{(i+1)}$  for the next iteration:

$$\begin{aligned} \tilde{\mathbf{A}}^{(i)} &= \mathbf{L}^{(i+1)} \tilde{\mathbf{A}}^{(i+1)} \mathbf{R}^{(i+1)} \\ &= \begin{bmatrix} \mathbf{I} & \mathbf{0} \\ \mathbf{0} & \tilde{\mathbf{L}}^{(i+1)} \end{bmatrix} \left[ \begin{array}{c|c} \mathbf{D}^{(i)} & \mathbf{0} \\ \hline \mathbf{0} & \begin{array}{c} \mathbf{S}_{11}^{(i)} & \mathbf{0} \\ \mathbf{0} & \mathbf{S}^{(i+1)} \end{array} \end{array} \right] \begin{bmatrix} \mathbf{I} & \mathbf{0} \\ \mathbf{0} & \tilde{\mathbf{R}}^{(i+1)} \end{bmatrix}. \end{aligned}$$

At the next iteration, the diagonal block includes  $\mathbf{D}^{(i+1)} = \text{diag}(\mathbf{D}^{(i)}, \mathbf{S}_{11}^{(i)})$  and we repeat the procedure on  $\mathbf{S}^{(i+1)}$ . Starting with  $\tilde{\mathbf{A}}^{(i)} = \mathbf{A}$  and proceeding repeatedly for  $r$  steps yields the desired factorization

$$\mathbf{A} = \mathbf{L}^{(1)} \mathbf{L}^{(2)} \dots \mathbf{L}^{(r)} \tilde{\mathbf{A}}^{(r)} \mathbf{R}^{(r)} \dots \mathbf{R}^{(2)} \mathbf{R}^{(1)}, \quad (3.2)$$

where  $\tilde{\mathbf{A}}^{(r)}$  is either diagonal or block-diagonal, depending on whether we chose to eliminate individual nodes or supernodes. Evaluating the product  $\mathbf{L}^{(1)} \mathbf{L}^{(2)} \dots \mathbf{L}^{(r)}$  yields a lower-triangular matrix and similarly for the right transforms. As such, we have succeeded in computing a factorization of the form  $\mathbf{A} = \mathbf{LDR}$ , where  $\mathbf{D}$  is block-diagonal and  $\mathbf{L}, \mathbf{R}$  are lower and upper-triangular. Computationally, it is more efficiently to store the  $\mathbf{L}$  and  $\mathbf{R}$  implicitly, i.e., individually and only as off-diagonal blocks, as this makes the application cheaper and reduces the storage requirements.



**Figure 3.3:** Elimination of the first node in the arrowhead matrix results in excessive fill-in. If we instead permute the first and last node, we can avoid it altogether.

### 3.3 Fill-in and reorderings

After seeing the graph elimination algorithm in Section 3.1, one might suspect that the order in which nodes are eliminated plays a big role in how much fill-in is created. Figure 3.3 illustrates an extreme case, where the elimination of the first node leads to a dense remainder matrix. In this example, fill-in can be avoided altogether by simply permuting node 1 and node 7. This structure is often referred to as an arrowhead pattern. Another, less extreme example of fill-in is shown in Figure 3.4, and depicts the sparsity pattern of the left transform  $L$  of the Galerkin matrix associated with Figure 1.3.

It is often beneficial to seek permutation matrices  $\Pi_I$  and  $\Pi_J$  such that the fill-in is minimized when  $\Pi_I A \Pi_J$  is factored.<sup>1</sup> This however, is a NP-hard combinatorial problem [14] as can be seen by considering the Graph elimination problem discussed earlier in Section 3.1. As such, we have to resort to heuristics for finding a suitable permutations  $\Pi_I$  and  $\Pi_J$ , which reduce (but not necessarily minimize) the fill-in. This has been a field of intense study and there exist a plethora of strategies for finding such permutations [14, 19–21]. We discuss two of the most popular strategies, which are bandwidth-reduction and nested dissection.

#### Bandwidth reduction

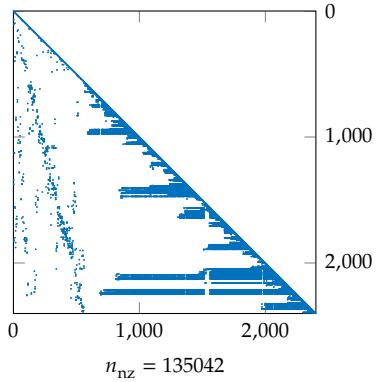
One intuitive way of reducing fill-in is to concentrate the entries as close to the diagonal as possible to get a banded matrix. This problem is related to the graph bandwidth problem, in which we arrange our nodes along a line and permute them to minimize the longest edge [14].

**Definition 3.3.1** (Bandwidth of a matrix) *Let  $A \in \mathbb{C}^{m \times n}$  be a matrix and  $I = \{1, 2, \dots, m\}$ ,  $J = \{1, 2, \dots, n\}$  its index sets. We call the smallest positive integer  $w$ , for which*

$$\forall i \in I, j \in J : |i - j| > w \implies A(i, j) = 0 \quad (3.3)$$

*the bandwidth of  $A$ .*

Let us establish that fill-in can only occur within the bandwidth of the matrix.



**Figure 3.4:** Sparsity pattern of the  $L$  factor in  $A = LR$  for the Helmholtz problem depicted in Figure 1.3.

1: One might also wish to permute  $A$  to ensure the stability of the Gaussian elimination [19].

In the context of graph elimination, so-called *triangulated graphs* are particularly interesting. A graph is triangulated if any cycle bigger than 3 has a chord, that is, an edge joining non-consecutive vertices within the cycle. It can be shown that these graphs allow for an elimination reordering without fill-in [22].

**Proposition 3.3.1** Let  $A \in \mathbb{C}^{n \times n}$  be a matrix with bandwidth  $w$  and let  $A = LR$ , where  $L$  and  $R$  are lower and upper triangular matrices. Then,  $L$  and  $R$  have a bandwidth no bigger than  $w$ .

*Proof.* We can partition  $A = LR$

$$A = \begin{bmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{bmatrix} = \underbrace{\begin{bmatrix} L_{11} & L_{12} \\ L_{21} & L_{22} \end{bmatrix}}_{=L} \underbrace{\begin{bmatrix} R_{11} & R_{12} \\ R_{21} & R_{22} \end{bmatrix}}_{=R},$$

such that  $L$  and  $R$  have a compatible block structure and  $A_{21} = L_{21}R_{11} + L_{22}R_{21}$ . Choose any integer  $i$  such that  $w+1 \leq i \leq n$  and let  $A_{21}$  be the bottom left zero block of dimensions  $(n-i+1) \times (i-w-1)$ . Taking  $L_{21}$  to have the same dimension as  $A_{21}$  yields

$$\mathbf{0} = A_{21} = L_{21}R_{11} + L_{22}R_{21} = L_{21}R_{11}$$

as  $R_{21} = \mathbf{0}$  is below the diagonal of  $R$ .  $R_{11}$  is the  $(i-w-1) \times (i-w-1)$  top left triangular block and depends on  $A_{11}$ . As the condition must be fulfilled for arbitrary  $A_{11}$ , we conclude that  $L_{21} = \mathbf{0}$  and therefore  $L$  has bandwidth  $w$ . A similar argument can be made for  $R$ , which concludes the proof.  $\square$

Thus, minimizing the bandwidth is likely to decrease the fill-in, although there is no guarantee. There exist a number of algorithms for the reduction of the bandwidth of a sparse matrix. The most prominent entries are Reverse Cuthill-McKee (RCM) and minimum degree reordering [20, 23].

We introduce the Cuthill-McKee algorithm, which constructs permutation  $\Pi$  (as an ordered tuple) based on the sparsity pattern  $\mathcal{G}(A)$ .

**Definition 3.3.2** Let  $A \in \mathbb{C}^{n \times n}$  and  $I = \{1, 2, \dots, n\}$ . Then, the adjacency of a set  $J \subseteq I$  on  $A$  is defined as

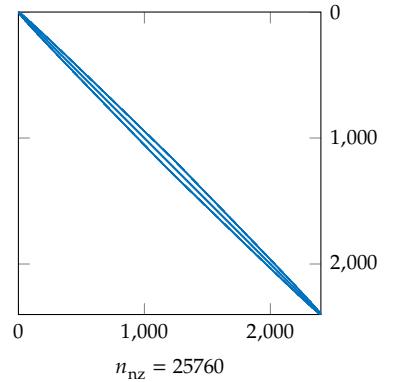
$$\text{adj}_A(J) = \{i \in I : \exists j \in J \text{ s.t. } A(i, j) \neq 0\}. \quad (3.4)$$

```

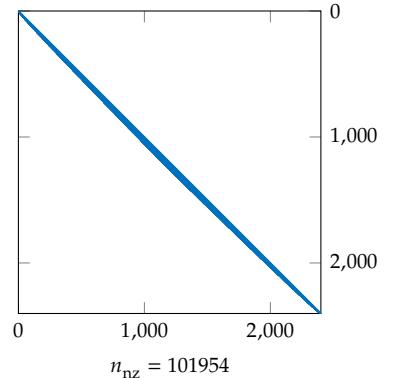
procedure REVERSE CUTHILL-MCKEE( $A$ )
  Set  $i = 0$ 
  Select a node  $v$  with a minimum number of neighbors
  Set  $\Pi = \Pi_0 = \{v\}$ 
  while  $|\Pi| < n$  do
    Construct the set of adjacent nodes  $\Pi_{i+1} = \text{adj}_A(\Pi_i) \setminus \Pi$ 
    Sort  $\Pi_{i+1}$  by earliest occurrence of a neighbor in  $\Pi$ 
    Append  $\Pi_{i+1}$  to  $\Pi$ 
     $i \leftarrow i + 1$ 
  end while
  Reverse  $\Pi$ 
  return  $\Pi$ 
end procedure

```

Figure 3.5 depicts the sparsity pattern of the Galerkin matrix (Figure 1.3) after reordering it with the reverse Cuthill-McKee heuristic. We can see that RCM succeeds in reducing the bandwidth and therefore the fill-in



(a) sparsity pattern of  $\Pi_I A \Pi_J$



(b) sparsity pattern of  $L$

**Figure 3.5:** Sparsity plots of reordered Galerkin matrix and associated  $L$  factor after RCM reordering.

**Algorithm 3.1:** Reverse Cuthill-McKee reordering. Constructs a bandwidth-reducing reordering  $\Pi$  based on the sparsity pattern of  $A$ .

in the  $L$  factor. Compared to the unordered version in Figure 3.4, it has lost about a quarter of the fill-in.

## Nested dissection

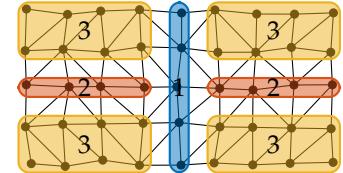
Let us now pursue a different approach. The adjacency graph of a  $p = 1$  finite element matrix is the same as the mesh on which it was constructed. As a consequence, the adjacency graphs of finite element matrices often look like the graph depicted in Figure 3.6a.

The core idea of nested dissection is to recursively find separating sets of indices, which split the adjacency graph into smaller subgraphs. In Figure 3.6a, we choose the separator 1, which splits the mesh and the graph, into the left and right subgraphs. We move the associated degrees of freedom to the end of the matrix, as illustrated in Figure 3.6b.

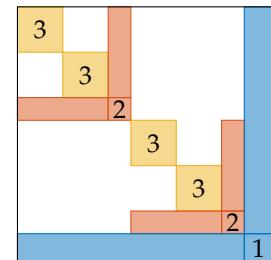
One way of storing the permutation is as a simple sorted tuple of the index set  $I$ . However, the nested dissection has more structure as it is hierarchically organized like a tree. We will see later that this can be exploited for parallel and distributed computations. As such, it is preferable to store the reordering in the form of a tree.

**Definition 3.3.3 (Elimination tree)** Let  $A$  be a sparse matrix of order  $n$  and  $I = \{1, 2, \dots, n\}$  its index set. Let  $\mathcal{E}$  be a tree, where each node  $\mu \in \mathcal{E}$  is associated with an index set  $I^\mu \subset I$ . Then,  $\mathcal{E}$  is called an elimination tree of  $A$  iff

- all index sets are disjoint:  $\forall \mu \neq \nu : I^\mu \cap I^\nu = \emptyset$
- the union of all index sets is the entire set,  $\bigcup_{\mu \in \mathcal{E}} I^\mu = I$
- For any two indices  $i \in I^\mu, j \in I^\nu$  with  $\mu > \nu$ ,  $A(i, j) \neq 0$  or  $A(j, i) \neq 0$  implies that  $\nu$  is a descendant of  $\mu$ . In other words, the node  $\nu$  is included in the subtree rooted at node  $\mu$ .

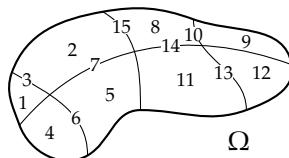


(a) nested dissection on mesh

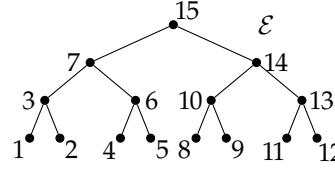


(b) corresponding permutation of the matrix

Figure 3.6: Illustration of a simple nested dissection on a finite element mesh.



(a) nested dissection of  $\Omega$



(b) corresponding elimination tree

The last property of Definition 3.3.3 guarantees that all fill-in for the node  $I^\mu$  has already been summed up, when it is being eliminated. Figure 3.7 depicts a nested dissection and its corresponding elimination tree on the domain  $\Omega$ .<sup>2</sup>

Algorithm 3.2 summarizes the nested dissection algorithm for computing an elimination tree  $\mathcal{E}$  of the degrees of freedom  $I$ , based on the adjacency graph  $\mathcal{G}(A)$  of the matrix. So far, we have exploited the equivalence of the finite element mesh and the adjacency graph, which allows us to use geometric information when choosing the separators. More precisely, we have used geometric bisection to compute it. We distinguish between geometric and algebraic nested dissection, depending on how the separators are chosen. In the latter case, only the adjacency information of  $\mathcal{G}(A)$  is used to compute such a reordering. For geometrically

Figure 3.7: Illustration of a nested dissection of the computational domain  $\Omega$  using separators. The figure on the right depicts the corresponding post-ordered tree data-structure  $\mathcal{E}$ , which induces an elimination order.

2: We call the nodes of the elimination tree supernodes.

```

procedure NESTED DISSECTION( $\mathcal{G}$ )
    Find an index set  $I \subset \text{vertices}(\mathcal{G})$  which splits  $\mathcal{G}$  into  $I$  and the
    disconnected graphs  $\mathcal{G}_1$  and  $\mathcal{G}_2$ 
    Call  $\mathcal{E}_1 \leftarrow \text{NESTED DISSECTION}(\mathcal{G}_1)$ 
    Call  $\mathcal{E}_2 \leftarrow \text{NESTED DISSECTION}(\mathcal{G}_2)$ 
    Create treenode  $\mathcal{E}$  containing  $I$ 
    Append  $\mathcal{E}_1$  and  $\mathcal{E}_2$  as children of  $\mathcal{E}$ 
    return  $\mathcal{E}$ 
end procedure

```

**Algorithm 3.2:** General nested dissection algorithm

chosen separator on meshes such as the one in Figure 3.6a, [21] proves that the nested dissection algorithm will yield a reordering which minimizes the fill-in. Figure 3.8 shows the reordered Galerkin matrix (Figure 1.3), where geometric nested dissection has been used to compute the permutation.

### 3.4 Structured elimination

Let us return to sparse Gaussian elimination. However this time, using an elimination tree  $\mathcal{E}$  to guide us through the elimination process. We assume that the elimination tree is a binary tree as this will considerably simplify our discussion. The generalization to more general elimination trees is straightforward albeit involved. We assume we are at node  $\sigma$  in the elimination tree, which has children nodes  $\mu$  and  $\nu$  as illustrated in Figure 3.9.

In contrast to the block elimination process in Section 3.2, we do not have to keep the entire Schur complement in memory to compute the factorization of node  $\sigma$ . Instead, we only have to consider  $I^\sigma$  and the adjacency of  $I^\sigma$ , which has not yet been eliminated. Let  $\text{desc}(\sigma)$  denote the set of all descendants of  $\sigma$  in  $\mathcal{E}$  and, similarly, let  $\text{anc}(\sigma)$  denote the set of all ancestors of  $\sigma$ . Then, we define the boundary of  $\sigma$  as the index set

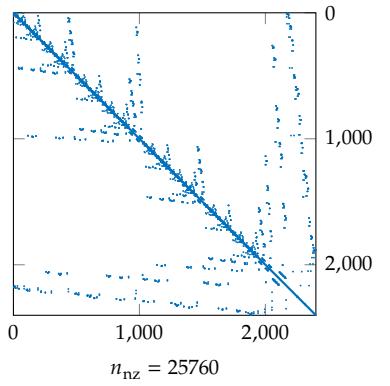
$$\begin{aligned} B^\sigma &= \left\{ i \in \bigcup_{\iota \in \text{anc}(\sigma)} I^\iota : \exists j \in I^\sigma \text{ s.t. } A(i, j) \neq 0 \text{ or } A(j, i) \neq 0 \right\} \\ &= \text{adj}_A(I^\sigma) \setminus \bigcup_{\iota \in \text{desc}(\sigma)} I^\iota. \end{aligned} \quad (3.5)$$

We consider the matrix  $\tilde{A}^{(\sigma)}$ , which is the result of eliminating all indices in  $\text{desc}(\sigma)$ . The short-hand notation  $\tilde{A}_{ib}^{(\sigma)} = \tilde{A}(I^\sigma, B^\sigma)$  is used to denote relevant submatrices of the big matrix  $\tilde{A}^{(\sigma)}$ . The relevant portion of  $\tilde{A}^{(\sigma)}$  is

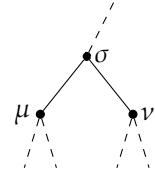
$$\hat{A}^{(\sigma)} = \begin{bmatrix} \tilde{A}_{ii}^{(\sigma)} & \tilde{A}_{ib}^{(\sigma)} \\ \tilde{A}_{bi}^{(\sigma)} & \tilde{A}_{bb}^{(\sigma)} \end{bmatrix} = \begin{bmatrix} I & 0 \\ \hat{L}^{(\sigma)} & I \end{bmatrix} \begin{bmatrix} \hat{D}^{(\sigma)} & 0 \\ 0 & \hat{S}^{(\sigma)} \end{bmatrix} \begin{bmatrix} I & \hat{R}^{(\sigma)} \\ 0 & I \end{bmatrix}. \quad (3.6)$$

Here, we have factored  $\hat{A}^{(\sigma)}$  by introducing the diagonal block for elimination

$$\hat{D}^{(\sigma)} = \tilde{A}_{ii}^{(\sigma)}, \quad (3.7)$$



**Figure 3.8:** Sparsity pattern of the Galerkin matrix after nested dissection reordering.



**Figure 3.9:** Elimination of the supernode  $\sigma$ . Contributions from the elimination of its children  $\mu$  and  $\nu$  have to be accounted for.

the left and right factors

$$\hat{L}^{(\sigma)} = \tilde{A}_{bi}^{(\sigma)} (\tilde{A}_{ii}^{(\sigma)})^{-1}, \quad (3.8a)$$

$$\hat{R}^{(\sigma)} = (\tilde{A}_{ii}^{(\sigma)})^{-1} \tilde{A}_{ib}^{(\sigma)}, \quad (3.8b)$$

and the Schur complement

$$\hat{S}^{(\sigma)} = \tilde{A}_{bb}^{(\sigma)} - \tilde{A}_{bi}^{(\sigma)} (\tilde{A}_{ii}^{(\sigma)})^{-1} \tilde{A}_{ib}^{(\sigma)}. \quad (3.9)$$

An important distinction that we make here is that matrices with a “ $\wedge$ ” denote intermediate matrices of the factorization. These are *local* and small. In other words, these matrices do not have the size of the overall matrix or the intermediate matrices denoted with a “ $\sim$ ”, which are “big” matrices. However, these matrices are associated with some indices of the big matrices  $A$  and  $\tilde{A}$ . To simplify our discussion, we will use *global* indices associated with the big matrix  $A$  to index the smaller matrices denoted with a “ $\wedge$ ”. This allows to handle these smaller matrices while keeping in mind where they belong in the overall scheme.

**Proposition 3.4.1** *To fully represent the overall factorization  $A = LDR$ , it is sufficient to store the small matrices  $\hat{D}^{(\sigma)}$ ,  $\hat{L}^{(\sigma)}$ ,  $\hat{R}^{(\sigma)}$ , as well as their index sets for each node  $\sigma$  in  $\mathcal{E}$ .*

One way of seeing this is to see how the left transform  $\tilde{L}^{(\sigma)}$  can be constructed from the corresponding small matrix  $\hat{L}^{(\sigma)}$ . For any index  $(i, j) \in B^\sigma \times I^\sigma$ , we have  $\tilde{L}^{(\sigma)}(B^\sigma, I^\sigma) = \hat{L}^{(\sigma)}$  and for  $(i, j) \notin B^\sigma \times I^\sigma$ ,  $\tilde{L}^{(\sigma)}(i, j) = \delta(i, j)$ . We recall from Section 3.2 that the left and right transforms can be inverted by changing the sign of the block below the diagonal. Consequently, we can apply the inverse of the factorization as in Algorithm 3.3, which confirms Proposition 3.4.1. It is worth noting that at the top level, due to the definition of the elimination tree, the boundary will be the empty set. Therefore, at the top level it suffices to invert the remaining diagonal block  $\hat{D}^{(\sigma)}$ .<sup>3</sup>

```

for all nodes  $\sigma \in \mathcal{E}$  from the bottom up do
    Apply  $\mathbf{b}(B^\sigma) \leftarrow -\hat{L}^{(\sigma)} \mathbf{b}(I^\sigma) + \mathbf{b}(B^\sigma)$ 
end for
for all nodes  $\sigma \in \mathcal{E}$  do
    Apply  $\mathbf{b}(I^\sigma) \leftarrow (\hat{D}^{(\sigma)})^{-1} \mathbf{b}(I^\sigma)$ 
end for
for all nodes  $\sigma \in \mathcal{E}$  from the top down do
    Apply  $\mathbf{b}(I^\sigma) \leftarrow \mathbf{b}(I^\sigma) - \hat{R}^{(\sigma)} \mathbf{b}(B^\sigma)$ 
end for
return  $\mathbf{b}$ 

```

<sup>3</sup>: In some cases, we will instead split the index set  $I$  at the root into some boundary degrees of freedom and interior degrees of freedom. In these cases, we will have to apply the inverse of the resulting top-level Schur complement.

**Algorithm 3.3:** Apply the inverse  $A^{-1} = R^{-1} D^{-1} L^{-1}$  to a vector  $\mathbf{b}$ .

We still need an operation to assemble the intermediate matrix  $\tilde{A}^{(\sigma)}$ , after factoring  $\tilde{A}^{(\mu)}$  and  $\tilde{A}^{(\nu)}$ . To do so, we introduce the update matrix

$$\hat{U}^{(\sigma)} = -\tilde{A}_{bi}^{(\sigma)} (\tilde{A}_{ii}^{(\sigma)})^{-1} \tilde{A}_{ib}^{(\sigma)}, \quad (3.10)$$

which is added to the corresponding indices of  $\tilde{A}^{(\sigma)}$  to form the Schur complement (3.9). More importantly,  $\hat{U}^{(\sigma)}$  contains all of the updates

of its children nodes, which is a consequence of the properties of the elimination tree  $\mathcal{E}$ . To form  $\hat{\mathbf{A}}^{(\sigma)}$ , we have to sum the contributions of  $\hat{\mathbf{U}}^{(\mu)}$  and  $\hat{\mathbf{U}}^{(\nu)}$  to the right places in  $A^{(\sigma)}$  and extract the right submatrix. This can be formalized by introducing the *extend-add* operator.

**Definition 3.4.1** (extend-add) *Let  $A$  be the matrix to be factored and  $I = \{1, 2, \dots, n\}$  its index set. Then, let  $\hat{\mathbf{B}}$  and  $\hat{\mathbf{C}}$  be matrices associated to the global index sets  $I_{\hat{\mathbf{B}}} \subseteq I$  and  $I_{\hat{\mathbf{C}}} \subseteq I$  respectively. We define the index sets  $I_1 = I_{\hat{\mathbf{B}}} \cap I_{\hat{\mathbf{C}}}$ ,  $I_2 = I_{\hat{\mathbf{B}}} \setminus I_{\hat{\mathbf{C}}}$  and  $I_3 = I_{\hat{\mathbf{C}}} \setminus I_{\hat{\mathbf{B}}}$ . Up to a permutation, the extend-add operation is then defined as*

$$\hat{\mathbf{B}} \oplus \hat{\mathbf{C}} = \begin{bmatrix} \hat{\mathbf{B}}(I_1, I_1) + \hat{\mathbf{C}}(I_1, I_1) & \hat{\mathbf{B}}(I_1, I_2) & \hat{\mathbf{C}}(I_1, I_3) \\ \hat{\mathbf{B}}(I_2, I_1) & \hat{\mathbf{B}}(I_2, I_2) & \mathbf{0} \\ \hat{\mathbf{C}}(I_3, I_1) & \mathbf{0} & \hat{\mathbf{C}}(I_3, I_3) \end{bmatrix}. \quad (3.11)$$

Using the short-hand notation  $A_{ib}^{(\sigma)} = A(I^\sigma, B^\sigma)$  from before, we can now write

$$\hat{\mathbf{A}}^{(\sigma)} = \begin{bmatrix} \tilde{A}_{ii}^{(\sigma)} & \tilde{A}_{ib}^{(\sigma)} \\ \tilde{A}_{bi}^{(\sigma)} & \tilde{A}_{bb}^{(\sigma)} \end{bmatrix} \oplus \hat{\mathbf{U}}^{(\mu)} \oplus \hat{\mathbf{U}}^{(\nu)}. \quad (3.12)$$

Instead of forming the Schur complement  $\hat{\mathbf{S}}^{(\sigma)}$ , we can factor

$$\hat{\mathbf{F}}^{(\sigma)} = \begin{bmatrix} \tilde{A}_{ii}^{(\sigma)} & \tilde{A}_{ib}^{(\sigma)} \\ \tilde{A}_{bi}^{(\sigma)} & \mathbf{0} \end{bmatrix} \oplus \hat{\mathbf{U}}^{(\mu)} \oplus \hat{\mathbf{U}}^{(\nu)}, \quad (3.13)$$

which directly yields the update matrix  $\hat{\mathbf{U}}^{(\sigma)}$  in place of the Schur complement. The matrix  $\hat{\mathbf{F}}^{(\sigma)}$  is called a frontal matrix. Sparse direct solvers, which organize the Gaussian elimination as described are also called frontal solvers [24]. These methods organize the elimination into a “front”, hence the name.

We wrap up the discussion of structured, sparse direct solvers by considering the computational work required to form the factorization. Let us consider the nested dissection of a  $d$ -dimensional domain. In relation to the overall size  $n$ , the top level separator has a size of  $\mathcal{O}(n^{\frac{d-1}{d}})$  for  $d \geq 2$ . Consequently, the size of matrices to factor at each level  $l$  behaves as

$$n_l \sim 2^{-\frac{d-1}{d}(l-1)} n^{\frac{d-1}{d}}. \quad (3.14)$$

In other words, the size of the separators is divided by  $2^{d-1}$  every  $d$  levels [25]. Assuming that the cost of forming the intermediate matrices is negligible, the work to be performed at each node is of order  $\mathcal{O}(n_l^3)$ . Summing over all levels  $l = 1, 2, \dots, L$  yields the total amount of work

$$\begin{aligned} W &\sim \sum_{l=1}^L 2^{l-1} n_l^3 \sim \sum_{l=1}^L \left(2^{-\frac{2d-3}{d}}\right)^{l-1} n^{3\frac{d-1}{d}} = n^{3\frac{d-1}{d}} \frac{1 - 2^{-\frac{2d-3}{d}L}}{1 - 2^{-\frac{2d-3}{d}}} \\ &\implies W \lesssim n^{3\frac{d-1}{d}}. \end{aligned} \quad (3.15)$$

We conclude that in two dimensions, the cost of structured elimination scales as  $\mathcal{O}(n^{\frac{3}{2}})$  while in three dimensions the scaling is  $\mathcal{O}(n^2)$ .

# 4

## Iterative solvers

In the previous chapter we focused on direct solvers that compute the solution to the linear system (1.22) in one step. The downside of such methods is their  $\mathcal{O}(n^2)$  cost in operation count and storage requirement. Moreover, in many cases we can compute matrix-vector products  $x \rightarrow Ax$  in  $\mathcal{O}(n)$  operations without needing to form the matrix. We might be tempted to exploit this and seek to solve the linear system only using matrix-vector multiplications. This gives rise to iterative methods. In this chapter we focus on Krylov subspace methods and GMRES in particular, as this is the focus of the subsequent work. For a much more in-depth treatment, we refer the reader to the excellent treatment of this topic in [1, 11].

4.1 Krylov spaces . . . . .	29
4.2 The Arnoldi iteration . . . . .	29
4.3 GMRES . . . . .	31
4.4 Convergence of GMRES . . . . .	32
4.5 Preconditioning . . . . .	34

### 4.1 Krylov spaces

Krylov subspace methods are closely related to the subspace- and power iteration [11]. These algorithms attempt to compute good approximations to the dominant  $k$  eigenvectors of  $A$ , by repeatedly applying  $A$  to the same vector.

**Definition 4.1.1** (Krylov subspace) *Let  $A \in \mathbb{C}^{n \times n}$  and  $b \in \mathbb{C}^n$ . Then,*

$$\mathcal{K}_k(A, b) = \text{span} \left\{ b, Ab, \dots, A^{k-1}b \right\} \quad (4.1)$$

*is called the  $k$ -th Krylov subspace of  $A$  and  $b$ .*

As a consequence, we can write any vector  $x \in \mathcal{K}_k(A, b)$  as a linear combination of powers of  $A$  times  $b$ :

$$x = c_0b + c_1Ab + c_2A^2b + \dots + c_{k-1}A^{k-1}b = p(A)b \quad (4.2)$$

In other words,  $x$  is a polynomial  $p(z) = c_0 + c_1z + c_2z^2 + \dots + c_{k-1}z^{k-1}$  in  $A$  times  $b$ .

### 4.2 The Arnoldi iteration

Let us consider that we would like to compute a basis for the  $k$ -th Krylov subspace  $\mathcal{K}_k(A, x_0)$  of a given matrix  $A$  and starting vector  $x_0$ . One might imagine that repeatedly multiplying  $A$  to  $x_0$  might indeed not be the best way to go about it, as we would expect vectors to quickly converge to the eigenvector of the dominant eigenvalue, as in the power-iteration. Therefore, some form of orthonormalization is needed for the method to instead converge to the  $k$ -dimensional subspace spanned by the first  $k$  dominant eigenvectors. This is the core idea of the Arnoldi method, which we will introduce in this section.

A key piece in all of this is the following Lemma, which gives us a hint of how we might construct an orthogonal basis:

**Lemma 4.2.1** Let  $A \in \mathbb{C}^{n \times n}$ ,  $x_0 \in \mathbb{C}^n$ , and let  $\{q_1, q_2, \dots, q_k\}$ , as well as  $\{q_1, q_2, \dots, q_k, q_{k+1}\}$ , be bases of  $\mathcal{K}_k(A, x_0)$  and  $\mathcal{K}_{k+1}(A, x_0)$ , respectively. Then

$$\mathcal{K}_{k+1}(A, x_0) = \text{span} \{q_1, q_2, \dots, q_k, Aq_k\}. \quad (4.3)$$

In other words, once we have an orthonormal set of vectors spanning  $\mathcal{K}_k(A, x_0)$ , we can simply multiply the last basis vector and orthonormalize the result. This is the Arnoldi iteration, which is given in Algorithm 4.1.

```

procedure ARNOLDI ITERATION( $A, x_0$ )
  Set  $q_1 = x_0 / \|x_0\|_2$  and  $Q_1 = q_1$ 
  for  $j = 1, 2, \dots, k$  do
    Compute  $w = Aq_j$ 
    Compute  $h_j = Q_j^* w$ 
    Compute  $\tilde{q}_{j+1} = w - Q_j h_j$ 
    Set  $h_{j+1,j} = \|\tilde{q}_{j+1}\|_2$ 
    Set  $q_{j+1} = \tilde{q}_{j+1} / h_{j+1,j}$ 
    Append  $Q_{j+1} = [Q_j, q_{j+1}]$ 
  end for
end procedure
```

**Algorithm 4.1:** Arnoldi iteration.  
Compute orthonormal basis  $Q_{k+1}$  of  $\mathcal{K}_{k+1}(A, x_0)$

To better analyze the algorithm, let us take a step back and consider a matrix  $A \in \mathbb{C}^{n \times n}$  and its reduction to upper Hessenberg form  $AQ = QH$  where  $Q$  is a unitary matrix and  $H$  a Hessenberg matrix. As  $n$  is typically very large, we are only interested in the first  $k$  columns of  $AQ = QH$ . Let  $Q_k = [q_1, q_2, \dots, q_k]$  contain the first  $k$  columns of  $Q$ . With  $\tilde{H}_k$  denoting the  $(k+1) \times k$  upper-left section of  $H$

$$\tilde{H}_k = \begin{bmatrix} h_{11} & h_{12} & \cdots & h_{1k} \\ h_{21} & h_{22} & \ddots & \vdots \\ \ddots & \ddots & h_{k-1,k} \\ & h_{k,k-1} & h_{k,k} \\ & & h_{k+1,k} \end{bmatrix}, \quad (4.4)$$

we can write the *Arnoldi decomposition*

$$AQ_k = Q_{k+1}\tilde{H}_k, \quad (4.5)$$

and more precisely,

$$\left[ \begin{array}{c} A \\ \vdots \\ A \end{array} \right] \left[ \begin{array}{c} q_1 & \cdots & q_k \end{array} \right] = \left[ \begin{array}{c} q_1 & \cdots & q_{k+1} \end{array} \right] \left[ \begin{array}{ccc} h_{11} & \cdots & h_{1k} \\ h_{21} & & \vdots \\ \ddots & & \vdots \\ & & h_{k+1,k} \end{array} \right].$$

By taking the last column of (4.5), we find the recurrence relation

$$\mathbf{A}\mathbf{q}_k = h_{1k}\mathbf{q}_1 + \cdots + h_{kk}\mathbf{q}_k + h_{k+1,k}\mathbf{q}_{k+1},$$

which connects the new Krylov vector  $\mathbf{q}_{k+1}$  to previous iterates. By rewriting this as

$$\mathbf{A}\mathbf{q}_k = \mathbf{Q}_k\mathbf{h}_k + h_{k+1,k}\mathbf{q}_{k+1} = \mathbf{Q}_k\mathbf{h}_k + \tilde{\mathbf{q}}_{k+1}$$

with  $\mathbf{w} = \mathbf{A}\mathbf{q}_k$  and

$$\mathbf{w} = \mathbf{A}\mathbf{q}_k, \quad \mathbf{h}_k = \mathbf{Q}_k^*\mathbf{w} = \begin{bmatrix} h_{1k} \\ \vdots \\ h_{1k} \end{bmatrix}, \quad \tilde{\mathbf{q}}_{k+1} = \mathbf{w} - \mathbf{Q}_k\mathbf{h}_k, \quad h_{k+1,k} = \|\tilde{\mathbf{q}}_{k+1}\|_2,$$

we recover the Gram-Schmidt process, applied to the Krylov vectors  $\mathbf{w}$ , and therefore Algorithm 4.1.

### 4.3 GMRES

Among the iterative solvers for linear systems, the *generalized minimum residual method* (GMRES) [26] is a popular choice for nonsymmetric problems. The idea of GMRES is a simple one: At step  $k$ , we seek to find a vector  $\mathbf{x}_k$  in the Krylov subspace  $\mathcal{K}_k(\mathbf{A}, \mathbf{r}_0)$ , which minimize the 2-norm of the residual  $\mathbf{r}_k = \mathbf{b} - \mathbf{A}\mathbf{x}_k$ . To this end, we use the Arnoldi method to compute an orthonormal basis  $\mathbf{Q}_k = [\mathbf{q}_1, \mathbf{q}_2, \dots, \mathbf{q}_k]$  which spans the Krylov space.

Thus, we consider the minimization problem

$$\min_{\mathbf{x} \in \mathbf{x}_0 + \mathcal{K}_k(\mathbf{A}, \mathbf{r}_0)} \|\mathbf{b} - \mathbf{A}\mathbf{x}\|_2. \quad (4.6)$$

Let  $\tilde{\mathbf{x}} = \mathbf{x}_0 + \mathbf{Q}_k\mathbf{y} \in \mathbf{x}_0 + \mathcal{K}_k(\mathbf{A}, \mathbf{r}_0)$  with some  $\mathbf{y} \in \mathbb{R}^k$ , and solve the above above minimization problem. We can then write

$$\begin{aligned} \|\mathbf{b} - \mathbf{A}\tilde{\mathbf{x}}\|_2 &= \|\mathbf{r}_0 - \mathbf{A}\mathbf{Q}_k\mathbf{y}\|_2 = \|\mathbf{r}_0 - \mathbf{Q}_{k+1}\tilde{\mathbf{H}}_k\mathbf{y}\|_2 \\ &= \|\mathbf{Q}_{k+1}(\beta_0\mathbf{e}_1 - \tilde{\mathbf{H}}_k\mathbf{y})\|_2 = \|\beta_0\mathbf{e}_1 - \tilde{\mathbf{H}}_k\mathbf{y}\|_2 \end{aligned} \quad (4.7)$$

where we have used Equation 4.5 and  $\beta_0 = \|\mathbf{r}_0\|_2$ . By projecting into the Krylov subspace, we have reduced the size of the minimization problem 4.6 to  $(k+1) \times k$  dimensions:

$$\min_{\mathbf{y} \in \mathbb{R}^k} \|\beta_0\mathbf{e}_1 - \tilde{\mathbf{H}}_k\mathbf{y}\|_2. \quad (4.8)$$

The least-squares problem (4.8) is then solved by

$$\mathbf{y}_k = \beta_0(\tilde{\mathbf{H}}_k^*\tilde{\mathbf{H}}_k)^{-1}\tilde{\mathbf{H}}_k^*\mathbf{e}_1, \quad (4.9)$$

which can be computed using the QR decomposition. The Hessenberg structure of  $\tilde{\mathbf{H}}_k$  can be exploited to form the QR decomposition using  $k$  Givens rotations. The complexity of doing so is  $\mathcal{O}(k^2)$  [11, 26]. In summary, this yields the GMRES Algorithm 4.2, which produces a solution  $\mathbf{x}_k = \mathbf{x}_0 + \mathbf{Q}_k\mathbf{y}_k$  at each iteration of the Arnoldi method 4.1. In

```

procedure GMRES( $A, b, x_0$ )
  Set  $r_0 = b - Ax_0$ ,  $\beta_0 = \|r_0\|_2$ ,  $q_1 = r_0/\beta_0$ ,  $Q_1 = q_1$ 
  for  $j = 1, 2, \dots, k$  do
    Compute  $w = Aq_j$ 
    Compute  $h_j = Q_j^*w$ 
    Compute  $\tilde{q}_{j+1} = w - Q_j h_j$ 
    Set  $h_{j+1,j} = \|\tilde{q}_{j+1}\|_2$ 
    Set  $q_{j+1} = \tilde{q}_{j+1}/h_{j+1,j}$ 
    Append  $Q_{j+1} = [Q_j, q_{j+1}]$ 
    Compute the least-squares solution  $y_j = \beta_0(\tilde{H}_j^*\tilde{H}_j)^{-1}\tilde{H}_j^*e_1$ 
    Compute the error  $\beta_j = \|r_j\|_2 = \|\beta_0e_1 - \tilde{H}_k y_j\|_2$ 
    Set  $x_j = x_0 + Q_j y_j$ 
  end for
  return  $x_k$ 
end procedure

```

**Algorithm 4.2:** GMRES algorithm. Computes an approximate solution  $x_k = x_0 + Q_k y_k$  to the linear system  $Ax = b$ .

practice, we stop the iteration once the error satisfies  $\|r_j\|_2 = \beta_j \leq \epsilon \beta_0$ , where  $\epsilon$  is a user-specified tolerance. One issue with this most straightforward algorithm is that the storage requirement for  $Q_k$  gradually increases with increasing  $k$ . A simple solution to this is to restart 4.2 after  $k$  iterations, keeping the maximum  $k$  low. This is referred to as restarted GMRES.

## 4.4 Convergence of GMRES

In the following, we will show how GMRES is fundamentally linked to a polynomial approximation problem. We consider the  $k$ -th iterate of GMRES  $x_k$ , which we can write as

$$x_k = x_0 + Q_k y_k = x_0 + q_k(A)r_0, \quad (4.10)$$

where  $q_k$  is a polynomial of order  $k - 1$ , with coefficients  $c_0, c_1, \dots, c_{k-1}$ . The coefficients  $c_k = [c_0, c_1, \dots, c_{k-1}]^\top$  can be obtained from the GMRES iterate by

$$c_k = K_k^{-1} Q_k y_k. \quad (4.11)$$

$K_k$  holds the natural basis vectors of the Krylov subspace  $\mathcal{K}_k(A, r_0)$

$$K_k = [r_0, Ar_0, \dots, A^{k-1}r_0]. \quad (4.12)$$

The residual at the  $k$ -th iteration can then be rewritten as

$$\begin{aligned} r_k &= b - Ax_k = b - Ax_0 - Aq_k(A)r_0 \\ &= (I - Aq_k(A))r_0 = p_k(A)r_0, \end{aligned} \quad (4.13)$$

where  $p_k = 1 - zq_k(z)$  is a polynomial of order  $k$ . It is easy to verify that polynomials of this form verify the property  $p_k(0) = 1$ . GMRES minimizes the residual  $r_k$ , by choosing the coefficients of  $p_k$ . This means that GMRES is equivalent to the following polynomial approximation problem:

**Problem 4.4.1** (GMRES polynomial approximation problem) Find a polynomial  $p_k \in \mathcal{P}_k$  of order  $k$  with  $p_k(0) = 1$ , such that

$$\|p_k(\mathbf{A})\mathbf{r}_0\|_2 \quad (4.14)$$

is minimized.

As  $\mathcal{P}_k \subseteq \mathcal{P}_{k+1}$ , it is easy to deduce that the residuals are monotonous and satisfy  $\mathbf{r}_{k+1} \leq \mathbf{r}_k$ . Moreover, in the limit of  $k \rightarrow n$ , we have  $\mathbf{r}_n = \mathbf{0}$  as there are enough terms in  $\mathcal{P}_n$  to interpolate all eigenvalues of the spectrum. This is, unless GMRES breaks down due to a singular  $\mathbf{A}$ . This is of course irrelevant for all practical purposes, as we are interested in solving the system (1.22) in  $k \ll n$  iterations.

As a consequence, we find that unless  $\mathbf{r}_0$  has special properties due to  $\mathbf{b}$ , convergence is determined by

$$\frac{\|\mathbf{r}_k\|_2}{\|\mathbf{r}_0\|_2} \leq \inf_{\substack{p_k \in \mathcal{P}_k \\ p_k(0)=1}} \|p_k(\mathbf{A})\|_2. \quad (4.15)$$

The key question now becomes what kind of properties  $\mathbf{A}$  needs to have to ensure fast convergence of the residuals.

**Theorem 4.4.1** Let  $\mathbf{A}$  be diagonalizable, such that  $\mathbf{V}^{-1}\mathbf{A}\mathbf{V} = \mathbf{\Lambda}$ , with some nonsingular matrix  $\mathbf{V}$  and a diagonal matrix  $\mathbf{\Lambda} = \text{diag}(\lambda_1, \lambda_2, \dots, \lambda_n)$ . Let  $\mathbf{r}_k$  denote the residual after  $k$  iterations of GMRES with the starting vector  $\mathbf{x}_0$ . Then

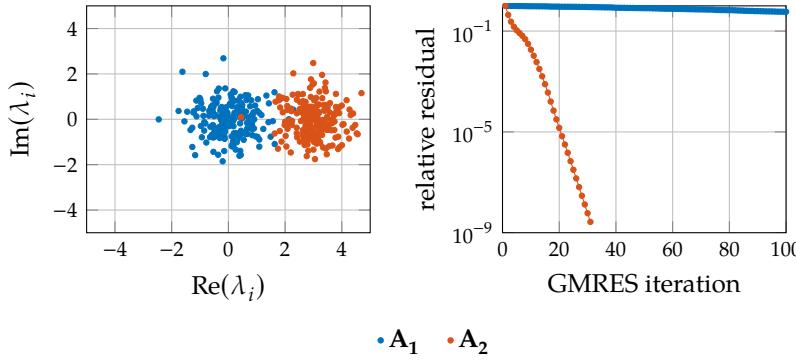
$$\frac{\|\mathbf{r}_k\|}{\|\mathbf{r}_0\|} \leq \kappa(\mathbf{V}) \inf_{\substack{p_k \in \mathcal{P}_k \\ p_k(0)=1}} \sup_{z \in \{\lambda_1, \dots, \lambda_n\}} |p_k(z)|. \quad (4.16)$$

*Proof.* As  $\mathbf{A}$  is diagonalizable, we can write

$$\|p_k(\mathbf{A})\|_2 \leq \|\mathbf{V}\|_2 \|p_k(\mathbf{\Lambda})\|_2 \|\mathbf{V}^{-1}\|_2 = \kappa(\mathbf{V}) \sup_{z \in \{\lambda_1, \dots, \lambda_n\}} |p_k(z)|, \quad (4.17)$$

where  $\kappa(\mathbf{V})$  is the spectral condition number (2.7). Combining this with (4.15) yields the desired result.  $\square$

This implies that for a given matrix  $\mathbf{A}$ , we can expect GMRES to converge quickly if the condition number  $\kappa(\mathbf{V})$  is small, i.e. the matrix is close to normal, or if a properly normalized degree  $k$  polynomials can be found, whose size decays quickly on the spectrum of  $\mathbf{A}$ . Figure 4.1 shows the spectra of two randomly generated matrices of order  $n = 200$ ,  $A_1$  and  $A_2$ . The spectrum of  $A_1$  is drawn from a random Gaussian centered around 0, while the spectrum of  $A_2$  is drawn from a Gaussian centered around 5. We see that GMRES shows much faster convergence for  $A_2$ , as we would expect from Theorem 4.4.1. For  $A_1$  on the other hand, the residual stays more or less of the same magnitude until the final iteration  $k = 200$ , at which point it converges in one step. This is caused by the spectral properties of  $A_1$  as both matrices have the same  $\kappa(\mathbf{V})$ .



**Figure 4.1:** Influence of the spectrum on the convergence rate of GMRES. The figure on the left shows the spectra of two randomly generated matrices  $\mathbf{A}_1$  and  $\mathbf{A}_2$ , whose eigenvalues are drawn from Gaussian distribution centered at 0 and 5, respectively. The matrices are then generated by multiplying them with a randomly generated, unitary transformation  $V$ . The figure on the right shows the relative residual  $\|r_k\|_2 / \|r_0\|_2$  after each iteration for both matrices.

## 4.5 Preconditioning

What if we would like to use GMRES but  $A$  does not have the properties described in Section 4.4? This problem is addressed by introducing preconditioning. Given a linear system (1.22), let us imagine that we can compute an invertible  $n \times n$  matrix  $P$ , or its inverse  $P^{-1}$ , such that we can apply it efficiently either to a vector or  $A$ . Then, if  $P^{-1}A$  has favorable properties, we can apply GMRES to the preconditioned linear system

$$P^{-1}Ax = P^{-1}b. \quad (4.18)$$

This system is often called the left-preconditioned system, as  $P^{-1}$  is applied from the left. We can also apply our preconditioner from the right, which yields the right-preconditioned system

$$AP^{-1}y = b \quad (4.19a)$$

$$x = P^{-1}y. \quad (4.19b)$$

It is worth noting that  $x = P^{-1}y$  comes at virtually no cost as we require  $P^{-1}$  to be applicable at low cost in the first place.

The obvious question is: what is a useful preconditioner? Two extreme cases come to mind:  $P = I$  and  $P = A$ . In the former case, applying  $P^{-1}$  is very easy but we have not improved the problem. In the latter case, applying  $P^{-1}$  is as hard as the original problem and we have therefore solved the original problem in one iteration. Useful preconditioner lie somewhere in between these two extreme cases. It seems only natural then, that many preconditioners are problem-specific and rely on a priori information.

Let us consider one such problem and give an example of a problem-specific preconditioner. We seek to solve the Helmholtz problem (1.3) using a finite element approximation and GMRES. To this end, we discretize the Helmholtz problem on  $\Omega = [-1, 1]^2$  using a finite element discretization of polynomial order  $p = 1$  and mesh width  $h = 1/50$ . This yields a linear system  $Ax = b$  of size 2401 with the problem matrix

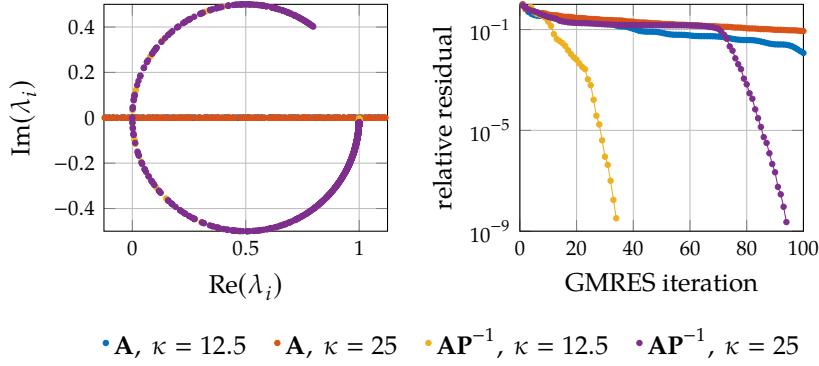
$$A = S - \kappa^2 M,$$

where  $S, M$  are the stiffness- and mass-matrices respectively and  $\kappa$  the wavenumber. Figure 4.2 depicts a fraction of the spectrum of  $A$  for the

wavenumbers  $\kappa = 12.5$  and  $\kappa = 25$ . In both cases, we obtain poor GMRES convergence, caused by the indefinite spectrum. Following [27, 28], we now use the *shifted Laplacian preconditioner*

$$\mathbf{P} = \mathbf{S} - \kappa^2(\beta_1 + \beta_2 i)\mathbf{M}, \quad (4.20)$$

with  $(\beta_1, \beta_2) = (1, 0.5)$ , to precondition our problem. In contrast to the original matrix, we observe that the spectrum of the preconditioned matrix  $\mathbf{AP}^{-1}$  is nicely clustered on a circle around  $0.5 + 0i$ . Unsurprisingly GMRES fares much better on the preconditioned system, as can be seen on the right figure. Effectively, the shifted Laplacian approach computes



**Figure 4.2:** Influence on the spectrum if the shifted Laplacian technique is used to precondition the Helmholtz problem. The figure on the left shows the spectra of  $\mathbf{A}$  and  $\mathbf{AP}^{-1}$  respectively. The spectrum of  $\kappa = 12.5$  is covered by the spectrum of  $\kappa = 25$ , however it shows the same behavior.

a solution to the shifted Helmholtz equation

$$-\nabla^2 u - \kappa^2(\beta_1 - \beta_2 i)u = f, \quad (4.21)$$

where the introduction of a complex component corresponds to the introduction of a damping term  $u_t$  to the original wave equation (1.1). If the complex shift is not too large, we can expect the method to yield a result that is still useful for preconditioning, yet easy to compute using an iterative solver. We have not discussed how  $\mathbf{P}^{-1}$  can be applied efficiently. For a detailed treatment, we refer the reader to [27, 28].

The following list offers a short (and incomplete) overview of popular preconditioning techniques for iterative solvers of linear systems:

**Diagonal/Jacobi preconditioner** A very simple preconditioning technique is to simply take the diagonal as a preconditioner:  $\mathbf{P} = \text{diag}(\mathbf{A})$ . More generally, one may choose  $\mathbf{c} \in \mathbb{C}^n$  and  $\mathbf{P} = \text{diag}(\mathbf{c})$ , in the attempt to minimize  $\kappa(\mathbf{P}^{-1}\mathbf{A})$ .

**Incomplete LU factorization** Another popular choice is the incomplete LU decomposition. For sparse matrices, we can compute an incomplete LU factorization  $\mathbf{A} \approx \mathbf{LU}$  by simply disregarding fill-in and only compute entries which lie in the sparsity pattern of the original matrix. Other variants use a less aggressive approach, where entries are dropped if their value is below a user-specified threshold  $\epsilon$ .

**Gauss-Seidel/Successive over-relaxation** This preconditioner is based on the Gauss-Seidel method, which is another iterative method for solving linear systems. The idea is to decompose the matrix into its upper- and lower-triangular parts  $\mathbf{A} = \mathbf{L} + \mathbf{U}$ . The resulting equation  $\mathbf{x} = \mathbf{L}^{-1}(\mathbf{b} - \mathbf{Ux})$  can be understood as a fixed-point equation and is solved via fixed-point iteration.

**Multigrid solvers** Perhaps the most prominent member in this group are multigrid methods, which exploit the hierarchical nature of problems, such as problems arising from PDEs. The core idea of these methods is to accelerate the convergence of a cheap solver which solves the solution on a grid with a solution computed on a coarser grid [11, 26, 29]. One can recursively apply the multigrid method on this coarser grid until the problem becomes cheap enough to be solved directly. As a consequence, multigrid methods can be used both as a solver and as a preconditioner. Typically, we distinguish between algebraic and geometric multigrid methods, where the former assumes no prior information and can therefore be applied without the prior knowledge of a multilevel hierarchy [26].

# 5

## Hierarchical matrices

This chapter introduces the notion of hierarchical matrices. Hierarchical matrices can be regarded as matrix representations of integral operators such as

$$\mathcal{G}(u) = \int_{\Omega} g(\mathbf{x}, \mathbf{y}) u(\mathbf{y}) d\mathbf{y} \quad (5.1)$$

where  $\Omega \subset \mathbb{R}^d$  is a domain of interest embedded in a  $d$ -dimensional space and

$$g : \mathbb{R}^d \times \mathbb{R}^d \rightarrow \mathbb{R} \quad (5.2)$$

is its *kernel function*, which may be singular along the diagonal  $\mathbf{x} = \mathbf{y}$ . Notable examples of kernel functions are fundamental solutions or kernel functions arising in Gaussian processes. These kernel functions are often non-local and their approximation using a Galerkin method

$$\mathbf{G}_{ij} = \langle \varphi_i, \mathcal{G}\varphi_j \rangle \quad \varphi_i, \varphi_j \in V_h \quad (5.3)$$

can therefore be expected to result in a dense matrix. However, if we consider  $g$  on  $X \times Y$ , where  $X \subset \mathbb{R}^d$  and  $Y \subset \mathbb{R}^d$  are sufficiently separated disjoint domains,  $g$  may admit a degenerate approximation of the form

$$g(\mathbf{x}, \mathbf{y}) \approx \sum_{l=0}^{k-1} p_l(\mathbf{x}) q_l(\mathbf{y}) \quad (5.4)$$

for  $\mathbf{x} \in X$  and  $\mathbf{y} \in Y$ . As a consequence, some blocks in  $\mathbf{G}$  may admit a rank- $k$  approximation, which we would like to exploit.

### 5.1 Approximate separability

We are concerned with the approximation of kernel functions using low-rank matrices. Our introduction borrows heavily from [30], which provides a detailed introduction to the topic. As we limit our analysis to the most important details, we point the reader to [25, 30] for a more rigorous and detailed treatment of the subject.

**Definition 5.1.1** (separable expression) *Let  $g_k : \mathbb{R}^d \times \mathbb{R}^d \rightarrow \mathbb{R}$  and  $X, Y \subset \mathbb{R}^d$ . We call  $g_k$  a separable expression in  $X \times Y$  iff there exist functions  $p_l, q_l$ , such that*

$$g_k(\mathbf{x}, \mathbf{y}) = \sum_{l=1}^k p_l(\mathbf{x}) q_l(\mathbf{y}) \quad \text{for } \mathbf{x} \in X, \mathbf{y} \in Y. \quad (5.5)$$

*The number  $k$  is called the separation rank of  $g_k$ .*

We are interested in separable approximations to integral operators of the form (5.5). To this end, we introduce the notion of *approximate separability*.

5.1 Approximate separability . . . . .	37
Taylor expansions . . . . .	38
Multipole expansions . . . . .	41
5.2 Block cluster trees . . . . .	43
5.3 Hierarchical matrices . . . . .	45
5.4 Nested bases . . . . .	47

**Definition 5.1.2** (separable expansion) Let  $g_k$  be a separable expression in  $X \times Y$ . For a small positive number  $\epsilon > 0$ , we call  $g_k$  a separable expansion of  $g$  to accuracy  $\epsilon$ , if it satisfies

$$g(\mathbf{x}, \mathbf{y}) = g_k(\mathbf{x}, \mathbf{y}) + R_k(\mathbf{x}, \mathbf{y}) \quad \text{for } \mathbf{x} \in X, \mathbf{y} \in Y, \quad (5.6)$$

with

$$|R_k| \leq \epsilon.$$

Clearly, we would like  $R_k$  to vanish quickly as we increase  $k$ , i.e.  $R_k \rightarrow 0$  as  $k \rightarrow \infty$ .<sup>1</sup> There are various ways of finding separable expansions. Popular approaches are Taylor expansions, multipole expansion and interpolation [25, 30].

## Separability with Taylor expansions

A general approach for finding a separable expansion is to use a Taylor expansion of the kernel in one of its two variables. Let  $g \in C^m(X \times Y)$ . We choose a point  $\mathbf{x}_0 \in X$  and formulate the Taylor expansion

$$g(\mathbf{x}, \mathbf{y}) = \sum_{\alpha \in \mathbb{N}_0^d, |\alpha| \leq m} (\mathbf{x} - \mathbf{x}_0)^\alpha \frac{1}{\alpha!} \partial_x^\alpha g(\mathbf{x}_0, \mathbf{y}) + R_k(\mathbf{x}, \mathbf{y}) \quad (5.7)$$

in  $\mathbf{x}$ , where we sum over all multiindices  $\alpha \in \mathbb{N}_0^d$ , which are below a certain order  $|\alpha| \leq m$ .<sup>2</sup> Here, the separation rank

$$k(m, d) = \frac{(m+d)(m+d-1)\dots(m+1)}{d!} = \binom{m+d}{d} \quad (5.8)$$

is the number of terms in the Taylor approximation (5.7) and therefore a function of  $m$  and the spatial dimension  $d$ . We note that we might also have chosen to expand  $g(\mathbf{x}, \mathbf{y})$  in  $\mathbf{y}$  with respect to some point  $\mathbf{y}_0 \in Y$ .

**Remark 5.1.1** Many kernels of interest have the form  $g(\mathbf{x}, \mathbf{y}) = G(\mathbf{r})$ , where  $\mathbf{r}$  is the radius vector  $\mathbf{x} - \mathbf{y}$ . In these cases, the expansion can be performed in  $\mathbf{r}$  and we obtain a separable expression (5.5) with polynomial coefficients  $p_l(\mathbf{x})$  and  $q_l(\mathbf{y})$ .

In the following, we discuss under which assumptions we can expect the remainder term in (5.7) to vanish exponentially. To understand this requires some additional properties with respect to the kernel  $g(\cdot, \cdot)$ :

To bound the remainder term  $R_k$ , we introduce the notion of asymptotical smoothness:

**Definition 5.1.3** (asymptotically smooth kernels) Let  $g : \mathbb{R}^d \times \mathbb{R}^d \rightarrow \mathbb{R}$  and  $X, Y \subseteq \mathbb{R}^d$ , such that  $g$  is defined and arbitrarily often differentiable if  $\mathbf{x} \in X$ ,  $\mathbf{y} \in Y$  and  $\mathbf{x} \neq \mathbf{y}$ . We call  $g$  asymptotically smooth in  $X \times Y$  if

$$\left| \partial_x^\alpha \partial_y^\beta g(\mathbf{x}, \mathbf{y}) \right| \leq C(\alpha + \beta)! c_0^{|\alpha| + |\beta|} \|\mathbf{x} - \mathbf{y}\|^{-|\alpha| - |\beta| - \sigma} \quad (5.9)$$

for  $\mathbf{x} \in X$ ,  $\mathbf{y} \in Y$ ,  $\mathbf{x} \neq \mathbf{y}$ , multiindices satisfying  $\alpha, \beta \in \mathbb{N}_0^d$ ,  $\alpha + \beta \neq \mathbf{0}$  and some constants  $C, c_0, \sigma \in \mathbb{R}_{>0}$ .

1: It is worth noting that the expansion terms  $p_l(\mathbf{x}), q_l(\mathbf{y})$  may implicitly depend on the concrete value chosen for  $k$ . If they do not however, convergence of  $g(\mathbf{x}, \mathbf{y}) - g_k(\mathbf{x}, \mathbf{y})$  is equivalent to the convergence of the infinite series

$$g(\mathbf{x}, \mathbf{y}) = \sum_{l=1}^{\infty} p_l(\mathbf{x}) q_l(\mathbf{y}).$$

2: We use multiindex notation to denote multidimensional powers, derivatives, etc. For instance, the notation  $\mathbf{x}^\alpha$  with the multiindex  $\alpha \in \mathbb{N}_0^d$  refers to  $\mathbf{x}^\alpha = x_1^{\alpha_1} x_2^{\alpha_2} \dots x_d^{\alpha_d}$ , where each component  $x_i$  is mapped to its  $\alpha_i$ -th power with the corresponding element in  $\alpha$ . Similar rules apply for the multidimensional derivative  $\partial_x^\alpha = \partial_{x_1}^{\alpha_1} \partial_{x_2}^{\alpha_2} \dots \partial_{x_d}^{\alpha_d}$ , factorial  $\alpha! = \alpha_1! \alpha_2! \dots \alpha_d!$ , etc. The absolute value of a multiindex denotes the sum of all entries:  $|\alpha| = \alpha_1 + \alpha_2 + \dots + \alpha_d$ .

In particular, if we consider derivatives only in  $\mathbf{x}$ , Definition 5.1.3 yields

$$|\partial_x^\alpha g(\mathbf{x}, \mathbf{y})| \leq C\alpha! c_0^{|\alpha|} \|\mathbf{x} - \mathbf{y}\|^{-|\alpha|-\sigma}. \quad (5.10)$$

Due to the asymptotical smoothness of  $g$ , we can write the remainder term in (5.7) as

$$R_k = \sum_{|\alpha| \geq m+1}^{\infty} \frac{(\mathbf{x} - \mathbf{x}_0)^\alpha}{\alpha!} \partial_x^\alpha g(\mathbf{x}_0, \mathbf{y}). \quad (5.11)$$

Using (5.10) and  $\sum_{|\alpha|=l} |(\mathbf{x} - \mathbf{x}_0)^\alpha| = \mathcal{O}(\|\mathbf{x} - \mathbf{x}_0\|^l)$ , we have

$$\begin{aligned} |R_k| &\leq C \sum_{|\alpha| \geq m+1} c_0^{|\alpha|} \frac{|(\mathbf{x} - \mathbf{x}_0)^\alpha|}{\alpha!} \alpha! \|\mathbf{x}_0 - \mathbf{y}\|^{-|\alpha|-\sigma} \\ &= \frac{C}{\|\mathbf{x}_0 - \mathbf{y}\|^\sigma} \sum_{|\alpha| \geq m+1} \frac{c_0^{|\alpha|}}{\|\mathbf{x}_0 - \mathbf{y}\|^{|\alpha|}} |(\mathbf{x} - \mathbf{x}_0)^\alpha| \\ &= \frac{C}{\|\mathbf{x}_0 - \mathbf{y}\|^\sigma} \sum_{l=m+1}^{\infty} \left( \frac{c_0}{\|\mathbf{x}_0 - \mathbf{y}\|} \right)^l \sum_{|\alpha|=l} |(\mathbf{x} - \mathbf{x}_0)^\alpha| \\ &\leq \frac{C}{\|\mathbf{x}_0 - \mathbf{y}\|^\sigma} \sum_{l=m+1}^{\infty} \left( \frac{\gamma \|\mathbf{x} - \mathbf{x}_0\|}{\|\mathbf{x}_0 - \mathbf{y}\|} \right)^l \\ &\leq \tilde{C} \sum_{l=m+1}^{\infty} \vartheta^l \leq \tilde{C} \frac{\vartheta^m}{1 - \vartheta}, \end{aligned} \quad (5.12)$$

where we have introduced the new constants  $\gamma, \tilde{C} \in \mathbb{R}_{>0}$ , which depend on  $C, \sigma, m, \mathbf{x}_0$ . Therefore, the remainder term  $R_k$  vanishes exponentially<sup>3</sup> if  $\vartheta < 1$ , which is given by

$$\vartheta = \frac{\gamma \|\mathbf{x} - \mathbf{x}_0\|}{\|\mathbf{x}_0 - \mathbf{y}\|} \leq \frac{\gamma \max_{\mathbf{x} \in X} \|\mathbf{x} - \mathbf{x}_0\|}{\|\mathbf{x}_0 - \mathbf{y}\|}.$$

Consequently, the condition

$$\text{diam}(X) \leq \eta \text{dist}(X, Y) \quad (5.13)$$

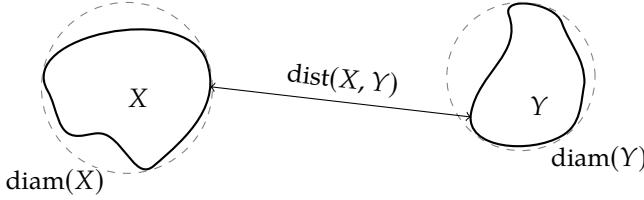
implies  $\vartheta < \eta\gamma$ . In combination with previous assumptions  $\eta\gamma < 1$  is sufficient to guarantee convergence of the separable approximation. A similar condition can be formulated for  $\mathbf{y}$  for cases, in which we choose to expand  $g(\cdot, \cdot)$  in  $\mathbf{y}$ :

$$\text{diam}(Y) \leq \eta \text{dist}(X, Y). \quad (5.14)$$

<sup>3</sup>: A more detailed analysis with sharper bounds for  $|R_k|$  can be found in [30].

**Definition 5.1.4** ( $\eta$ -admissibility) *Let  $X$  and  $Y$  be  $d$ -dimensional subdomains  $X, Y \subset \mathbb{R}^d$  and  $\eta \in \mathbb{R}_{>0}$ . We call the pair  $(X, Y)$   $\eta$ -admissible if (5.13) or (5.14) (or both) are satisfied.*

This admissibility condition is depicted in Figure 5.1. A smaller  $\eta$  means that  $g$  will be better-suited for a separable expansion, and therefore an approximation with the matrix formats that we will introduce in this chapter.



**Figure 5.1:** Illustration of the admissibility condition.

This brings us to the final theorem of this section, which summarizes the previous statements

**Theorem 5.1.1** Let  $g : \mathbb{R}^d \times \mathbb{R}^d \rightarrow \mathbb{R}$  be an asymptotically smooth kernel in  $X, Y \subseteq \mathbb{R}^d$ . Moreover, let  $X$  and  $Y$  be  $\eta$ -admissible, where  $\eta < 1/\gamma$  with the constant  $\gamma \in \mathbb{R}_{>0}$  from (5.12). Then,  $g$  admits a separable approximation  $g_k$  of separation rank  $k = k(m, d)$  in  $X \times Y$ , with an error term that vanishes exponentially, according to (5.12).

*Proof.* The proof is given by preceding calculations.  $\square$

Thus, we have identified sufficient conditions under which we may find separable approximations of the form (5.7). It is important to note that those are not necessary conditions as we may have chosen another form of the separable expression, for instance via interpolation or multipole expansion [25, 30].

In the following, we state an independent result, which demonstrates a sufficient separability condition for the Green's function of the two-dimensional Laplacian in unbounded domains.

**Example 5.1.1** (Green's function of the two-dimensional Laplacian) The Green's function of the Laplacian in an unbounded, two-dimensional domain is given by  $\ln \|\mathbf{r}\|/2\pi$ . Here  $\mathbf{r} = \mathbf{x} - \mathbf{y} = (r_1, r_2)^\top$  denotes the radius vector. We ignore the constant and set  $G(\mathbf{r}) = \ln \|\mathbf{r}\|$ . We then compute its  $\alpha$ -th derivative, where  $\alpha = (\alpha_1, \alpha_2) \in \mathbb{N}_0^2$ . After some calculations, we obtain

$$\begin{aligned} \partial_r^\alpha G(\mathbf{r}) &= -\frac{(\alpha_1 + \alpha_2 - 1)!}{(r_1^2 + r_2^2)^{\alpha_1 + \alpha_2}} \left( (-1)^{\alpha_1 + \alpha_2} (r_1 - ir_2)^{\alpha_1} (ir_1 + r_2)^{\alpha_2} \right. \\ &\quad \left. + (i)^{\alpha_1 + \alpha_2} (ir_1 - r_2)^{\alpha_1} (r_1 + ir_2)^{\alpha_2} \right) \end{aligned}$$

and consequently,

$$|\partial_r^\alpha G(\mathbf{r})| \leq \frac{(\alpha_1 + \alpha_2 - 1)!}{\|\mathbf{r}\|^{\|\alpha\|}}.$$

Thus, we can bound the remainder term as

$$\begin{aligned} |R_k| &\leq \sum_{|\alpha| \geq m+1} \frac{|(\mathbf{r} - \mathbf{r}_0)^\alpha|}{\alpha!} |\partial_r^\alpha G(\mathbf{r}_0)| \\ &\leq \sum_{|\alpha| \geq m+1} \frac{|(\mathbf{r} - \mathbf{r}_0)^\alpha|}{\alpha_1 + \alpha_2} \binom{\alpha_1 + \alpha_2}{\alpha_2} \frac{1}{\|\mathbf{r}_0\|^{\|\alpha\|}} \end{aligned}$$

$$\begin{aligned}
&= \sum_{l=m+1}^{\infty} \frac{1}{l \|\mathbf{r}_0\|^l} \sum_{|\alpha|=l} \binom{\alpha_1 + \alpha_2}{\alpha_2} |(\mathbf{r} - \mathbf{r}_0)^\alpha| \\
&= \sum_{l=m+1}^{\infty} \frac{1}{l} \frac{\|\mathbf{r} - \mathbf{r}_0\|_1^l}{\|\mathbf{r}_0\|_2^l}.
\end{aligned}$$

We insert  $\mathbf{r} = \mathbf{x} - \mathbf{y}$  and  $\mathbf{r}_0 = \mathbf{x}_0 - \mathbf{y}_0$ , where  $\mathbf{x}_0$  and  $\mathbf{y}_0$  are suitably chosen expansion centers for both  $\mathbf{x}$  and  $\mathbf{y}$ . To ensure exponential convergence, we must bound the term

$$\begin{aligned}
\vartheta &= \frac{\|(x - x_0) - (y - y_0)\|_1}{\|\mathbf{x}_0 - \mathbf{y}_0\|_2} \leq \sqrt{2} \frac{\|(x - x_0) - (y - y_0)\|_2}{\|\mathbf{x}_0 - \mathbf{y}_0\|_2} \\
&\leq \sqrt{2} \frac{\|(x - x_0)\| + \|(y - y_0)\|_2}{\|\mathbf{x}_0 - \mathbf{y}_0\|_2} \\
&\leq \sqrt{2} \frac{\text{diam}(X) + \text{diam}(Y)}{\text{dist}(X, Y)}
\end{aligned}$$

by 1. A valid admissibility condition there would be the  $\eta$ -admissibility condition with  $\eta = 2\sqrt{2}$ . In many cases however, this condition can be much relaxed, as we can choose  $\mathbf{x}_0$  and  $\mathbf{y}_0$ .

## Separability with multipole expansions

So far, we have motivated the use of separable approximations using Taylor expansions of asymptotically smooth kernel functions. This approach is fairly general but seldom practical. An alternative expansion is the *multipole expansion*, which is particularly useful in the context of physically motivated kernel functions [25, 31, 32].

Most Green's functions of practical relevance are related to the potential of a single unit charge. The electrostatic field of a unit point charge in three dimensions is given by

$$g(\mathbf{x}, \mathbf{y}) = \frac{1}{\|\mathbf{x} - \mathbf{y}\|_2}, \quad (5.15)$$

which is the Green's function of the corresponding Laplace equation in  $\mathbb{R}^3$ . We imagine that  $n$  charges  $q_j$  are located at source points  $\mathbf{y}_j$  within the subdomain  $Y$  centered at  $\mathbf{y}_0$ . The resulting potential  $u$  at the target location  $\mathbf{x}$  is then given by

$$u(\mathbf{x}) = \sum_{j=1}^n q_j g(\mathbf{x}, \mathbf{y}_j). \quad (5.16)$$

A well-known identity for the electrostatic potential (5.15) is its relation to the generating function of the Legendre polynomials  $P_k$  [33, pp. 148]. In particular, we have

$$\frac{1}{\|\mathbf{x} - \mathbf{y}\|_2} = \sum_{l=0}^{\infty} P_l(\cos \vartheta(\mathbf{y}, \mathbf{x})) \frac{\|\mathbf{x}\|_2^l}{\|\mathbf{y}\|_2^{l+1}}, \quad (5.17)$$

where  $\|\mathbf{y}\|_2 > \|\mathbf{x}\|_2$  and  $\vartheta(\mathbf{y}, \mathbf{x})$  is the angle between  $\mathbf{x}$  and  $\mathbf{y}$ . We remark that this expansion in itself is not yet a separable approximation of (5.15) as the angle depends on both  $\mathbf{x}$  and  $\mathbf{y}$ . In two dimensions, we can use

the addition theorem for spherical harmonics [34]

$$P_l(\cos(\vartheta_1 + \vartheta_2)) = \sum_{m=0}^l P_l^m(\cos(\vartheta_1))P_l^m(\cos(\vartheta_2)),$$

where  $P_k^m$  denote the associated Legendre polynomials [34]. In three dimensions, a similar addition theorem for angles on the sphere yields the spherical harmonics for the multipole expansion (5.17). Inserting the addition theorem gives us

$$\begin{aligned} \frac{1}{\|\mathbf{x} - \mathbf{y}\|_2} &= \sum_{l=0}^{\infty} P_l(\cos(\vartheta(\mathbf{x}, \mathbf{y}_0) + \vartheta(\mathbf{y}_0, \mathbf{y}))) \frac{\|\mathbf{y} - \mathbf{y}_0\|_2^l}{\|\mathbf{x} - \mathbf{y}_0\|_2^{l+1}}, \\ &= \sum_{l=0}^{\infty} \sum_{m=0}^l P_l^m(\cos \vartheta(\mathbf{x}, \mathbf{y}_0)) P_l^m(\cos \vartheta(\mathbf{y}_0, \mathbf{y})) \frac{\|\mathbf{y} - \mathbf{y}_0\|_2^l}{\|\mathbf{x} - \mathbf{y}_0\|_2^{l+1}}, \end{aligned} \quad (5.18)$$

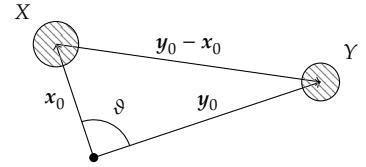
which is a separable expansion in  $\mathbf{x}$  and  $\mathbf{y}$ . Here we have introduced the point  $\mathbf{y}_0$  as the center for our expansion, as illustrated in Figure 5.2. To make a statement regarding the quality of a truncated expansion of (5.18), we need to develop error bounds. We point the reader to the original introduction of the fast multipole method [31] for an extended discussion on the matter.

Let us imagine that we want to evaluate (5.16) at  $m$  target points  $\mathbf{x}_i$  located in the subdomain  $X$  centered at  $\mathbf{x}_0$ . This is clearly an  $\mathcal{O}(mn)$  operation. Substituting (5.18) into (5.16) yields

$$\begin{aligned} u(\mathbf{x}_i) &= \sum_{j=1}^n q_j \sum_{l=0}^{\infty} \sum_{m=0}^l P_l^m(\cos \vartheta(\mathbf{x}_j, \mathbf{y}_0)) P_l^m(\cos \vartheta(\mathbf{y}_0, \mathbf{y})) \frac{\|\mathbf{y} - \mathbf{y}_0\|_2^l}{\|\mathbf{x} - \mathbf{y}_0\|_2^{l+1}} \\ &\approx \sum_{l=0}^{\infty} \frac{1}{\|\mathbf{x}_i - \mathbf{y}_0\|_2^{l+1}} \sum_{j=1}^n q_j \underbrace{\sum_{m=0}^l P_l^m(\cos \vartheta(\mathbf{x}_0, \mathbf{y}_0)) P_l^m(\cos \vartheta(\mathbf{y}_0, \mathbf{y}_j)) \|\mathbf{y}_j - \mathbf{y}_0\|_2^l}_{=Q_{lj}} \\ &\approx \sum_{l=0}^k \frac{1}{\|\mathbf{x}_i - \mathbf{y}_0\|_2^{l+1}} \sum_{j=1}^n q_j Q_{lj}. \end{aligned} \quad (5.19)$$

We have used the fact that  $X$  and  $Y$  are well-separated subdomains, which allowed us to replace the angle  $\vartheta(\mathbf{x}_j, \mathbf{y}_0)$  with  $\vartheta(\mathbf{x}_0, \mathbf{y}_0)$ . We then proceed by computing the so-called multipole moments  $Q_{lj}$  which contribute to the approximation of the potential centered at  $\mathbf{y}_0$ . In the final step we have chosen a separation rank  $k$  at which we truncate the multipole expansion. The final result is a separable approximation of rank  $k$ , which reduces the computational cost to  $\mathcal{O}(k(m+n))$  operations. Evidently, this separation rank should be chosen with respect to some error tolerance  $\epsilon$ . We refer the reader again to [31] for a rigorous mathematical treatment of the subject.

The multipole expansion (5.19) illustrates how separable approximations can be obtained in practice. In contrast to Taylor expansions, multipole expansions only span harmonic functions. An order  $k$  multipole expansion in three dimensions as discussed here, will have only  $\mathcal{O}(k^2)$  terms, whereas the corresponding Taylor expansion will have  $\mathcal{O}(k^3)$  terms. This makes multipole expansions superior to Taylor expansions and lower truncation errors can be achieved with the same number of terms. The



**Figure 5.2:** Illustration of the situation in the multipole expansion. The source points  $\mathbf{y}_j$  are located in the area  $Y$ , centered at  $\mathbf{y}_0$ . Similarly, target locations  $\mathbf{x}_i$  are contained in the subdomain  $X$ .

hierarchically structured matrices that we present later in this chapter can be regarded as matrix counterparts to multipole expansions (and the fast multipole method in particular [35]).

The following example presents an estimation of the separation rank for multipole expansions of the Helmholtz problem in two and three dimensions based on results derived in [36].

**Example 5.1.2** (Expansion errors for the Green's function of the Helmholtz problem in two dimensions) We consider the Green's function

$$g(\mathbf{x}, \mathbf{y}) = \frac{i}{4} H_0^{(1)}(\kappa \|\mathbf{x} - \mathbf{y}\|) \quad (5.20)$$

of the Helmholtz problem (1.3) on the unbounded  $\mathbb{R}^2$  with radiating boundary condition.  $H_\alpha^{(1)}(x)$  denotes the Hankel function of the first kind, defined as  $H_\alpha^{(1)}(x) = J_\alpha(x) + iY_\alpha(x)$ , where  $J_\alpha(x)$  and  $Y_\alpha(x)$  are the Bessel functions of the first and second kind, respectively. As in the computations before, we aim to separate the points  $\mathbf{x}$  and  $\mathbf{y}$  into well-separated subdomains. Suppose that  $\mathbf{p} = \kappa(\mathbf{x}_0 - \mathbf{y}_0)$  and  $\mathbf{q} = \kappa(\mathbf{x}_0 - \mathbf{y}_0 - \mathbf{x} + \mathbf{y})$ , where  $\mathbf{x}_0$  and  $\mathbf{y}_0$ . At the core of the multipole methods lies the expansion of the Green's function into an infinite series. We use Graf's addition theorem [36]

$$H_0^{(1)}(\kappa \|\mathbf{x} - \mathbf{y}\|) = \sum_{l=-\infty}^{\infty} H_l^{(1)}(\mathbf{p}) J_l(\mathbf{q}) \exp(i l (\vartheta_p - \vartheta_q)), \quad (5.21)$$

where we have introduced polar coordinates  $(\mathbf{p}, \vartheta_p)$  and  $(\mathbf{q}, \vartheta_q)$  to specify the vectors  $\mathbf{p}$  and  $\mathbf{q}$ . Truncating this expansion at  $k$  terms yields the remainder term

$$|R_k| \lesssim \sum_{l=k+1}^{\infty} |H_l^{(1)}(\mathbf{p}) J_l(\mathbf{q})| \leq \max_{\{\mathbf{p} \geq p_{\min}, \mathbf{q} \leq q_{\max}\}} \sum_{l=k+1}^{\infty} |H_l^{(1)}(\mathbf{p}) J_l(\mathbf{q})|. \quad (5.22)$$

As  $\mathbf{x}$  and  $\mathbf{y}$  are usually contained in well-separated clusters  $X$  and  $Y$ , it makes sense to bound  $p \geq p_{\min} = \kappa \text{dist}(X, Y)$  and  $q \leq q_{\max} = \kappa(\text{diam } X + \text{diam } Y)$ .

Amini and Profit [36] derive a theoretical error bounds for (5.22) and an algorithm to compute them. The proper separation rank for a given tolerance  $\epsilon > 0$  can then be determined by computing the error bound (5.22) until  $|R_k| < \epsilon$  is met. As in the prior analysis, we assume that the satisfy an  $\eta$ -admissibility condition and fix  $q = \eta p$ . For  $\eta = 0.5$  and a fixed tolerance of  $2^{-20}$ , Amini and Profit report the separation rank  $k$  to behave roughly as

$$k \sim q + 5 \log(q + \pi), \quad (5.23)$$

which grows linearly with the wave number for large  $\kappa$  [36].

## 5.2 Block cluster trees

For a general matrix  $\mathbf{A} \in \mathbb{R}^{m \times n}$ , consider the row index set  $I = \{1, 2, \dots, m\}$  and column index set  $J = \{1, 2, \dots, n\}$ . In the preceding section, we established that separable approximations may be used in

well-separated domains that satisfy some form of admissibility condition. Therefore, if  $A$  is similar to  $G$  we can expect that some blocks of  $A$  admit rank- $k$  approximations.

To find such partitions, we start by hierarchically partitioning the index sets  $I$  and  $J$ . Thus, we introduce the notion of cluster trees:

**Definition 5.2.1** (cluster tree) *Let  $I = \{1, 2, \dots, m\}$  be an index set and let  $\mathcal{T}_I$  be a tree of maximal depth  $L$ , where each of its nodes  $I_i^l \in \mathcal{T}$  is a subset of  $I$ . The superindex  $l$  in  $I_i^l$  denotes the level, starting from the top-level  $l = 1$ , and the subscript  $i$  enumerates the node within the tree in post-order. We call  $\mathcal{T}_I$  a cluster tree iff*

- ▶ the root node is the entire index set  $I$ ,
- ▶ nodes at each level  $l$  are disjoint:  $\forall i \neq j : I_i^l \cap I_j^l = \emptyset$ ,
- ▶ every node  $I_i^l$  with children nodes  $\text{children}(i)$  is the disjoint union of all its children  $I_i^l = \bigcup_{c \in \text{children}(i)} I_c^{l+1}$ ,
- ▶ each index set  $I_i^l$  is a contiguous range of integers, i.e. indices appear in incremental order.

The second to last condition ensures that the union of all index sets on a given level  $l$  is a valid partitioning of the entire index set  $I$  (assuming that  $\mathcal{T}_I$  is a balanced tree). The last requirement of having contiguous index sets is an optional one as it can be achieved by simply reordering the underlying matrix accordingly. As such, there is no loss of generality and we will assume that this is generally true for the cluster trees that we consider.<sup>4</sup> <sup>5</sup>

A straight-forward way of generating cluster trees is bisection. Figure 5.3 depicts a cluster tree of depth 3 of the index set  $I = \{1, 2, \dots, 12\}$ .

Such hierarchical partitions can also be constructed by using the underlying geometry. This may lead to better results but it is a much more involved process as the geometry has to be taken into account. More details can be found in [25, 30, 37, 38].

We are interested in hierarchically partitioning the matrix  $A$ . This is achieved with block cluster trees:

**Definition 5.2.2** (block cluster tree) *Let  $I = \{1, 2, \dots, m\}$  and  $J = \{1, 2, \dots, n\}$  be index sets and let  $\mathcal{T}_{I \times J}$  be a tree. We call  $\mathcal{T}_{I \times J}$  a block cluster tree iff:*

- ▶  $I \times J$  is the root of  $\mathcal{T}_{I \times J}$ ,
- ▶ any two distinct nodes  $B_i^l, B_j^l \in \mathcal{T}_{I \times J}$ ,  $i \neq j$  that share the same level  $l$  are disjoint:  $B_i^l \cap B_j^l = \emptyset$ ,
- ▶ every node  $B_i^l$  with children nodes  $\text{children}(i)$  is the disjoint union of its children:  $B_i^l = \bigcup_{c \in \text{children}(i)} B_c^{l+1}$ ,
- ▶ for each block  $B_i^l$ , there exist contiguous index sets  $I_i^l$  and  $J_i^l$  such that  $B_i^l$  can be expressed as their cartesian product:  $B_i^l = I_i^l \times J_i^l$ .

Evidently, block cluster trees share many of the properties of cluster trees. The last property ensures that we are only considering matrix blocks and not random entries.

4: The notation  $I_i^l$  is slightly redundant as we do not need to specify the level  $l$  in the tree once we have specified its position  $i$  in the tree. For the sake of clarity, we will drop the superscript  $l$  whenever the circumstances allow.

5: With a slight abuse of notation, we shall interchangably refer to nodes in  $\mathcal{T}_I$  either by their index  $i$  or by the concrete index set  $I_i^l$ .

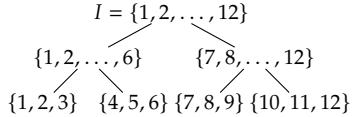
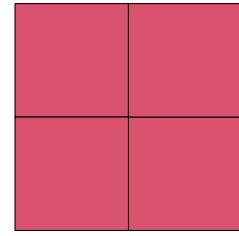
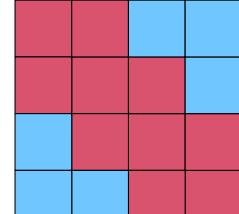


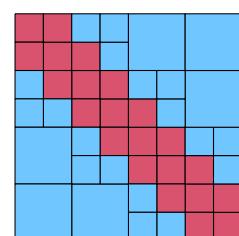
Figure 5.3: Example of a cluster tree generated by bisecting  $I = \{1, 2, \dots, 12\}$ .



(a)  $L = 2$



(b)  $L = 3$



(c)  $L = 4$

Figure 5.4: Block cluster partitioning generated from two binary cluster trees and the admissibility condition  $|i - j| > 1$ . This admissibility can for instance be derived from the  $\eta$ -admissibility of one dimensional meshes. In admissible blocks are colored in red, admissible ones in blue.

For practical applications, we are particularly interested in block cluster trees that are constructed from cluster trees of the row and column indices  $\mathcal{T}_I$  and  $\mathcal{T}_J$ .<sup>6</sup> We demonstrate the construction of a block cluster tree from the cluster trees  $\mathcal{T}_I$  and  $\mathcal{T}_J$ , which we assume to have identical tree structures. Moreover, we require a suitable admissibility condition, which tells us whether the block  $A(I_i^l, J_j^l)$  admits a low-rank approximation. Clearly, this admissibility condition is closely related to the admissibility condition (5.13), (5.14).

Starting from the root node, we construct the block cluster tree recursively. We proceed as follows: If the current block  $I_i^l \times J_j^l$  meets the admissibility condition, we stop the recursion and return the current block. If it does not, we subdivide the current block into  $\{I_c^{l+1} \times J_d^{l+1} : c \in \text{children}(i), d \in \text{children}(j)\}$  and call the routine recursively on these blocks. The subtrees that are returned are rooted at the current node and the resulting tree is returned. Algorithm 5.1 summarizes the procedure. Such a construction

```
procedure BLOCK CLUSTER TREE( $i, j, \mathcal{T}_I, \mathcal{T}_J$ )
    Get row and column clusters  $I_i^l \in \mathcal{T}_I$  and  $J_j^l \in \mathcal{T}_J$ 
    Create tree node  $\mathcal{T}_B$  with  $B = I_i^l \times J_j^l$  at the root
    if  $I_i^l \times J_j^l$  is not admissible then
        for all  $c \in \text{children}(i), d \in \text{children}(j)$  do
            Append BLOCK CLUSTER TREE( $c, d, \mathcal{T}_I, \mathcal{T}_J$ ) to  $\mathcal{T}_B$ 
        end for
    end if
    return  $\mathcal{T}_B$ 
end procedure
```

6: In principle, we may construct more general hierarchical partitionings of  $I \times J$ , which may not necessarily be generated by cluster trees  $\mathcal{T}_I$  and  $\mathcal{T}_J$ . However, to the best of our knowledge, such matrix partitionings are not of practical interest and for the sake of clarity, we chose to omit them.

preserves the overall depth of the cluster tree. In practice, we do not have to store the block cluster tree  $\mathcal{T}_{I \times J}$  as its structure is fully described by  $\mathcal{T}_I$ ,  $\mathcal{T}_J$  as well as the admissibility condition  $\sigma(I_i^l, J_j^l)$ .

**Example 5.2.1** Discrete admissibility condition for the one-dimensional discretization of the one-dimensional Green's function.

$$|i - j| > 1 \quad (5.24)$$

Figure 5.4 shows the block cluster tree that is derived using this admissibility condition.

### 5.3 Hierarchical matrices

This finally brings us to hierarchical matrices, which is the main subject of this chapter:

**Definition 5.3.1 ( $\mathcal{H}$ -matrices)** Let  $\mathbf{A} \in \mathbb{R}^{m \times n}$  be a matrix and  $\mathcal{T}_{I \times J}$  a block cluster tree on  $I = \{1, 2, \dots, m\}, J = \{1, 2, \dots, n\}$ . Moreover, let  $\sigma$  be a suitable admissibility condition and  $k \in \mathbb{N}_0$ . Then, we call  $\mathbf{A}$  a hierarchical matrix (in short  $\mathcal{H}$ -matrix) of maximum rank  $k$ , iff

$$\text{rank } \mathbf{A}(B_i^l) \leq k$$

is satisfied for every admissible block  $B_i^l \in \mathcal{T}_{I \times J} : \sigma(B_i^l)$ .

The obvious advantage of  $\mathcal{H}$ -matrices is that they admit accelerated arithmetic and reduced storage requirements compared to dense matrices, due to the low-rank property of admissible blocks.<sup>7</sup> As such, they are often referred to as rank-structured matrices. Similar to sparse matrices, they exploit structure and data sparsity to improve storage and operation complexity.  $\mathcal{H}$ -matrices provide a great deal of flexibility and represent the most general format among hierarchically rank-structured matrix formats [30, 37, 40].

While  $\mathcal{H}$ -matrices offer flexibility, this comes at the cost of performance and increased difficulty for practical implementation. Without going into much detail on  $\mathcal{H}$ -matrices, we shall move on to an import subset of  $\mathcal{H}$ -matrices, namely *hierarchically off-diagonal low-rank (HODLR)* matrices. As the name implies, these are hierarchical matrices, where the partitioning has been constructed from binary cluster trees with the simple admissibility condition  $i \neq j$  and  $i, j$  are sibling nodes. Such a hierarchical partitioning is illustrated in Figure 5.5 and the format is formalized in Definition 5.3.2

**Definition 5.3.2 (HODLR matrix)** Let  $A \in \mathbb{R}^{m \times n}$  be a matrix and  $\mathcal{T}_I, \mathcal{T}_J$  binary cluster trees with equal tree structures on  $I = \{1, 2, \dots, m\}$ ,  $J = \{1, 2, \dots, n\}$ . Also, let  $k \in \mathbb{N}_0$  be a positive integer. Then, we call  $A$  a hierarchically off-diagonal low-rank matrix (in short HODLR matrix) of off-diagonal rank  $k$ , iff

$$\text{rank } A(I_i^l, J_j^l) \leq k$$

hold for all disting sibling nodes  $I_i^l \in \mathcal{T}_I, J_j^l \in \mathcal{T}_J$ , with  $\text{parent}(i) = \text{parent}(j)$  and  $i \neq j$ . The minimum  $k$  for which this is true is called the HODLR rank of  $A$  (with respect to  $\mathcal{T}_I, \mathcal{T}_J$ ).

Consequently, HODLR matrices allow a simple recursive representation, where diagonal matrix blocks can be represented as

$$A(I_i^l, J_i^l) = A_{ii}^{(l)} = \begin{bmatrix} A_{c_1, c_1}^{(l+1)} & A_{c_1, c_2}^{(l+1)} \\ A_{c_2, c_1}^{(l+1)} & A_{c_2, c_2}^{(l+1)} \end{bmatrix}, \quad (5.25)$$

and  $c_1, c_2$  are the two children nodes of  $i$  in both  $\mathcal{T}_I$  and  $\mathcal{T}_J$ . The diagonal blocks are again HODLR matrices, unless they are smaller than some minimum block size, at which point we simply stop the recursion and store a dense matrix. We call

$$\beta = \max_{I_i^l \in \text{leaves}(\mathcal{T}_I)} |I_i^l| \quad (5.26)$$

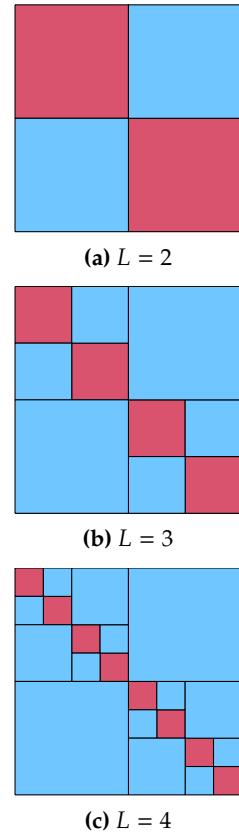
the block size of the partitioning. On the other hand, off-diagonal blocks  $A_{i,j}^{(l)}$  admit a low-rank representation and are stored accordingly:

$$A_{i,j}^{(l)} = \tilde{\mathbf{U}}_i^{(l)} (\tilde{\mathbf{V}}_j^{(l)})^*, \quad (5.27)$$

where  $\tilde{\mathbf{U}}_i^{(l)} \in \mathbb{R}^{|I_i^l| \times k}$ ,  $\tilde{\mathbf{V}}_j^{(l)} \in \mathbb{R}^{|J_j^l| \times k}$  are called its generators.

In a similar fashion to low-rank matrices, we introduce the notion of approximate HODLR matrices:

7: Another prominent example of rank-structured matrices are block low-rank (BLR) matrices, which are partitioned matrices, with the important difference that the partitionings are not hierarchical [39].



**Figure 5.5:** HODLR partitioning generated from two binary cluster trees and the strong admissibility condition  $|i - j| > 0$ . Inadmissible blocks are colored in red, admissible ones in blue.

**Definition 5.3.3** (approximate HODLR matrix) Let  $A \in \mathbb{R}^{m \times n}$ ,  $k \in \mathbb{N}_0$  and  $\epsilon > 0$ . We call  $A$  a matrix of approximate HODLR rank  $k$ , iff for a given block-cluster tree, there exists a HODLR matrix  $\tilde{A}$  with HODLR rank  $k$ , such that

$$\|A - \tilde{A}\| \leq \epsilon \quad (5.28)$$

holds for a suitable norm  $\|\cdot\|$ .

Oftentimes it is more practical to control the error locally, in the sense that

$$\|A(I_i^l, J_j^l) - \tilde{A}(I_i^l, J_j^l)\| \leq \epsilon \quad (5.29)$$

holds true for all low-rank blocks in  $\tilde{A}$ . For a HODLR matrix with  $L$  levels, controlling each block with  $\epsilon$  results in a total approximation error that is bounded by

$$\|A - \tilde{A}\| \leq (2^L - 2)\epsilon.$$

A more interesting choice is to control the relative error locally, such that

$$\|A(I_i^l, J_j^l) - \tilde{A}(I_i^l, J_j^l)\| \leq \epsilon \|A(I_i^l, J_j^l)\|. \quad (5.30)$$

If we choose to do so in the Frobenius norm, (5.30) guarantees

$$\|A - \tilde{A}\|_F \leq \sqrt{2^L - 2}\epsilon. \quad (5.31)$$

## 5.4 Nested bases

Again, let us consider matrices with HODLR block structure. Let  $i, j$  be sibling nodes at level  $l$  and let  $c_1, c_2$  be the children of  $i$  and  $d_1, d_2$  the children of  $j$ . Moreover, let  $\tilde{U}_i^{(l)}$  and  $\tilde{V}_j^{(l)}$  denote the generators of the off-diagonal blocks. Then, we call the bases nested, if the generators can be constructed recursively from its children generators

$$\tilde{U}_i^{(l)} = \begin{bmatrix} \tilde{U}_{c_1}^{(l+1)} & 0 \\ 0 & \tilde{U}_{c_2}^{(l+1)} \end{bmatrix} U_i^{(l)}, \quad (5.32a)$$

$$\tilde{V}_j^{(l)} = \begin{bmatrix} \tilde{V}_{d_1}^{(l+1)} & 0 \\ 0 & \tilde{V}_{d_2}^{(l+1)} \end{bmatrix} V_j^{(l)}, \quad (5.32b)$$

with translation matrices  $U_i^{(l)} \in \mathbb{R}^{2k \times k}$  and  $V_j^{(l)} \in \mathbb{R}^{2k \times k}$ .<sup>8</sup> The low-rank block  $A(I_i^l, J_j^l)$  can then be factored as

$$A(I_i^l, J_j^l) = \tilde{U}_i^{(l)} \tilde{B}_{i,j}^{(l)} (\tilde{V}_j^{(l)})^*, \quad (5.33)$$

where we have introduced the block  $\tilde{B}_{i,j}^{(l)} \in \mathbb{R}^{k \times k}$  to decouple it from the bases  $\tilde{U}_i^{(l)}$  and  $\tilde{V}_j^{(l)}$ . In our notation, generators with a  $\sim$  denote “tall” matrices, with the leading dimension corresponding to the blocksize of the low-rank block (5.33). Therefore, to construct the low-rank block (5.33), we have to first construct generators, that span its row- and column-space. To do so, we apply the nestedness property (5.32) recursively until the leaf level is reached. At the leaf level, we set  $U_i^{(L)} = \tilde{U}_i^{(L)}$  and  $V_j^{(L)} = \tilde{V}_j^{(L)}$ .

8: Due to this additional hierarchy of the bases,  $\mathcal{H}$ -matrices with nested bases are referred to as  $\mathcal{H}^2$ -matrices in the literature [40].

HODLR matrices with such an additional hierarchy in their generators are called *hierarchically semi-separable (HSS)* matrices. A formal definition follows later in Definition 5.4.1.<sup>9</sup> The nestedness of generators allows us to represent the HSS matrix  $A$  in a recursive manner. We define

$$\mathbf{B}_i^{(l)} = \begin{bmatrix} 0 & \tilde{\mathbf{B}}_{c_1, c_2}^{(l+1)} \\ \tilde{\mathbf{B}}_{c_2, c_1}^{(l+1)} & 0 \end{bmatrix},$$

$$\mathbf{D}_i^{(L)} = A(I_i^L, J_i^L),$$

as well as

$$\mathbf{U}^{(l)} = \text{diag}(\mathbf{U}_1^{(l)}, \mathbf{U}_2^{(l)}, \dots), \quad (5.34a)$$

$$\mathbf{V}^{(l)} = \text{diag}(\mathbf{V}_1^{(l)}, \mathbf{V}_2^{(l)}, \dots), \quad (5.34b)$$

$$\mathbf{B}^{(l)} = \text{diag}(\mathbf{B}_1^{(l)}, \mathbf{B}_2^{(l)}, \dots), \quad (5.34c)$$

$$\mathbf{D}^{(L)} = \text{diag}(\mathbf{D}_1^{(L)}, \mathbf{D}_2^{(L)}, \dots). \quad (5.34d)$$

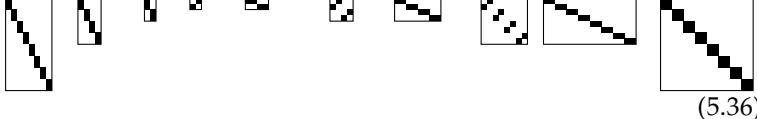
Then  $A$  can be expressed via the recursion

$$A^{(0)} = \mathbf{B}^{(0)}, \quad (5.35a)$$

$$A^{(l)} = \mathbf{U}^{(l)} A^{(l-1)} (\mathbf{V}^{(l)})^* + \mathbf{B}^{(l)} \quad \text{for } l = 1, 2, \dots, L-1 \quad (5.35b)$$

$$A^{(L)} = \mathbf{U}^{(L)} A^{(L-1)} (\mathbf{V}^{(L)})^* + \mathbf{D}^{(L)}. \quad (5.35c)$$

By writing out this recursion, we obtain a telescoping factorization. For a 3-level HSS matrix with balanced cluster trees, this yields

$$A = \mathbf{U}^{(3)} \left( \mathbf{U}^{(2)} \left( \mathbf{U}^{(1)} \mathbf{B}^{(0)} (\mathbf{V}^{(1)})^* + \mathbf{B}^{(1)} \right) (\mathbf{V}^{(2)})^* + \mathbf{B}^{(2)} \right) (\mathbf{V}^{(3)})^* + \mathbf{D}^{(3)},$$

(5.36)

where the structure of each matrix is illustrated below its symbol. A matrix  $A$  in HSS format is therefore fully defined by specifying its row- and column-cluster trees  $\mathcal{T}_I, \mathcal{T}_J$ , as well as the matrices that appear in (5.34).

This telescoping factorization reveals an alternative definition of HSS matrices. We observe in (5.36), that the first row of  $A - D^{(3)}$  is spanned by  $\mathbf{U}_1^{(3)}$ . Moreover, we see that this holds for all levels  $A^{(l)}$  of the hierarchical definition (5.35), disregarding the diagonal part. Thus, by requiring that each HSS block row

$$A(I_i^l, J \setminus J_i^l)$$

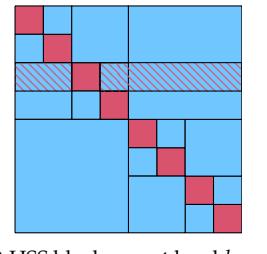
and HSS block column

$$A(I \setminus I_j^l, J_j^l)$$

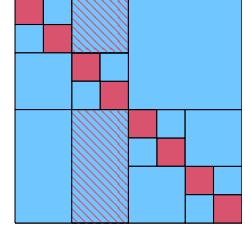
is low-rank, we can ensure that  $A$  is a HSS matrix. Figure 5.7 illustrates HSS block rows and columns. Consequently, we define:

**Definition 5.4.1 (HSS matrix)** Consider a matrix  $A \in \mathbb{R}^{m \times n}$  with row indices  $I$ , column indices  $J$  and corresponding row- and column-cluster trees  $\mathcal{T}_I, \mathcal{T}_J$ , with matching tree structure.

9: The structure of hierarchically semi-separable matrices is related to separable and semi-separable matrices [41].



(a) HSS block row at level  $l = 4$



(b) HSS block column at level  $l = 3$

Figure 5.7: Illustration of a HSS block row and a block column.

- We select a row partition  $I_i^l \in \mathcal{T}_I$  at level  $l$ . Then the block row  $\mathbf{A}(I_i^l, J \setminus J_i^l)$ , which omits the diagonal part is called a HSS block row. Similarly, we call  $\mathbf{A}(I \setminus I_j^l, J_j^l)$  a HSS block column.
- We call  $\mathbf{A}$  a HSS matrix with respect to  $\mathcal{T}_I, \mathcal{T}_J$ , if there exists a positive integer  $k \in \mathbb{N}_0$ , such that the rank of every HSS block row and block column is smaller than, or equal to  $k$ :

$$\forall I_i^l \in \mathcal{T}_I : \text{rank } \mathbf{A}(I_i^l, J \setminus J_i^l) \leq k, \quad (5.37a)$$

$$\forall J_j^l \in \mathcal{T}_J : \text{rank } \mathbf{A}(I \setminus I_j^l, J_j^l) \leq k. \quad (5.37b)$$

We call the minimum  $k$  for which (5.37) is satisfied, the HSS rank of  $\mathbf{A}$ .

We remark that the HSS rank as introduced here differs from the HODLR rank, which only takes the rank of off-diagonal blocks into account. This definition resembles the *quasi-separable rank*, which is the maximum rank of any block which lies strictly in the upper or lower triangular part of the matrix.<sup>10</sup> In practice, we are often more interested in the ranks of the matrices (5.34), required to represent the matrix. In these cases we refer to the HODLR rank.

As for HODLR matrices, we are interested in matrices of approximate HSS rank  $k$ :

**Definition 5.4.2** (approximate HSS matrix) Let  $\mathbf{A} \in \mathbb{R}^{m \times n}$ ,  $k \in \mathbb{N}_0$  and  $\epsilon > 0$ . We call  $\mathbf{A}$  a matrix of approximate HSS rank  $k$ , iff for a given block-cluster tree, there exists a HSS matrix  $\tilde{\mathbf{A}}$  with HSS rank  $k$ , such that

$$\|\mathbf{A} - \tilde{\mathbf{A}}\| \leq \epsilon \quad (5.38)$$

holds for a suitable norm  $\|\cdot\|$ .

It is useful to bound the approximation error by bounding it locally. For HSS matrices, this is done by controlling the approximation error for each individual HSS block row and column. Controlling the relative error means enforcing

$$\begin{aligned} \|\mathbf{A}(I_i^l, J \setminus J_i^l) - \tilde{\mathbf{A}}(I_i^l, J \setminus J_i^l)\| &\leq \epsilon \|\mathbf{A}(I_i^l, J \setminus J_i^l)\| \\ \|\mathbf{A}(I \setminus I_j^l, J_j^l) - \tilde{\mathbf{A}}(I \setminus I_j^l, J_j^l)\| &\leq \epsilon \|\mathbf{A}(I \setminus I_j^l, J_j^l)\| \end{aligned}$$

for each block row and column. For the Frobenius norm, this results in the tighter error bound

$$\|\mathbf{A} - \tilde{\mathbf{A}}\|_F \leq \sqrt{2^{L-1}} \epsilon \quad (5.39)$$

as compared to (5.30).

Tighter error bounds for the spectral norm are reported in [42], however these bounds require that  $\mathbf{A}$  and  $\tilde{\mathbf{A}}$  share the same generators. This is the case if a truncated SVD is used to compute the approximation. In general, however, it is not the case and it is unclear whether improved error bounds can be found without making assumptions on the generators [42, 43]. Conversely, it is also possible to show the existence of a HSS rank  $k$  approximant  $\tilde{\mathbf{A}}$ , which satisfies  $\|\mathbf{A} - \tilde{\mathbf{A}}\|_2 \leq \sqrt{2^{L+1}} - 4\epsilon$  if each HSS block row and column of  $\mathbf{A}$  allow a rank- $k$  truncation with an error smaller

10: Quasiseparable matrices can be understood as HSS matrices with leaf size 1 and HSS rank 1 [41]. These matrices include tridiagonal and semi-separable matrices.

than  $\epsilon$  [44]. This however, does not guarantee that we have found this approximant.

# Algorithms for hierarchical matrices

# 6

The focus of this chapter is to introduce some of the many useful algorithms for hierarchical matrices, which will be useful subsequently. We keep our focus on hierarchical matrices with HODLR/HSS structures. Many of the algorithms for HODLR matrices can be derived by formulating algorithms for block matrices of the form

$$A = \begin{bmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{bmatrix} \quad (6.1)$$

and then modifying them to be recursive in  $A_{11}$  and  $A_{22}$ . Section 6.1 introduces a number of such algorithms to perform arithmetic using HODLR matrices. These algorithms serve as a baseline for the HSS arithmetic presented in Section 6.2.

## 6.1 HODLR arithmetic

We introduce some algorithms for performing arithmetic with HODLR matrices. One of the most basic, yet essential operations is the computation of the matrix-vector product

$$\mathbf{x} \rightarrow A\mathbf{x},$$

where  $A$  is a matrix partitioned according to (6.1) and  $\mathbf{x}$  a vector of similar size. Then, Algorithm 6.1 computes the matrix vector product in a straight-forward manner. As mentioned earlier, we can modify the

```

procedure BLOCK MATVEC( $A, \mathbf{x}$ )
    Partition  $\mathbf{x} = [\mathbf{x}_1, \mathbf{x}_2]^\top$ 
     $y_1 \leftarrow A_{11}\mathbf{x}_1$ 
     $y_1 \leftarrow y_1 + A_{12}\mathbf{x}_2$ 
     $y_2 \leftarrow A_{22}\mathbf{x}_2$ 
     $y_2 \leftarrow y_2 + A_{21}\mathbf{x}_1$ 
    return  $\mathbf{y} = [y_1, y_2]^\top$ 
end procedure

```

6.1 HODLR arithmetic . . . . .	51
6.2 HSS arithmetic . . . . .	53
Matrix-vector multiplication	53
ULV factorization and solver	54
6.3 HSS compression . . . . .	57
Direct compression . . . . .	57
Randomized compression .	60
6.4 HssMatrices.jl . . . . .	63

**Algorithm 6.1:** A simple algorithm for the multiplication of a two-by-two block matrix  $A$  with a vector  $\mathbf{x}$ .

above algorithm to be called recursively on the diagonal blocks. This yields Algorithm 6.2. We would like to know the computational cost of Algorithm 6.2 before we proceed. To determine this, we need to make some assumptions regarding the HODLR structure of  $A$ . Firstly, we assume that  $A$  is square and of order  $n$ , i.e.  $A \in \mathbb{C}^{n \times n}$ . Moreover, we assume that the HODLR rank is  $k$ , i.e., the rank of all off-diagonal blocks is bounded by  $k$ . Finally we have to make an assumption about the depth of the block cluster tree. As it is unreasonable to continue the hierarchical partitioning once the size of leaf partitions match the rank  $k$ , we assume that

$$2^{L-1} \approx \frac{n}{k}. \quad (6.2)$$

```

procedure HODLR MATVEC( $A, x$ )
  if  $A$  is in HODLR format then
    Partition  $x = [x_1, x_2]^\top$ 
     $y_1 \leftarrow$  HODLR MATVEC( $A_{11}, x_1$ )
     $y_1 \leftarrow y_1 + A_{12}x_2$ 
     $y_2 \leftarrow$  HODLR MATVEC( $A_{22}, x_2$ )
     $y_2 \leftarrow y_2 + A_{21}x_1$ 
    return  $y = [y_1, y_2]^\top$ 
  else
    return  $y = Ax$ 
  end if
end procedure

```

**Algorithm 6.2:** Algorithm for matrix-vector multiplication of a HODLR matrix  $A$  with a vector  $x$ .

Then, the total amount of work (number of FLOPs)  $W(n, k)$  to execute Algorithm 6.2 amounts to

$$\begin{aligned} W(n, k) &= \sum_{l=1}^L 2 \cdot 2^{l-1} \frac{n}{2^{l-1}} k + \frac{n^2}{2^{L-1}} \\ &= 2(L-1)nk + \frac{n^2}{2^{L-1}} = 2nk \log \frac{n}{k} + nk = \mathcal{O}(kn \log n). \quad (6.3) \end{aligned}$$

This is an encouraging result, considering the  $\mathcal{O}(n^2)$  cost of matrix-vector multiplication with a dense matrix.

Similarly, we construct an algorithm for the addition of two HODLR matrices  $A$  and  $B$  with identical block structures. Algorithm 6.3 reveals

```

procedure HODLR ADD( $A, B$ )
  if  $A$  and  $B$  are in HODLR format then
     $C_{11} \leftarrow$  HODLR ADD( $A_{11}, B_{11}$ )
     $C_{22} \leftarrow$  HODLR ADD( $A_{22}, B_{22}$ )
    Perform low-rank addition  $C_{12} \leftarrow B_{12} + B_{12}$ 
    Perform low-rank addition  $C_{21} \leftarrow B_{12} + B_{21}$ 
    return  $C = \begin{bmatrix} C_{11} & C_{12} \\ C_{21} & C_{22} \end{bmatrix}$ 
  else
    return  $C = A + B$ 
  end if
end procedure

```

**Algorithm 6.3:** Algorithm for the addition of two HODLR matrices  $A$  and  $B$  with identical block structure.

one of the main caveats of hierarchical matrix, that is, ranks tend to grow, when arithmetic is performed. The simple addition of two low-rank matrices of rank  $k$ , in Algorithm 6.3 results in a low-rank matrix of rank  $2k$ . In practice however, many matrices retain their low-rank property after arithmetic is performed. Thus, we may need to perform recompression on each off-diagonal block to maintain the computational efficiency of our methods.

Other algorithms for matrix arithmetic using HODLR matrices can be formulated in a similar fashion [25]. For instance, we can formulate algorithms for computing  $AB$  or  $A^{-1}B$ , where  $A$  and  $B$  are HODLR matrices with compatible clusters, i.e., the column cluster tree of  $A$  should match the row cluster tree of  $B$ .

An algorithm for the inversion of a two-by-two block matrix can be achieved using the block-LDR factorization (3.1) and the Schur complement  $S_{11} = A_{11} - A_{12}A_{22}^{-1}A_{21}$ .<sup>1</sup> Algorithm 6.4 outlines such an algorithm were

```

procedure HODLR INVERSE( $A$ )
  if  $A$  is in HODLR format then
     $X_{22} \leftarrow$  HODLR INVERSE( $A_{22}$ )
     $X_{11} \leftarrow$  HODLR INVERSE( $A_{11} - A_{12}X_{22}A_{21}$ )
     $C \leftarrow \begin{bmatrix} X_{11} & -X_{11}A_{12}X_{22} \\ -X_{22}A_{21}X_{11} & X_{22} + X_{22}A_{21}X_{11}A_{12}X_{22} \end{bmatrix}$ 
    Recompress bottom-right block of  $C$ 
    return  $C$ 
  else
    return  $A^{-1}$ 
  end if
end procedure
```

<sup>1</sup>: In the majority of the literature the block-LDR factorization (3.1) is referred to as block-LU factorization.

Table 6.1 provides an overview of the computational complexities of HODLR arithmetic involving either two HODLR matrices or a HODLR matrix and a dense vector [45]. We observe that all operations are quasilinear with dependencies of either  $n \log n$  or  $n \log^2 n$ , if we assume the ranks to be a small constant  $k \ll n$ .

## 6.2 HSS arithmetic

The log-factors in the complexity of HODLR algorithms are a consequence of the nested tree structure, as we would expect for algorithms involving tree structures. Nested bases allow to improve upon the quasilinear complexities to achieve true linear complexity (assuming that off-diagonal ranks  $k$  are constant). Table 6.2 lists the computational cost of various arithmetic operations involving HSS matrices. To showcase how the nested bases can be exploited, we present two of the algorithms, matrix-vector multiplication and solving linear systems involving HSS matrices.

### HSS matrix-vector multiplication

To formulate an algorithm for HSS matrix-vector multiplications, we consider the recursive definition (5.35) of a HSS matrix  $A$  and apply it to a vector  $x$ . Going from the bottom up, we can split the product  $b = Ax$  into the sequence

$$y^{(L)} = (V^{(L)})^* x, \quad (6.4a)$$

$$y^{(l)} = (V^{(l)})^* y^{(l+1)}, \quad \text{for } l = L-1, L-2, \dots, 1 \quad (6.4b)$$

$$z^{(1)} = B^{(0)} y^{(1)}, \quad (6.4c)$$

$$z^{(l)} = U^{(l-1)} z^{(l-1)} + B^{(l-1)} y^{(l)}, \quad \text{for } l = 2, \dots, L \quad (6.4d)$$

$$b = Ax = U^{(L)} z^{(L)} + D^{(L)} x, \quad (6.4e)$$

**Algorithm 6.4:** Algorithm to compute the inverse of the HODLR matrix  $A$ , assuming that it is.

**Table 6.1:** Computational complexity of common operations using HODLR matrices.  $A$  and  $B$  denote HODLR matrices of order  $n$  with compatible cluster trees and maximal HODLR rank  $k$ .  $x$  denotes a vector of suitable size.

operation	complexity
$x \rightarrow Ax$	$\mathcal{O}(kn \log n)$
$x \rightarrow A^{-1}x$	$\mathcal{O}(k^2 n \log^2 n)$
$B \rightarrow A + B$	$\mathcal{O}(k^2 n \log n)$
$B \rightarrow AB$	$\mathcal{O}(k^2 n \log^2 n)$
$B \rightarrow A^{-1}B$	$\mathcal{O}(k^2 n \log n)$

**Table 6.2:** Computational complexity of arithmetic using HSS matrices.  $A$  and  $B$  denote HSS matrices of order  $n$  with compatible cluster trees and maximum HSS rank  $k$ .  $x$  denotes a vector of suitable size.

operation	complexity
$x \rightarrow Ax$	$\mathcal{O}(kn)$
$x \rightarrow A^{-1}x$	$\mathcal{O}(k^2 n)$
$B \rightarrow A + B$	$\mathcal{O}(k^2 n)$
$B \rightarrow AB$	$\mathcal{O}(k^2 n)$
$B \rightarrow A^{-1}B$	$\mathcal{O}(k^2 n)$

where (6.4a)-(6.4b) traverse the HSS tree structure from the bottom up and (6.4c)-(6.4e) do so from top to bottom. Algorithm 6.5 implements this scheme while simultaneously exploiting the blockdiagonal structure of the involved matrices.

```

procedure HSS MATVEC( $A, x$ )
  for all nodes  $j$  from the bottom-up do
    if  $j$  is a leaf node then
      Set  $y_j^{(L)} \leftarrow (V_j^{(L)})^* x(J_j^{(L)})$ 
    else
      Compute  $y_j^{(l)} \leftarrow (V_j^{(l)})^* \begin{bmatrix} y_{c_1}^{(l-1)} \\ y_{c_2}^{(l-1)} \end{bmatrix}$ 
    end if
  end for
  for all nodes  $j$  from top to bottom do
    if  $j$  is the root node then
      Compute  $\begin{bmatrix} z_{c_1}^{(l+1)} \\ z_{c_2}^{(l+1)} \end{bmatrix} \leftarrow \begin{bmatrix} \mathbf{0} & \tilde{B}_{c_1, c_2}^{(l+1)} \\ \tilde{B}_{c_1, c_1}^{(l+1)} & \mathbf{0} \end{bmatrix} \begin{bmatrix} y_{c_1}^{(l+1)} \\ y_{c_2}^{(l+1)} \end{bmatrix}$ 
    else if  $j$  is not a leaf node then
      Compute  $\begin{bmatrix} z_{c_1}^{(l+1)} \\ z_{c_2}^{(l+1)} \end{bmatrix} \leftarrow U_j^{(l)} z_j^{(l)} + \begin{bmatrix} \mathbf{0} & \tilde{B}_{c_1, c_2}^{(l+1)} \\ \tilde{B}_{c_1, c_1}^{(l+1)} & \mathbf{0} \end{bmatrix} \begin{bmatrix} y_{c_1}^{(l+1)} \\ y_{c_2}^{(l+1)} \end{bmatrix}$ 
    end if
  end for
  for all leaf nodes  $j$  do
    Set  $b(I_j^L) \leftarrow U_j^{(L)} + D_j^{(L)} x(J_j^L)$ 
  end for
  return  $b$ 
end procedure

```

**Algorithm 6.5:** Efficient algorithm for matrix-vector computation  $x \rightarrow Ax$  with a HSS matrix  $A$ .  $c_1, c_2$  denote the indices pointing to the children elements of  $j$ . An implementation can be found in [46].

We determine the computational work required to compute the HSS matrix-vector product as for Algorithm 6.5. Assuming that matrix-vector multiplication with a dense matrix is a  $\mathcal{O}(mn)$  operation, summing all operations in (6.4) yields a total work of

$$\begin{aligned}
W(n, k) &\sim 2^{L-1}k \frac{n}{2^{L-1}} + \sum_{l=1}^{L-1} 2^{l-1}k^2 = nk + k^2 \frac{2^{L-1}-1}{2-1} \sim nk \\
&= \mathcal{O}(kn).
\end{aligned} \tag{6.5}$$

Thus, we have eliminated the log-factor in the computational complexity courtesy of the nested bases and the resulting re-use of matrix-vector products. A parallel can be drawn to fast-multipole methods where such nestedness properties are exploited to obtain linear complexity algorithms [25, 31].

## ULV factorization and solver

As we are solving large linear systems, one of the central arithmetic operations will be the action of the inverse on a vector  $x$ , i.e.  $x \rightarrow A^{-1}x$ , as well as the application of the inverse on a HSS matrix  $B$  with compatible clustering,  $B \rightarrow A^{-1}B$ . The core idea has been presented in [47] and, a modification for the application to HSS matrices has later been presented

in [45]. We outline the basic idea of the former and refer the reader to [47] for a detailed description of the algorithm. We use a balanced HSS matrix as depicted in Figure 6.1a. The algorithm is recursive and operates in two modes.

For the first mode, let us assume that we are situated at the leaf level  $L$  of the HSS matrix. We observe that the column generators  $\mathbf{U}_i^{(L)}$  at this level span the HSS block rows. We assume that  $k_i$ , the number of columns of  $\mathbf{U}_i^{(L)}$  is strictly smaller than the number of rows  $m_i$ . Then, by forming a QL-decomposition<sup>2</sup> of  $\mathbf{U}_i^{(L)}$ , we find an orthogonal transform  $\mathbf{Q}_i^{(L)}$ , such that

$$\bar{\mathbf{U}}_i^{(L)} = (\mathbf{Q}_i^{(L)})^* \mathbf{U}_i^{(L)} = \begin{bmatrix} \mathbf{0} \\ \hat{\mathbf{U}}_i^{(L)} \end{bmatrix}, \quad (6.6)$$

which introduces  $m_i - k_i$  zero rows into the HSS block row. This situation is illustrated in Figure 6.1b. Simultaneously, the right-hand side  $\mathbf{b}_i^{(L)} = \mathbf{b}(J_i^L)$  has been modified to

$$(\mathbf{Q}_i^{(L)})^* \mathbf{b}_i^{(L)} = \begin{bmatrix} \check{\mathbf{b}}_i^{(L)} \\ \star \end{bmatrix}, \quad (6.7)$$

where we have again exposed the first  $m_i - k_i$  rows  $\check{\mathbf{b}}_i^{(L)}$ . For each of the modified diagonal blocks  $(\mathbf{Q}_i^{(L)})^* \mathbf{D}_i^{(L)}$ , we now compute its LQ-factorization, which yields the orthogonal transform  $\mathbf{W}_i^{(L)}$ , such that

$$\bar{\mathbf{D}}_i^{(L)} = (\mathbf{Q}_i^{(L)})^* \mathbf{D}_i^{(L)} (\mathbf{W}_i^{(L)})^* = \begin{bmatrix} \bar{\mathbf{D}}_{i,1,1}^{(L)} & \mathbf{0} \\ \bar{\mathbf{D}}_{i,2,1}^{(L)} & \bar{\mathbf{D}}_{i,2,2}^{(L)} \end{bmatrix}. \quad (6.8)$$

As for the matrix on the right, it has been partitioned to expose the first  $(m_i - k_i) \times (m_i - k_i)$  block as in (6.6). To account for the action of  $\mathbf{W}_i^{(L)}$  on the off-diagonal blocks, we can simply multiply it onto the shared generator of the row space of the HSS block column  $\mathbf{V}_i^{(L)}$ , which yields

$$\bar{\mathbf{V}}_i^{(L)} = \mathbf{W}_i^{(L)} \mathbf{V}_i^{(L)} = \begin{bmatrix} \check{\mathbf{V}}_i^{(L)} \\ \hat{\mathbf{V}}_i^{(L)} \end{bmatrix}. \quad (6.9)$$

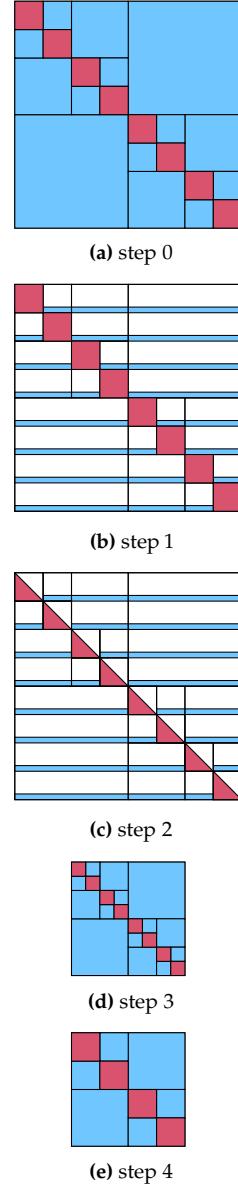
This transformation has to be taken into account for the vector of unknowns  $\check{\mathbf{x}}_i^{(L)} = \mathbf{x}(J_i^L)$ , and we write

$$\mathbf{W}_i^{(L)} \check{\mathbf{x}}_i^{(L)} = \begin{bmatrix} \check{\mathbf{x}}_i^{(L)} \\ \hat{\mathbf{x}}_i^{(L)} \end{bmatrix}, \quad (6.10)$$

where  $\check{\mathbf{x}}_i^{(L)}$  corresponds to the first  $m_i - k_i$  rows as usual. Figure 6.1c depicts the matrix  $\mathbf{A}$  after the application of  $\mathbf{W}_i^{(L)}$  at each leaf node. Due to the introduction of the zero rows, the linear system corresponding to the first  $m_i - k_i$  rows at each node is

$$\bar{\mathbf{D}}_{i,1,1}^{(L)} \check{\mathbf{x}}_i^{(L)} = \check{\mathbf{b}}_i^{(L)}, \quad (6.11)$$

which we can solve for  $\check{\mathbf{x}}_i^{(L)}$  through back-substitution. At this stage, we have to update the right-hand side  $\check{\mathbf{b}}_i^{(L)}$ , by multiplying the action of all  $\check{\mathbf{x}}_i^{(L)}$  with the corresponding blocks of the modified matrix  $\mathbf{A}$ , which has



**Figure 6.1:** Illustration of the ULV factorization algorithm for HSS matrices. White blocks indicate zero blocks. Steps 0–3 show the reduction procedure if off-diagonal blocks are compressible. If they are not, leaf nodes are merged as illustrated in step 4. The matrices in step 3 and 4 are not scaled to size.

2: The QL- and LQ-decompositions here are related to the QR-factorization (2.14), where  $\mathbf{L}$  is a lower triangular matrix. These factorizations can be computed with slight modifications to the algorithms for computing the QR-factorization.

the form

$$\text{diag}(\mathbf{Q}_i^{(L)})^* \mathbf{A} \text{diag}(\mathbf{W}_i^{(L)})^*. \quad (6.12)$$

Because the block diagonal matrices conform to the cluster of  $\mathbf{A}$ , we observe that this is just another HSS matrix. We introduce the vector  $\check{\mathbf{x}}$ , which holds  $\check{x}_i^{(L)}$  at the entries  $J_i^L$  and zero everywhere else. Then, we can compute the updated right-hand side

$$\bar{\mathbf{b}} = \text{diag}(\mathbf{Q}_i^{(L)})^* \mathbf{b} - \text{diag}(\mathbf{Q}_i^{(L)})^* \mathbf{A} \text{diag}(\mathbf{W}_i^{(L)})^* \check{\mathbf{x}} \quad (6.13)$$

using the HSS matrix-vector multiplication. Ideally this is done in a way which exploits the zero-blocks that appear in the product. We can disregard the first  $m_i - k_i$  rows and form a new linear system

$$\hat{\mathbf{A}} \hat{\mathbf{x}} = \hat{\mathbf{b}}, \quad (6.14)$$

where we have stacked the remaining non-zero rows at each node to form  $\hat{\mathbf{A}}$ ,  $\hat{\mathbf{x}}$  and  $\hat{\mathbf{b}}$ . In particular, we have

$$\bar{\mathbf{b}} = \begin{bmatrix} \star \\ \hat{\mathbf{b}}_i^{(L)} \end{bmatrix}, \quad (6.15)$$

where  $\hat{\mathbf{b}}_i^{(L)}$  denotes the  $k_i$  rows that are to be passed on to the next step. The matrix  $\hat{\mathbf{A}}$  denotes the modified HSS matrix with diagonal blocks  $\tilde{\mathbf{D}}_{i,2,2}^{(L)}$  and generators  $\tilde{\mathbf{U}}_i^{(L)}, \tilde{\mathbf{V}}_i^{(L)}$  on the bottom level. On the other hand, the blocks  $\mathbf{B}_{i,j}^{(L)}, \mathbf{U}_i^{(L)}, \mathbf{V}_i^{(L)}$  at higher levels in the hierarchy remain unchanged. The resulting linear system is illustrated in Figure 6.1d. Once the remaining system is solved and  $\hat{\mathbf{x}}$  is known, we can recover the solution  $\mathbf{x}$  using (6.10).

This brings us to the second mode, in which  $k_i$  is equal or larger than  $m_i$ . In this case, we can not further reduce  $\mathbf{A}$ . Instead, we merge the leaf nodes by setting

$$\mathbf{D}_i^{(L-1)} \leftarrow \begin{bmatrix} \mathbf{D}_{c_1}^{(L)} & \mathbf{U}_{c_1}^{(L)} \tilde{\mathbf{B}}_{c_1, c_2}^{(L)} (\mathbf{V}_{c_2}^{(L)})^* \\ \mathbf{U}_{c_2}^{(L)} \tilde{\mathbf{B}}_{c_2, c_1}^{(L)} (\mathbf{V}_{c_1}^{(L)})^* & \mathbf{D}_{c_2}^{(L)} \end{bmatrix}, \quad (6.16a)$$

$$\mathbf{U}_i^{(L-1)} \leftarrow \begin{bmatrix} \mathbf{U}_{c_1}^{(L)} \\ \mathbf{U}_{c_2}^{(L)} \end{bmatrix} \mathbf{U}_i^{(L-1)}, \quad (6.16b)$$

$$\mathbf{V}_i^{(L-1)} \leftarrow \begin{bmatrix} \mathbf{V}_{c_1}^{(L)} \\ \mathbf{V}_{c_2}^{(L)} \end{bmatrix} \mathbf{V}_i^{(L-1)}, \quad (6.16c)$$

for each node  $i$  at level  $L - 1$  with children  $c_1$  and  $c_2$ , which yields a modified HSS matrix with one less level. This is illustrated in Figure 6.1e. The algorithm proceeds by restarting the recursion and by checking whether the modified system is reducible. If the matrix is simply one diagonal block, the recursion terminates and the result is computed using dense arithmetic. Algorithm 6.6 summarizes all of the above in a high-level overview.

Starting from this, various algorithms can be formulated. By skipping the application to the right-hand side (6.13) and the solve steps, we can formulate an algorithm, which implicitly forms the inverse  $\mathbf{A}^{-1}$  to be applied later. More importantly, we can formulate an algorithm which

```

procedure HSS SOLVE( $A, b$ )
  if  $A$  is reducible then
    for all leaf nodes  $i$  do
      Reduce to ULV form according to Equations (6.6) to (6.9)
    end for
    Compute the updated right-hand side  $\tilde{b}$  as in Equation 6.13
    Form  $\hat{A}\hat{x} = \tilde{b}$  according to Equation 6.14
    Call  $\hat{x} \leftarrow$  HSS SOLVE( $\hat{A}, \tilde{b}$ )
    Compute  $x$  from  $\hat{x}$  according to Equation 6.10
    return  $x$ 
  else if  $A$  is a full matrix then
    return  $A^{-1}b$ 
  else
    Prune leaf nodes of  $A$  according to Equation 6.16
    Call  $x \leftarrow$  HSS SOLVE( $A, b$ )
    return  $x$ 
  end if
end procedure

```

**Algorithm 6.6:** Solves the linear system where  $A$  is a HSS matrix and  $b$  a vector using the implicit ULV factorization. An implementation can be found in [46].

applies the inverse  $A^{-1}$  to another HSS matrix with compatible tree clusters, in the sense that  $A$  and  $B$  have identical row cluster trees [45]. Implementations of this algorithm and the efficient solver can be found in [45, 46].

## 6.3 HSS compression

The efficient arithmetic that we have discussed so far becomes irrelevant, unless we have access to efficient methods for computing the hierarchical representations of the relevant matrices. There are two main approaches when it comes to constructing such representations. The first one is analytical, in the sense that the representation can be constructed from the underlying analytical expressions of the continuous integral operator [48]. Such methods are often used in the literature on boundary element methods and  $\mathcal{H}^2$ -matrices [40]. Such approaches usually use analytical expansions such as the multipole expansion to find suitable representations [40, 48]. This is very problem-specific and generally inapplicable, especially in the case of weak admissibility conditions, where such expansions cannot be computed. In many cases, one may suspect that the matrix can be represented in a hierarchical matrix format, but without a proof for this. Compression methods aim to find the representation algebraically, by directly attempting to compute the hierarchical matrix representation [49]. We present two different algorithms to compute HSS representations of a matrix.

### Direct compression

The first method for compression is a direct method, much in the same way that a QR-factorization can be used to compute low-rank representations. Our treatment closely follows the discussion in [49], where the method was first described.

To understand the direct compression algorithm, we start with a concrete example of a matrix  $A$ , that we partition into a 4 by 4 block matrix

$$A = \begin{bmatrix} A_{11} & A_{12} & A_{14} & A_{15} \\ A_{21} & A_{22} & A_{24} & A_{25} \\ A_{41} & A_{42} & A_{44} & A_{45} \\ A_{51} & A_{52} & A_{54} & A_{55} \end{bmatrix}, \quad (6.17)$$

which is numbered according to the post-ordering of the HSS tree illustrated in Figure 6.2.

Furthermore, we introduce the notation  $A_{3,:}$  to denote the block row corresponding to node 3, i.e.

$$A_{3,:} = \begin{bmatrix} A_{14} & A_{15} \\ A_{24} & A_{25} \end{bmatrix},$$

and so on. In a first step, we determine all the diagonal blocks  $D_1 = A_{11}$ ,  $D_2 = A_{22}$ , etc., and set

$$D = \text{diag}(D_1, D_2, D_3, D_4). \quad (6.18)$$

Then, we start with node 1 at the leaf level. Using a pivoted QR-factorization, we can extract a column-space of the first HSS block row

$$U_1 [U_1^* A_{12} \quad U_1^* A_{14} \quad U_1^* A_{15}] = U_1 [\tilde{A}_{12} \quad \tilde{A}_{14} \quad \tilde{A}_{15}]. \quad (6.19)$$

For an adaptive algorithm, this is done with a rank-revealing QR factorization, such that the numerical rank of the HSS block row is revealed. As  $U_1$  has the dimensions  $m_1 \times k_1$ , the blocks  $\tilde{A}_{12}, \tilde{A}_{14}, \tilde{A}_{15}$  are effectively compressed to have a leading dimension of  $k_1$ . We repeat the same procedure for the first HSS block column, to extract a row space  $V_1$ , such that

$$V_1 [V_1^* A_{21}^* \quad V_1^* A_{41}^* \quad V_1^* A_{51}^*] = V_1 [\tilde{A}_{21}^* \quad \tilde{A}_{41}^* \quad \tilde{A}_{51}^*]. \quad (6.20)$$

After processing node 1,  $A$  can be written as

$$A = \begin{bmatrix} U_1 & & \\ & I & \\ & & I \end{bmatrix} \begin{bmatrix} 0 & \tilde{A}_{12} & \tilde{A}_{14} & \tilde{A}_{15} \\ \tilde{A}_{21} & 0 & A_{24} & A_{25} \\ \tilde{A}_{41} & A_{42} & 0 & A_{45} \\ \tilde{A}_{51} & A_{52} & A_{54} & 0 \end{bmatrix} \begin{bmatrix} V_1^* & & \\ & I & \\ & & I \end{bmatrix} + D.$$

We observe that we can safely ignore any previously constructed bases while proceeding through the compression. Repeating the procedure for node 2 and taking into account previously compressed parts, we compute  $U_2, V_2$  from the compressed HSS block rows and column corresponding to node 2. This yields

$$\begin{aligned} U_2 [U_2^* \tilde{A}_{21} &\quad U_2^* A_{24} \quad U_2^* A_{25}] = U_2 [\tilde{B}_{21} \quad \tilde{A}_{24} \quad \tilde{A}_{25}], \\ V_2 [V_2^* A_{12}^* &\quad V_2^* A_{42}^* \quad V_2^* A_{52}^*] = V_2 [\tilde{B}_{12}^* \quad \tilde{A}_{42}^* \quad \tilde{A}_{52}^*], \end{aligned}$$

where we have already identified the blocks  $\tilde{B}_{21}$  and  $\tilde{B}_{12}$ , which make

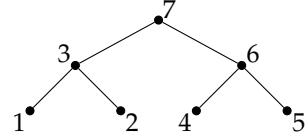


Figure 6.2: Illustration of numbering in the HSS hierarchy.

up the block  $B_3$ . Consequently, we can rewrite  $A$  as

$$A = \begin{bmatrix} U_1 & U_2 & \\ & I & \\ & & I \end{bmatrix} \begin{bmatrix} \mathbf{0} & \tilde{B}_{12} & \tilde{A}_{14} & \tilde{A}_{15} \\ \tilde{B}_{21} & \mathbf{0} & \tilde{A}_{24} & \tilde{A}_{25} \\ \tilde{A}_{41} & \tilde{A}_{42} & \mathbf{0} & A_{45} \\ \tilde{A}_{51} & \tilde{A}_{52} & A_{54} & \mathbf{0} \end{bmatrix} \begin{bmatrix} V_1^* & & \\ & V_2^* & \\ & & I \end{bmatrix} + D,$$

which already reveals some of the HSS structure. Moreover, we observe that we can safely ignore any previously constructed bases while proceeding through the compression.

We move on to node 3, which is the parent of nodes 1 and 2. We extract the modified HSS block row  $\tilde{A}_{3,:}$  and use it to compute the translators  $U_3$

$$\tilde{A}_{3,:} = \begin{bmatrix} \tilde{A}_{14} & \tilde{A}_{15} \\ \tilde{A}_{24} & \tilde{A}_{25} \end{bmatrix} = U_3 \begin{bmatrix} \tilde{A}_{34} & \tilde{A}_{35} \end{bmatrix}.$$

At this stage we can write  $A$  as

$$A = \begin{bmatrix} U_1 & U_2 & \\ & I & \\ & & I \end{bmatrix} \left( \begin{bmatrix} U_3 & & \\ & I & \\ & & I \end{bmatrix} \begin{bmatrix} \mathbf{0} & \tilde{A}_{34} & \tilde{A}_{34} \\ \tilde{A}_{43} & \mathbf{0} & A_{45} \\ \tilde{A}_{53} & A_{54} & \mathbf{0} \end{bmatrix} \begin{bmatrix} V_3^* & & \\ & I & \\ & & I \end{bmatrix} \right. \\ \left. + \begin{bmatrix} B_3 & & \\ & \mathbf{0} & \\ & & \mathbf{0} \end{bmatrix} \right) \begin{bmatrix} V_1^* & & \\ & V_2^* & \\ & & I \end{bmatrix} + D,$$

which partly mirrors the HSS structure shown in (5.36). To continue the compression for the nodes 4 and 5, we can use the newly compressed blocks  $\tilde{A}_{34}, \tilde{A}_{35}$ . We proceed with the compression, by processing nodes 4,5,6 and 7. At the root node 7, there are no HSS block columns or rows to compress and the method terminates with the extraction of the off-diagonal blocks  $B_7$ . The procedure is implemented using recursion in Algorithm 6.7, where we omit the details of how the block columns and rows are formed as it should be evident from the above discussion.

```

procedure HSS COMPRESS( $A$ )
    Initialize the HSS block structure
    for all nodes  $i$  in post-order do
        if  $i$  is a leaf node then
            Extract the diagonal block  $D_i$ 
            Extract  $U_i, V_i$  from HSS block row/column  $i$ 
            Apply  $U_i^*, V_i^*$  to the HSS block row/column3
        else
            Find children nodes  $c_1$  and  $c_2$ 
            Extract off-diagonal blocks  $\tilde{B}_{c_1,c_2}$  and  $\tilde{B}_{c_2,c_1}$ 
            if  $i$  is not the root node then
                Extract  $U_i, V_i$  from HSS block row/column  $i$ 
                Apply  $U_i^*, V_i^*$  to the HSS block row/column3
            end if
        end if
    end for
    return  $A$  in HSS format
end procedure

```

<sup>3</sup>: When we overwrite the HSS block rows and columns, their first/second dimension has changed, which has to be accounted for.

**Algorithm 6.7:** Direct compression algorithm for HSS matrices. A concrete implementation can be found in [46].

This approach to HSS compression is general and will attempt to compress any matrix  $A$  into HSS format. It will also reveal the HSS rank of  $A$ , provided that a rank-revealing QR factorization is used to compute the steps (6.19) and (6.20). However, its computational complexity is  $\mathcal{O}(n^2)$ , which makes it too expensive for most applications and undermines the linear complexity arithmetic of HSS matrices.

Much in the same way as HODLR and low-rank approximations require frequent recompression, this is also the case for HSS matrices. It is therefore useful to derive a recompression method from the direct compression Algorithm 6.7, which can be used to recompress matrices when ranks grow too large. The recompression algorithm is presented in [49] and implementations can be found in [45, 46].

## Randomized compression

In this section we discuss an alternative approach for compression, which is based on randomized compression and has been first presented in [50]. In most cases, we do not have direct access to the matrix  $A$  as assumed in the previous section. Instead, the matrix is given to us implicitly, in the sense that we can compute matrix-vector products  $x \rightarrow Ax, x \rightarrow A^*x$  and we can access individual entries, i.e.  $(i, j) \rightarrow A(i, j)$ . Much like the randomized methods for low-rank approximation, this method uses randomized sampling to construct a HSS representation of  $A$ . Under the assumption that the matrix-vector products can be computed in  $\mathcal{O}(n)$  operations and assuming that the access to individual entries is a  $\mathcal{O}(1)$  operation, we obtain an algorithm which can compress the matrix  $A$  in  $\mathcal{O}(k^2n)$  operations.

To simplify the discussion, we assume that we are given a symmetric matrix  $A$  of order  $n$ , which we know to be an HSS matrix of exact HSS rank  $k$ . Let us assume that we have sampled the columns of  $A$  using the  $m \times (k + p)$  random Gaussian  $\Omega$ , where  $p$  is again a small integer for oversampling:

$$S = A\Omega. \quad (6.22)$$

Moreover, let us assume that the diagonal blocks at the leaf level  $D^{(L)}$  are known to us. Then, we observe that we can easily remove the action of the diagonal block by computing

$$S^{(L)} = (A - D^{(L)})\Omega = S - D^{(L)}\Omega. \quad (6.23)$$

If  $I_1$  are the rows corresponding to the first leaf node in the cluster tree, we have

$$S_{\text{loc},1} = S^{(L)}(I_1, :) = A(I_1, I \setminus I_1)\Omega(I \setminus I_1, :). \quad (6.24)$$

In other words, we have sampled the first HSS block row of  $A$ , which is spanned by the column generators  $U_1^{(L)}$ . Thus, we can use the techniques introduced in Chapter 2 to extract the generators. In a similar way, for non-symmetric matrices, we can obtain the row generators  $V^{(L)}$  by sampling  $A^*$ .

The question that arises is how to proceed with the algorithm recursively. To obtain a sample matrix  $S^{(L-1)}$  of the HSS block rows on the next level,

we have to eliminate the diagonal element

$$\mathbf{D}^{(L-1)} = \mathbf{U}^{(L)} \mathbf{B}^{(L-1)} (\mathbf{V}^{(L)})^* + \mathbf{D}^{(L)}, \quad (6.25)$$

which now includes contributions from the off-diagonal blocks  $\mathbf{B}^{(L-1)}$  of the previous level. If we have a way to obtain  $\mathbf{B}^{(L-1)}$ , we can therefore compute

$$\begin{aligned} \mathbf{S}^{(L-1)} &= (\mathbf{A} - \mathbf{D}^{(L-1)}) \boldsymbol{\Omega} = (\mathbf{A} - \mathbf{D}^{(L)}) \boldsymbol{\Omega} - (\mathbf{D}^{(L)} - \mathbf{D}^{(L-1)}) \boldsymbol{\Omega} \\ &= \mathbf{S}^{(L)} - \mathbf{U}^{(L)} \mathbf{B}^{(L-1)} (\mathbf{V}^{(L)})^* \boldsymbol{\Omega}, \end{aligned} \quad (6.26)$$

to obtain the sample matrix for the next level. Thus, we need to find a way to determine the entries in the compressed blocks of  $\mathbf{B}^{(L-1)}$  after determining the generators  $\mathbf{U}^{(L)}$  and  $\mathbf{V}^{(L)}$ . This is where the randomized interpolative decomposition (2.31) comes into play. By applying the interpolative decomposition to the transpose of  $\mathbf{S}^{(L)}(\tilde{I}_1, :)$  in (6.24), we select up to  $k$  rows  $\tilde{I}_1 \subseteq I_1$ , such that

$$S_{\text{loc},1} = \mathbf{U}_1^{(L)} S_{\text{loc}}(\tilde{I}_1, :) = \mathbf{U}_1^{(L)} \mathbf{A}(\tilde{I}_1, I \setminus I_1) \boldsymbol{\Omega}(I \setminus I_1, :).$$

We repeat this for the node 2 to recover  $\mathbf{U}_2^{(L)}$  and  $\tilde{I}_2$ . By the nature of the interpolative decomposition, we find that

$$\begin{aligned} S_{\text{loc},1} &= \mathbf{U}_1^{(L)} \mathbf{A}(\tilde{I}_1, \tilde{I}_2) (\mathbf{U}_2^{(L)})^* \boldsymbol{\Omega}(I_2, :) \\ &\quad + \mathbf{U}_1^{(L)} \mathbf{A}(\tilde{I}_1, I \setminus I_3) (\mathbf{U}_2^{(L)})^* \boldsymbol{\Omega}(I \setminus I_3, :), \end{aligned} \quad (6.27)$$

and, we have successfully isolated the action of the block associated to  $\mathbf{A}(I_1, I_2)$ . More importantly, we can extract  $\tilde{\mathbf{B}}_{1,2}^{(L)}$  by setting

$$\tilde{\mathbf{B}}_{1,2}^{(L)} = \mathbf{A}(\tilde{I}_1, \tilde{I}_2), \quad (6.28)$$

which requires access to only  $\mathcal{O}(k^2)$  entries to determine the action of the block  $\mathbf{A}(I_1, I_2)$ . Moreover, due to the nestedness of the bases, we can continue the factorization by only considering the selected rows and columns  $\tilde{I}_1 \cup \tilde{I}_2$ . The method is summarized in Algorithm 6.8. We refer the reader to [50] for the non-symmetric case and a complete discussion of the algorithm.

The merit of this algorithm is its low computational cost. It requires  $\mathcal{O}(k)$  matrix-vector products and access to  $\mathcal{O}(kn)$  entries of  $\mathbf{A}$ . The remaining operations then require  $\mathcal{O}(k^2n)$  floating point operations. This brings the overall cost to  $\mathcal{O}(k^2n)$ , assuming that the matrix-vector multiplications can be computed in  $\mathcal{O}(n)$  operations and that the access to individual entries is  $\mathcal{O}(1)$ .

In most cases, we are presented with a matrix that we assume to be compressible, but we do not know its HSS rank  $k$  beforehand. In such cases, it is useful to utilize an adaptive version of the algorithm, which applies an error estimator to control the quality of the approximation. A very simple approach using the Frobenius norm estimator (2.29) is presented in Algorithm 6.9. Here, we simply restart the compression if the estimator exceeds the specified tolerance  $\epsilon$ . The sampling matrix used to estimate the norm can be reused for the next iteration. This algorithm

```

procedure HSS RANDOM COMPRESS( $A, k$ )
  Generate initial  $n \times (k + p)$  random Gaussian matrix  $\Omega$ 
  Evaluate  $S \leftarrow A\Omega$  via matrix-vector multiplication
  for all levels  $l$ , starting from  $L$  to 1 do
    for all nodes  $i$  on level  $l$  do
      if  $i$  is a leaf node then
         $I_{\text{loc}} \leftarrow I_i$ 
         $\Omega_{\text{loc}} \leftarrow \Omega(I_i, :)$ 
         $S_{\text{loc}} \leftarrow S(I_i, :) - A(I_i, I_i)\Omega_{\text{loc}}$ 
         $D_i^{(l)} \leftarrow A(I_i, I_i)$ 
      else
        Let  $c_1$  and  $c_2$  be the two children of node  $i$ 
         $I_{\text{loc}} \leftarrow [\tilde{I}_{c_1}, \tilde{I}_{c_2}]$ 
         $\Omega_{\text{loc}} \leftarrow \begin{bmatrix} \Omega_{c_1} \\ \Omega_{c_2} \end{bmatrix}$ 
         $S_{\text{loc}} \leftarrow \begin{bmatrix} S_{c_1} - A(\tilde{I}_{c_1}, \tilde{I}_{c_2})\Omega_{c_2} \\ S_{c_2} - A(\tilde{I}_{c_2}, \tilde{I}_{c_1})\Omega_{c_1} \end{bmatrix}$ 
         $B_i^{(l)} \leftarrow \begin{bmatrix} 0 & A(\tilde{I}_{c_1}, \tilde{I}_{c_2}) \\ A(\tilde{I}_{c_2}, \tilde{I}_{c_1}) & 0 \end{bmatrix}$ 
      end if
      Form interpolative Decomposition  $S^* \approx S_{\text{loc}}^*(:, J_i)(U_i^{(l)})^*$ 
       $\Omega_i \leftarrow (U_i^{(l)})^* \Omega_{\text{loc}}$ 
       $S_i \leftarrow S_{\text{loc}}(J_i, :)$ 
       $\tilde{I}_i \leftarrow I_{\text{loc}}(J_i)$ 
    end for
  end for
end procedure

```

**Algorithm 6.8:** Randomized HSS compression of a symmetric HSS matrix  $A$  of exact HSS rank  $k$ .

```

Generate initial  $n \times (k + p)$  random Gaussian matrix  $\Omega$ 
Evaluate  $S \leftarrow A\Omega$  via matrix-vector multiplication
Form  $A_{\text{HSS}}$  using  $S, \Omega$  and access to  $A$ 
loop
  Generate  $n \times r$  random Gaussian matrix  $\tilde{\Omega}$ 
  Evaluate  $\tilde{S} \leftarrow A\tilde{\Omega}$  via matrix-vector multiplication
  if  $\|\tilde{S} - A_{\text{HSS}}\tilde{\Omega}\|_{\text{F}}^2/r \leq \epsilon^2$  then
    return  $A_{\text{HSS}}$  in HSS format, defined by  $U_i^{(l)}, D_i^{(l)}$  and  $B_i^{(l)}$ 
  else
    Increase the rank:  $k \leftarrow k + r$ 
    Update  $\Omega \leftarrow [\Omega \quad \tilde{\Omega}]$  and  $S \leftarrow [S \quad \tilde{S}]$ 
  end if
end loop

```

**Algorithm 6.9:** Adaptive version of the randomized HSS compression algorithm 6.8.

results in a worst-case complexity of  $\mathcal{O}(k^3n)$  and depends largely on how good the initial guess for  $k$  is. More sophisticated approaches which use local error estimators to estimate the overall error are presented in [42, 51, 52]. Moreover, restarting approaches can be utilized to not only re-use the sample matrix from previous steps but also the rows and columns selected by the interpolative decomposition, as done in [45]. Finally, we note that a fully matrix-free algorithm for the construction of HSS matrices, purely based on randomized sampling and matrix-vector products has been proposed [53]. However, this algorithm requires  $\mathcal{O}(\log n)$  extra matrix-vector products, increasing the overall cost to  $\mathcal{O}(n \log n)$  assuming linear complexity matrix-vector products.

## 6.4 HssMatrices.jl

The algorithms that we have presented for HSS matrices have been implemented in the programming language JULIA and have been made available as a Julia package under the name `HssMATRICES.JL` [46]. Its main purpose is to provide a package for research and development purposes in the programming language JULIA. We give a brief introduction of its core functionality here.

After installation, we can load it by running `using HssMatrices`. Let us construct a simple example matrix `hssG`, using the direct compression algorithm:

```
# A simple example using HssMatrices.jl
using LinearAlgebra
using HssMatrices

g(x,y) = abs(x-y) > 0. ? 1/abs(x-y) : 1.
G = [g(x,y) for x=-1:0.001:1, y=-1:0.001:1]
hssG = hss(G)
```

In this example, we first generate a discrete representation of the kernel function  $g(x, y) = 1/(x - y)$  in `G`. The smart constructor `hss(G)` detects that `G` is dense and uses the appropriate algorithm for constructing the HSS representation. We can visualize the resulting HSS structure by running

```
plotranks(hssG)
```

which yields a result similar to Figure 6.3. We observe that the smart constructor automatically generates cluster trees for the row- and column-indices through bisection. If we want more control over the cluster trees, we can generate them manually and pass them to the constructor. The command `bisection_cluster(2001, leafsize=100)` will construct a cluster tree of length 2001 and maximum leafsize 100 using bisection. We can pass these cluster trees to the constructor.

```
rcl = bisection_cluster(2001, leafsize=100)
ccl = bisection_cluster(2001, leafsize=100)
hssG = hss(G, rcl, ccl)
```

We can also extract the cluster trees of an existing matrix.

```
clusters(hssG)
```

We might want to manually specify which algorithm we would like to use for compression.

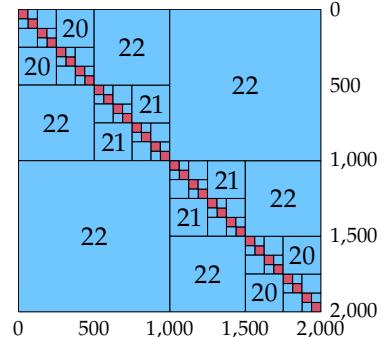
```
hssG = compress(G, rcl, ccl)
hssG = randcompress(G, rcl, ccl, 22)
hssG = randcompress_adaptive(G, rcl, ccl)
```

The first line is the direct compression Algorithm 6.7, the second one is the randomized approach (Algorithm 6.8), where a guess for the rank is specified, and finally the third line is the adaptive version of the randomized algorithm (Algorithm 6.9).

Due to the functional nature of JULIA, we can also specify functions to be passed on to the constructor. To use the randomized compression, we can construct a `LinearMap` object which holds the functions for matrix-vector multiplication and indexed access. Passing it to the constructor will cause it to use the efficient algorithm with random sampling, which will then

**Listing 6.1:** `HssMATRICES.JL` can be installed using the package manager `Pkg`.

```
using Pkg
Pkg.add("HssMatrices.jl")
```



**Figure 6.3:** Result of using the command `plotranks(hssG)`.

call the routines when needed. The following code creates a low-rank matrix which is then compressed using the randomized algorithm:

```
U = randn(2001, 3);
V = randn(2001, 3);
L = LinearMap{Float64}(2001, 2001, (y, _, x) -> U*V'*x, (y, _, x) -> V
    *U'*x, (i, j) -> U[i, :]*V[j, :]');
hssL = randcompress_adaptive(L, rcl, ccl)
```

Alternatively, we could call a specialized constructor to convert the low-rank matrix into HSS format:

```
hssL = lowrank2hss(U, V, rcl, ccl)
```

To check the HSS rank we can run `hssrank(hssL)` which, unsurprisingly, returns 3. Another important feature for compression is error control. We can directly control these parameters and others by passing them to the constructor.

```
hssG = hss(G, leafsize=64, atol=1e-6, rtol=1e-6)
```

Normally, `HssMATRICES.JL` uses default values for these parameters. In some settings, it is useful to set these parameters once. This can be done in the following way:

```
# Example for changing some of the standard parameters
HssMatrices.setopts!(leafsize=64)
HssMatrices.setopts!(atol=1e-9)
HssMatrices.setopts!(rtol=1e-9)
HssMatrices.setopts!(nooversampling=10)
```

Knowing how to generate elementary HSS matrices and how to visualize them, we are ready to try out some arithmetic. `HssMATRICES.JL` provides all of the common arithmetic operations that one would expect. We can define a vector and run the following commands to call the HSS matrix-vector multiplication and the solve step via ULV factorization.

```
x = randn(2001, 1)
hssA*x
hssA\`x
```

Moreover, `HssMATRICES.JL` implements a number of algorithms for arithmetic operations using multiple HSS matrices. We can for instance run:

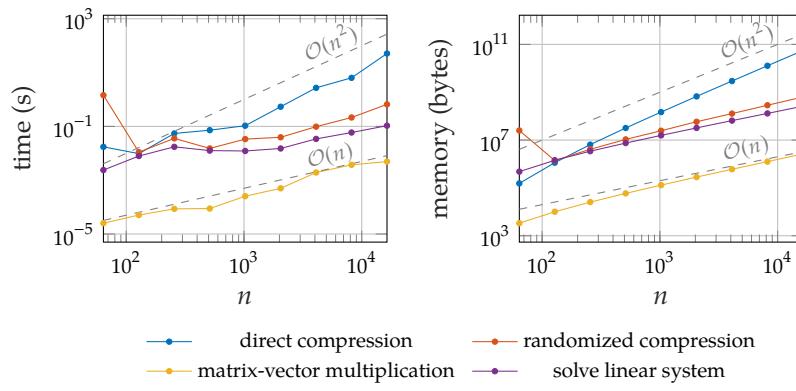
```
hssG+hssL
hssG-hssL
hssG*hssL
hssG\hssL
hssL/hssG
```

One important thing to note is that `HssMATRICES.JL` does not perform recompression automatically. Instead, it gives the user control of when and how to perform recompression. In this way, we avoid over-compressing, potentially with the wrong tolerances as we assume that the user is aware of when best to recompress. To perform recompression we can run:

```
recompress!(hssG, atol=1e-3, rtol=1e-3)
```

Figure 6.4 shows the performance of selected algorithms implemented in `HssMATRICES.JL`. In particular, it shows execution times and memory usage for both compression algorithms, as well as matrix-vector multiplication and the efficient solver based on the ULV factorization.<sup>4</sup> The experiments were performed with the matrix  $K(i, j) = \log |x_i - x_j|$ , where  $x_i \in [0, 1]$  is a set of equidistant points on the unit interval. It is clear that all

<sup>4</sup>: The performance figures were obtained on a 2019 MacBook Pro with an Intel i9 Processor clocked at 2.3 GHz and 32GB of RAM.



**Figure 6.4:** Timings and memory requirements of important HSS algorithms using HssMATRICES.JL.

algorithms apart from the direct compression Algorithm 6.7 achieve linear complexity as we would expect.

For more information on the library, we invite the reader to take a look online [github.com/bonevbs/HssMatrices.jl](https://github.com/bonevbs/HssMatrices.jl) [46].

# Hierarchical approximate solvers

In Chapter 3, we saw that the main contributor to the computational cost of sparse direct solvers is the fill-in that is created during the factorization. The main idea to pursue is to reduce this cost by using rank-based approximations. More specifically, the Schur complements arising in the factorization of matrices stemming from the discretization of elliptic PDEs are known to be compressible using rank-structured matrix formats [54–59]. The original idea can be traced back to [32, 55, 57, 60] and since then, variations of this idea have been proposed in the literature [51, 61–69]. We loosely refer to these methods as *hierarchical approximate solvers* as they exploit the hierarchical nature of both the nested dissection and the rank structure of the fill-in.

In Section 7.1 we offer a rough intuition of why these methods work before we give an overview of some existing methods in Section 7.2. A new algorithm is then introduced in Section 7.3, which is the main contribution of this work. The computational cost of the algorithm is analyzed in Section 7.4.

## 7.1 Compressing the fill-in

In Chapter 5 we gained some intuition regarding discrete representations of Green's functions and their compressibility using hierarchical matrices. This is essentially linked to the separability of variables corresponding to row and column indices in disjoint regions. It is therefore inherently intuitive that the inverse of the FE Galerkin matrix  $A^{-1}$ , which corresponds to the discrete representation of the Green's function should be compressible using hierarchically rank-structured matrices. Bebendorf and Hackbusch prove that the inverses of Galerkin matrices can indeed be approximated with  $\mathcal{H}$ -matrices [54]. [54] also proves useful theoretical bounds on the off-diagonal ranks of the inverse, based on the quality of the FE discretization.<sup>1</sup> Similar results are also reported for Schur complements that arise in the structured factorization of  $A$  [55, 58]. We pursue a purely algebraic approach to motivate the compressibility of the inverse  $A^{-1}$  and associated Schur complements.

### Compressibility of the inverse

The following Lemma introduces a useful relationship between the ranks of off-diagonal blocks in  $A$  and its inverse [70].

**Lemma 7.1.1** (Ranks of the block-inverse) *Let*

$$A = \begin{bmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{bmatrix}, \quad B = A^{-1} = \begin{bmatrix} B_{11} & B_{12} \\ B_{21} & B_{22} \end{bmatrix}$$

7.1 Compressing the fill-in . . . . .	66
The inverse . . . . .	66
Schur complements . . . . .	68
7.2 Existing methods . . . . .	69
7.3 Approximate factorization . . . . .	70
Well-separated nodes . . . . .	71
Block elimination . . . . .	73
Computing $\hat{L}^{(\sigma)}$ and $\hat{R}^{(\sigma)}$ . . . . .	76
Compressing $\hat{S}^{(\sigma)}$ . . . . .	77
Forming the factorization . . . . .	78
7.4 Complexity of the algorithm	79

1: The result is reported in Section 8.2, where it is compared to our numerical experiments.

be invertible block matrices such that  $A_{11}, B_{11} \in \mathbb{R}^{m_1 \times m_1}$  and  $A_{22}, B_{22} \in \mathbb{R}^{m_2 \times m_2}$ . Moreover, let  $A_{11}$  and  $A_{22}$  be invertible. Then we have

$$\text{rank } B_{12} = \text{rank } A_{12}, \quad (7.1)$$

$$\text{rank } B_{21} = \text{rank } A_{21}, \quad (7.2)$$

*Proof.* As  $A_{11}$  and  $A_{22}$  are invertible, the diagonal blocks of  $B$  are given by the inverse of the respective Schur complements

$$B_{11}^{-1} = A_{11} - A_{12}A_{22}^{-1}A_{21}, \quad (7.3)$$

$$B_{22}^{-1} = A_{22} - A_{21}A_{11}^{-1}A_{12}. \quad (7.4)$$

The Schur determinant formula [71, pp. 5] states

$$\det(A) = \det(A_{11})\det(B_{22}^{-1}) = \det(A_{22})\det(B_{11}^{-1}), \quad (7.5)$$

which implies that the diagonal blocks in  $B$  have full rank. Computing  $AB = I$  yields

$$A_{11}B_{12} + A_{12}B_{22} = 0, \quad (7.6)$$

$$A_{21}B_{11} + A_{22}B_{21} = 0, \quad (7.7)$$

which implies  $\text{rank } A_{12} = \text{rank } B_{12}$  and  $\text{rank } A_{21} = \text{rank } B_{21}$ .  $\square$

Unfortunately, Lemma 7.1.1 is not very useful in finding bounds for a HODLR matrix  $A$  and the off-diagonal ranks of its inverse. The situation is slightly different with HSS matrices, where the definition of the HSS rank allows us to prove the following Theorem.

**Theorem 7.1.2** (The inverse of HSS matrices) *Let  $A$  be an invertible matrices and  $B = A^{-1}$  its inverse. Assume that  $A$  is a HSS matrix of HSS rank  $k$  for a given block cluster tree  $\mathcal{T}_{I \times I}$  with invertible diagonal blocks. Then, the inverse  $B$  is also a HSS matrix with clustering according to  $\mathcal{T}_{I \times I}$  and HSS rank  $k$ .<sup>2</sup>*

*Proof.* For any diagonal block  $I_i^l \times I_i^l \in \mathcal{T}_{I \times I}$ , we can find a permutation matrix  $\Pi$  which exposes the current diagonal block:

$$\Pi A \Pi^{-1} = \begin{bmatrix} A(I_i^l, I_i^l) & A(I_i^l, I \setminus I_i^l) \\ A(I \setminus I_i^l, I_i^l) & A(I \setminus I_i^l, I \setminus I_i^l) \end{bmatrix}. \quad (7.8)$$

Applying Lemma 7.1.1 then yields

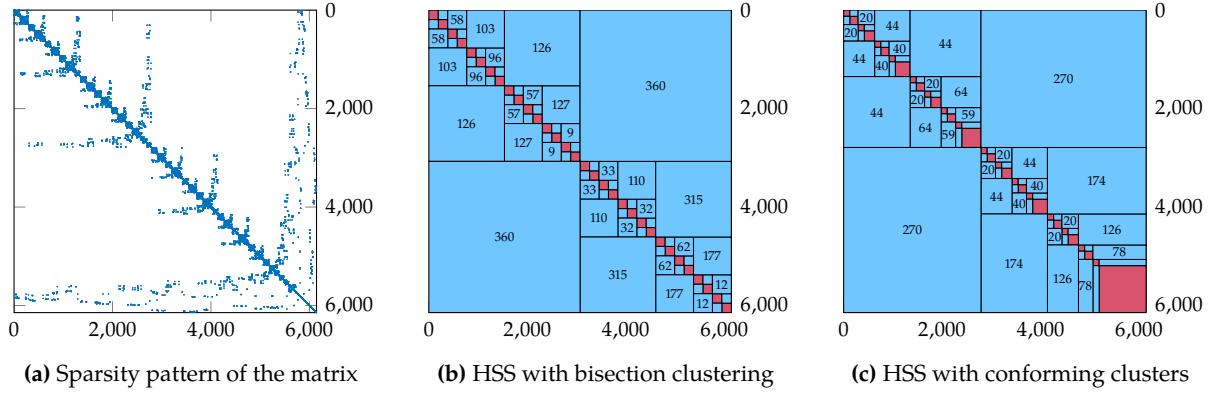
$$\text{rank } B(I_i^l, I \setminus I_i^l) = \text{rank } A(I_i^l, I \setminus I_i^l) \leq k,$$

$$\text{rank } B(I \setminus I_i^l, I_i^l) = \text{rank } A(I \setminus I_i^l, I_i^l) \leq k,$$

which completes the proof.  $\square$

2: Quasiseparable matrices possess a similar property, which can be regarded as special case of this theorem. Quasiseparable matrices can be understood as HSS matrices with a leaf size of 1 and HSS rank 1. Consequently, the inverse of a quasiseparable matrix is equally quasiseparable.

To apply this theorem to FE Galerkin matrices, we need to convince ourselves that these matrices can indeed be compressed into HSS format. Figure 7.1 shows the Galerkin matrix of a Poisson problem in two dimensions, compressed into HSS format with two different block cluster trees. The clustering in Figure 7.1c respects the nested-dissection structure



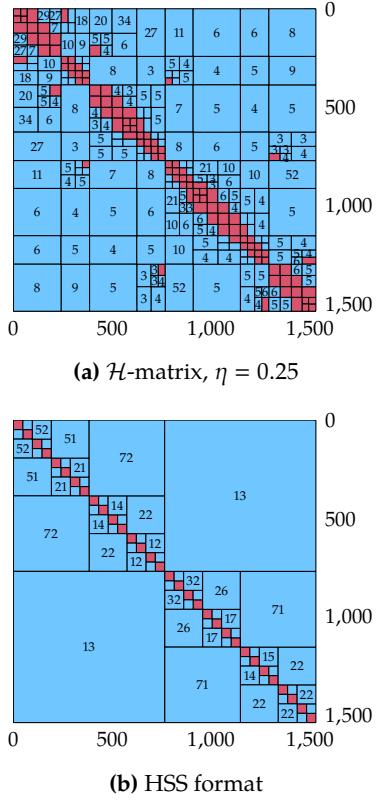
**Figure 7.1:** Stiffness matrix of the two-dimensional Poisson problem. The first figure shows the sparsity pattern due to the nested dissection reordering. The two figures on the right show it in HSS format using a simple bisection clustering, as well as a conforming clustering on the right.

of the matrix, whereas the clustering in Figure 7.1b is a straight-forward bisection cluster. The former yields lower off-diagonal ranks but has a suboptimal structure compared to the latter. We observe small off-diagonal ranks, which implies good compressibility. This also implies that a valid strategy for solving the linear system (1.22) is to compress  $A$  into HSS format, and then solve it using the ULV algorithm 6.6. Such approaches have been used in situations, in which  $A$  can be constructed directly or efficiently in hierarchical format, such as boundary element methods (BEM) [40, 72–74]. For FE methods, this has a few drawbacks however. Chief among them is the missed opportunity of exploiting the sparsity of  $A$ . Representing  $A$  as HSS matrix is in many cases less efficient than the original, sparse representation. This could be remedied by using sparse HSS representations. This is not sufficient however, as large zero blocks are not exploited by the ULV algorithm in the same way as they are with the structured elimination using nested dissection, as presented in Section 3.4.

## Compressibility of Schur complements

We pursue another idea. Theorem 7.1.2 not only implies that the inverse is compressible, but also the Schur complements that appear in the factorization. This can be seen by considering the diagonal blocks of the inverse, which are the inverses of the associated Schur complements. As these diagonal blocks are themselves HSS matrices, we know that their inverses must be HSS as well. We can therefore expect the Schur complements in the structured factorization of  $A$  to be compressible as well.<sup>3</sup> This property can be further explained by the properties of the so-called *Dirichlet-to-Neumann operators*, which is the integral operator linked to the Schur complements. A detailed treatment on the matter can be found in [25].

We investigate the rank structure of the top-level Schur complements as they appear in the structured elimination. Figure 7.2 shows the Schur complements for Poisson problems in two dimensions. In the top figure we use an adaptive clustering to represent it as  $\mathcal{H}$ -matrix. We generate the cluster by refining inadmissible blocks into four submatrices. This procedure is repeated recursively until a minimum block-size is reached.



**Figure 7.2:** Rank structure of the top-level Schur complement of a Poisson matrix in two dimensions. In both cases the matrices are compressed as  $\mathcal{H}$ -matrices with binary cluster trees using the  $\eta$ -admissibility condition and a compression tolerance of  $\epsilon = 10^{-6}$ .

3: This is equally true for the update matrices (3.10), which have comparable off-diagonal and HSS ranks as the associated Schur complements.

Blocks that remain inadmissible are then represented as dense matrices. We can clearly see that most blocks in the upper and lower triangular half of the matrix are admissible, which leads to a good compressibility also in HSS format. This is illustrated in the bottom figure, which shows the same matrix, but in HSS format.

The situation becomes somewhat worse in three dimensions. Figure 7.3 depicts the Schur complements for three-dimensional Poisson problems and their rank structure in  $\mathcal{H}$ -format. We observe that many more blocks on the off-diagonal are not admissible. While we can relax the admissibility condition to obtain a block diagonal structure, we can see that many of the resulting blocks have ranks that are too large to be efficient. Hence, we can expect considerably larger ranks, when compressing to HSS format as evidenced in Figure 7.4.

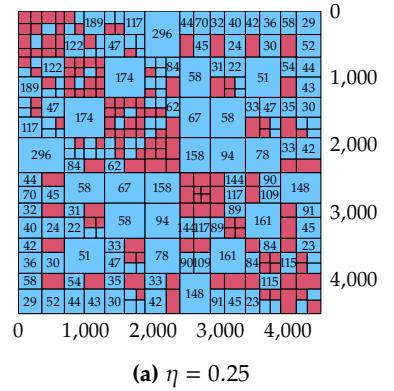
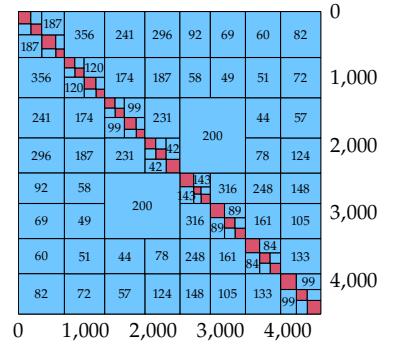
While this does not mean that rank-structured techniques are fundamentally ineffective in three dimensions, it does require much larger matrices for them to become effective.

## 7.2 Existing methods

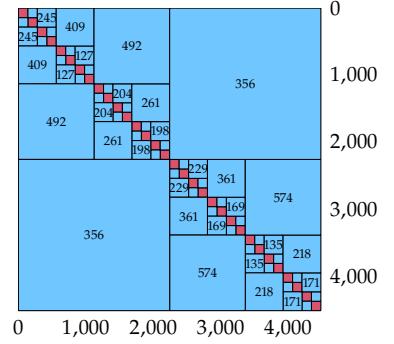
As mentioned earlier, the central idea that is pursued here is that the fill-in is compressible with rank-structured formats. This is an active field of study and a variety of algorithms have been proposed that exploit this idea. The key question with these methods is how to efficiently accumulate the action of the Schur complements which are now represented in a hierarchically rank-structured format. More precisely, how to perform the extend-add operation (3.12) and how to compute the fill-in in compressed format are the two key questions here. Before we proceed with our algorithm, we give an overview of existing methods, which aims to point out the differences to our approach.

**Arithmetic, based on multifrontal elimination** A straight-forward approach is obtained by replacing dense operations in the multifrontal elimination with HSS arithmetic. Some entries in this category are [60, 61, 75]. This requires the development of strategies to form the frontal matrices and perform the extend-add operation (3.12) using HSS matrices. In [60], the authors develop algorithms to permute HSS matrices and extend them with zero blocks. Particular care has to be taken for situations in which children nodes have overlaps in their boundaries (3.5). Finally, the HSS blocks need to be predetermined in a symbolic factorization stage. Consequently, the algorithm is difficult to implement and not applicable to general connectivities. In [61], the method is then extended to more general, unstructured grids. In [75], the authors further integrate the HSS ULV factorization algorithm 6.6 with the multifrontal method. This allows them to use reduced representations for intermediate matrices, which allows them to replace the specialized algorithms for the HSS extend-add operation with simpler ones. The authors report a complexity of  $\mathcal{O}(kn \log n)$  in two dimensions and  $\mathcal{O}(kn^{4/3})$  in three dimensions to compute the factorization.

**Arithmetic, based on sequential elimination** It is also possible to use the sequential Gaussian elimination as presented in Section 3.2. In

(a)  $\eta = 0.25$ (b)  $\eta = 0.15$ 

**Figure 7.3:** Rank structure of the top-level Schur complement of a Poisson matrix in three dimensions. In both cases the matrices are compressed as  $\mathcal{H}$ -matrices with binary cluster trees using the  $\eta$ -admissibility condition and a compression tolerance of  $\epsilon = 10^{-6}$ .



**Figure 7.4:** HSS Rank structure of the top-level Schur complement for the Poisson matrix in three dimensions. As in previous examples the matrix is compressed using a compression tolerance of  $\epsilon = 10^{-6}$ .

[57], a fast direct solver based on sequential Gaussian elimination and HODLR matrices is proposed. This approach works well on meshes which can be arranged into concentric annuli, such that the associated sparsity pattern consists of a tridiagonal and a conical part [57]. The resulting algorithm requires  $\mathcal{O}(n \log^2 n)$  operations to solve the system.

**Based on random sampling and multifrontal elimination** [66, 67, 75] and equally use structured elimination, however use some form of randomized sampling and in particular, Algorithm 6.8 to compress the fill-in. In the case of [75] and [66], the structured extend-add process is replaced by a “skinny” extend-add operation, which operates only on the matrix products of the fill-in and a random matrix. In this way, the fill-in can be reconstructed using randomized compression techniques. [67] similarly makes use of randomized compression to avoid the extend-add operation, however we believe there is an oversight concerning the applicability of the compression algorithm. More precisely, the necessity of accessing individual entries for compression is not discussed and it is not evident to us how this could be achieved.

**Approximating  $A$  first** There are also a large number of methods that attempt to first approximate the matrix  $A$  with some hierarchically structured matrix format, before it is factorized [40, 72–74, 76]. As these are mostly unrelated to our approach, we do not discuss them in detail and refer the reader to the original literature.

Our approach can be understood as a hybrid one, as we use both randomized compression and structured matrix arithmetic to accelerate the structured Gaussian elimination.

### 7.3 Approximate factorization

We introduce our algorithm for forming an approximate factorization

$$A \approx P = LDR, \quad (7.9)$$

which can either be used as a fast approximate solver or, alternatively, as a preconditioner for an iterative method such as GMRES. We are mainly concerned with the latter, but as we can control the error and guarantee

$$\|A - P\| \leq \epsilon \|A\|$$

for a specified  $\epsilon \in \mathbb{R}_{>0}$  with high probability, we point out that we can use it as an approximate solver as well. To see how this affects the error of the solution  $x$ , we can turn to a classical result from perturbation analysis [12]. Solving the perturbed linear system

$$(A + \delta F)\tilde{x} = b$$

with  $A, F \in \mathbb{R}^{n \times n}$  and  $\delta \in \mathbb{R}_{>0}$  reveals that for any vector norm and consistent matrix norm, the solution satisfies

$$\frac{\|\tilde{x} - x\|}{\|x\|} \leq \delta \|A^{-1}\| \|F\| + \mathcal{O}(\delta^2).$$

In particular, this means that the solution  $\tilde{x} = P^{-1}b$ , computed with the approximate factorization, satisfies

$$\frac{\|\tilde{x} - x\|}{\|x\|} \leq \epsilon \|A^{-1}\| \|A\| + \mathcal{O}(\epsilon^2) = \epsilon \kappa(A) + \mathcal{O}(\epsilon^2). \quad (7.10)$$

Thus, we can control the error in the approximate solution  $\tilde{x}$  by controlling the quality of the approximate factorization  $P$ . The caveat is that according to this error bound, which is conservative, the tolerance  $\epsilon$  has to be adapted to the condition number  $\kappa(A)$ , which may lead to a prohibitively small  $\epsilon$ .

## Well-separated nodes

We return to the task at hand, which is to improve the cost of computing  $P = LDR$  by employing rank-structured approximations. We return to the discussion in Section 3.4 and the elimination degrees of freedom in  $A$  based on the nested dissection  $\mathcal{E}$ .

Again, we are located at node  $\sigma$  in the elimination tree as depicted in Figure 3.9. We remind ourselves of the notation  $I^\sigma$  and  $B^\sigma$ ;  $I^\sigma$  are the degrees of freedom that have been designated for elimination, loosely referred to as the interior of  $\sigma$ .  $B^\sigma$  is the boundary of  $\sigma$  and contains all degrees of freedom which receive contributions from the elimination of  $\sigma$ . We recall that we only have to consider  $\hat{A}^{(\sigma)}$ , which is the submatrix of  $\tilde{A}^{(\sigma)}$  containing all relevant entries ( $I^\sigma$  and  $B^\sigma$ ) and proceed to eliminate the interior degrees of freedom  $I^\sigma$  by factoring

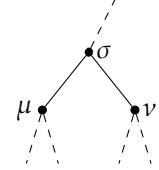
$$\hat{A}^{(\sigma)} = \begin{bmatrix} \tilde{A}_{ii}^{(\sigma)} & \tilde{A}_{ib}^{(\sigma)} \\ \tilde{A}_{bi}^{(\sigma)} & \tilde{A}_{bb}^{(\sigma)} \end{bmatrix} = \begin{bmatrix} I & \mathbf{0} \\ \hat{L}^{(\sigma)} & I \end{bmatrix} \begin{bmatrix} \hat{D}^{(\sigma)} & \mathbf{0} \\ \mathbf{0} & \hat{S}^{(\sigma)} \end{bmatrix} \begin{bmatrix} I & \hat{R}^{(\sigma)} \\ \mathbf{0} & I \end{bmatrix} \quad (3.6)$$

$$\hat{D}^{(\sigma)} = \tilde{A}_{ii}^{(\sigma)}, \quad (3.7)$$

$$\hat{L}^{(\sigma)} = \tilde{A}_{bi}^{(\sigma)} (\tilde{A}_{ii}^{(\sigma)})^{-1}, \quad (3.8a)$$

$$\hat{R}^{(\sigma)} = (\tilde{A}_{ii}^{(\sigma)})^{-1} \tilde{A}_{ib}^{(\sigma)}, \quad (3.8b)$$

$$\hat{S}^{(\sigma)} = \tilde{A}_{bb}^{(\sigma)} - \tilde{A}_{bi}^{(\sigma)} (\tilde{A}_{ii}^{(\sigma)})^{-1} \tilde{A}_{ib}^{(\sigma)}, \quad (3.9)$$



**Figure 3.9:** Elimination of node  $\sigma$ . Contributions from the children  $\mu$  and  $\nu$  have to be accounted for. (repeated from page 26)

where  $\tilde{A}$  represents the large intermediate matrix in which all nodes leading up to  $\sigma$  have been eliminated.

Let us imagine that we are eliminating  $\nu$  at the leaf level. Forming  $\hat{A}^{(\nu)}$  is as straight-forward as extracting all entries corresponding to the interior  $I^\nu$  and the boundary  $B^\nu$  degrees of freedom from  $\tilde{A}^{(\nu)}$ . Because  $\nu$  is a leaf node,  $\tilde{A}_{ii}^{(\nu)}$  only contains entries from the original matrix. In general, this is not the case for  $\tilde{A}_{bi}^{(\nu)}$ ,  $\tilde{A}_{ib}^{(\nu)}$  and  $\tilde{A}_{bb}^{(\nu)}$ , as the boundary degrees of freedom  $B^\nu$  might be shared with its sibling node  $\mu$ , i.e.  $B^\mu \cap B^\nu \neq \emptyset$ .

In the general scenario, we can proceed in one of two ways. The straight-forward fashion is to eliminate node  $\mu$ , then node  $\nu$  and then  $\sigma$ . When we eliminate  $\nu$ , the matrix  $\hat{A}^{(\nu)}$  already contains contributions from the elimination of  $\mu$  in the parts associated with the boundary. These contributions come in the form of entries extracted from the Schur complement  $\hat{S}^{(\mu)}$ . The alternative is to use the extend-add formalism (3.11) introduced in Section 3.4. In this case, we store the update matrices  $\hat{U}^{(\mu)}$

and  $\hat{\mathbf{U}}^{(\nu)}$  to keep track of the contributions from previous eliminations. These two procedures are mathematically equivalent. However, this choice has important consequences for algorithmic aspects of the method. The big advantage of the extend-add formalism is its parallel nature. We can simultaneously factor the sibling nodes  $\mu$  and  $\nu$  without having to account for the fill-in of the other sibling. Merging their contributions happens at the parent node  $\sigma$ , when we perform the extend-add operation. In the former, this happens when we have to extract a submatrix from the  $\hat{\mathbf{S}}^{(\mu)}$ , which contributes to  $\hat{\mathbf{A}}^{(\sigma)}$ .

Let us now examine the core idea of compressing the dense fill-in using rank-structured matrices. This either means that we compress and store the Schur complements  $\hat{\mathbf{S}}^{(\mu)}, \hat{\mathbf{S}}^{(\nu)}, \hat{\mathbf{S}}^{(\sigma)}$  or their counterparts, the update matrices  $\hat{\mathbf{U}}^{(\mu)}, \hat{\mathbf{U}}^{(\nu)}, \hat{\mathbf{U}}^{(\sigma)}$ . It is apparent that there are multiple challenges to overcome. First of all, we would like to form the factorization (3.6) efficiently. This we seek to overcome by employing efficient algorithms for hierarchical matrices. Then there is the challenge of compression, which we address later on. Perhaps the biggest challenge is how to assemble the matrix  $\hat{\mathbf{A}}^{(\sigma)}$  and integrate the contributions from the children nodes  $\mu$  and  $\nu$ , which are represented in some hierarchical format.

We propose to make a slight constraint to the nested dissection, which will greatly help our efforts in forming an approximate factorization efficiently. We propose to consider elimination trees  $\mathcal{E}$  which have the well-separated property:

**Definition 7.3.1** (well-separated nodes) *Let  $\mathcal{E}$  be a (binary) elimination tree and  $\mu, \nu, \sigma \in \mathcal{E}$ , such that  $\mu$  and  $\nu$  are the children of  $\sigma$  as usual. Then, we call  $\mu$  and  $\nu$  well-separated nodes, iff their boundaries  $B^\mu$  and  $B^\nu$  are disjoint, i.e.*

$$B^\mu \cap B^\nu = \emptyset. \quad (7.11)$$

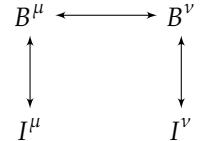
*Then, we call  $\mathcal{E}$  a well-separated elimination tree iff all sibling nodes in  $\mathcal{E}$  are well-separated.*

The concept of well-separated nodes is illustrated in Figure 7.5a. This property implies that sibling nodes  $\mu$  and  $\nu$  only interact with each other through their disjoint boundaries  $B^\mu \cap B^\nu = \emptyset$ . In other words, the elimination of  $\mu$  modifies only entries that are left unmodified by the elimination of  $\nu$  and vice-versa. To generate such an elimination tree, we modify the nested dissection algorithm to separate the degrees of freedom using disjoint boxes as illustrated in Figure 7.5b. By identifying boundary and interior degrees of freedom, we make sure that the elimination tree satisfies the well-separated property.

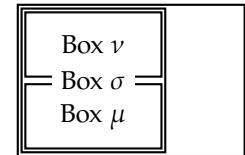
We return to the elimination of node  $\sigma$ . Because  $\mu$  and  $\nu$  are well-separated,  $B^\mu$  and  $B^\nu$  are disjoint, and we can form the four disjoint index sets

$$I^\sigma \cap B^\mu, I^\sigma \cap B^\nu, B^\sigma \cap B^\mu, B^\sigma \cap B^\nu,$$

which forms a partitioning of the indices associated with  $\sigma$ . With this partitioning, we take a renewed look at the matrix  $\hat{\mathbf{A}}^{(\sigma)}$ , which we can



(a) interaction of well-separated nodes



(b) nested dissection with boxes

**Figure 7.5:** Illustration of well-separated nodes. The top figure shows how the sibling nodes interact with each other only through their boundaries. The bottom figure shows how the nested dissection can be organized using boxes to generate an elimination tree with this property.

write as

$$\hat{A}^{(\sigma)} = \left[ \begin{array}{cc|cc} \hat{S}_{ii}^{(\mu)} & \tilde{A}_{ii}^{(\mu,\nu)} & \hat{S}_{ib}^{(\mu)} & \tilde{A}_{ib}^{(\mu,\nu)} \\ \tilde{A}_{ii}^{(\nu,\mu)} & \hat{S}_{ii}^{(\nu)} & \tilde{A}_{ib}^{(\nu,\mu)} & \hat{S}_{ib}^{(\nu)} \\ \hline \hat{S}_{bi}^{(\mu)} & \tilde{A}_{bi}^{(\mu,\nu)} & \hat{S}_{bb}^{(\mu)} & \tilde{A}_{bb}^{(\mu,\nu)} \\ \tilde{A}_{bi}^{(\nu,\mu)} & \hat{S}_{bi}^{(\nu)} & \tilde{A}_{bb}^{(\nu,\mu)} & \hat{S}_{bb}^{(\nu)} \end{array} \right] \quad (7.12)$$

Let us specify the newly defined matrices.  $\hat{S}^{(\mu)}$  and  $\hat{S}^{(\nu)}$  are the Schur complements that are the result of factoring the children nodes  $\mu$  and  $\nu$ . With an abuse of notation, we will name them with global indices, even though they are “small” matrices and therefore require local indices. Then,  $\hat{S}_{ib}^{(\mu)}$  indicates  $\hat{S}^{(\mu)}(I^\sigma \cap B^\mu, B^\sigma \cap B^\mu)$  and so on. Similarly, we write  $\tilde{A}_{ib}^{(\mu,\nu)} = A(I^\sigma \cap B^\mu, B^\sigma \cap B^\nu)$  to indicate entries taken from the original matrix. Equation (7.12) makes it evident that there is no overlap between the Schur complements of the children nodes. We identify the same block structure as in (3.6) and the corresponding blocks  $\tilde{A}_{bb}^{(\sigma)}$ ,  $\tilde{A}_{bi}^{(\sigma)}$ ,  $\tilde{A}_{ib}^{(\sigma)}$  and  $\tilde{A}_{ii}^{(\sigma)}$  and write the Schur complement as

$$\begin{aligned} \hat{S}^{(\sigma)} &= \tilde{A}_{bb}^{(\sigma)} - \tilde{A}_{bi}^{(\sigma)} (\tilde{A}_{ii}^{(\sigma)})^{-1} \tilde{A}_{ib}^{(\sigma)} \\ &= \begin{bmatrix} \hat{S}_{bb}^{(\mu)} & \tilde{A}_{bb}^{(\mu,\nu)} \\ \tilde{A}_{bb}^{(\nu,\mu)} & \hat{S}_{bb}^{(\nu)} \end{bmatrix} - \begin{bmatrix} \hat{S}_{bi}^{(\mu)} & \tilde{A}_{bi}^{(\mu,\nu)} \\ \tilde{A}_{bi}^{(\nu,\mu)} & \hat{S}_{bi}^{(\nu)} \end{bmatrix} \begin{bmatrix} \hat{S}_{ii}^{(\mu)} & \tilde{A}_{ii}^{(\mu,\nu)} \\ \tilde{A}_{ii}^{(\nu,\mu)} & \hat{S}_{ii}^{(\nu)} \end{bmatrix}^{-1} \begin{bmatrix} \hat{S}_{ib}^{(\mu)} & \tilde{A}_{ib}^{(\mu,\nu)} \\ \tilde{A}_{ib}^{(\nu,\mu)} & \hat{S}_{ib}^{(\nu)} \end{bmatrix}. \end{aligned} \quad (7.13)$$

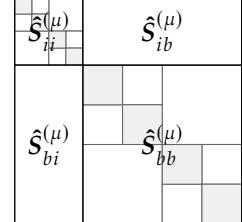
This contains all the products we need to form  $\sigma$ , as well as the Schur complement itself, which we need to pass on to consecutive factorization steps.

Now, if  $\hat{S}^{(\mu)}$  and  $\hat{S}^{(\nu)}$  are HSS matrices, it becomes apparent that we need to extract the submatrices which appear in Equation 7.12.<sup>4</sup> To make this simple, we choose to store the Schur complement of the children nodes  $\mu, \nu$  using a clustering that conforms to the partitioning  $I^\sigma \cap B^\mu, B^\sigma \cap B^\mu$  of  $\sigma$ . This is illustrated in Figure 7.6. This partitioning exposes the submatrix that will be eliminated next in the top left block. Consequently, we can directly access  $\hat{S}_{ii}^{(\mu)}$  and  $\hat{S}_{bb}^{(\mu)}$  as HSS matrices. At the same time, this permits the off-diagonal blocks  $\hat{S}_{ib}^{(\mu)}$  and  $\hat{S}_{bi}^{(\mu)}$  to be extracted as low-rank matrices. In the following, we simply assume that the Schur complements have been compressed in HSS format with the described partitioning. The compression of Schur complements into HSS format and their reordering is explained later in this Chapter.

## Block elimination

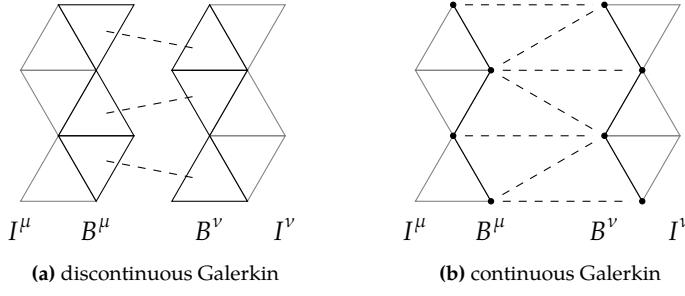
Computing (7.13), and the factors  $\hat{S}^{(\sigma)}, \hat{L}^{(\sigma)}, \hat{R}^{(\sigma)}$  requires the efficient application of the inverse

$$(\hat{A}_{ii}^{(\sigma)})^{-1} = \begin{bmatrix} \hat{S}_{ii}^{(\mu)} & \tilde{A}_{ii}^{(\mu,\nu)} \\ \tilde{A}_{ii}^{(\nu,\mu)} & \hat{S}_{ii}^{(\nu)} \end{bmatrix}^{-1}. \quad (7.14)$$



**Figure 7.6:** HSS block structure of the Schur complement. This partitioning mirrors the structure in the nested dissection to allow easy extraction of submatrices in either HSS or low-rank format.

4: We also could have chosen to compress them in HODLR format.



**Figure 7.7:** Illustration of the connectivity between  $\mu$  and  $\nu$  for finite element discretizations. Dashed lines represent interactions between  $B^\mu \cap I^\sigma$  and  $B^\nu \cap I^\sigma$  and correspond to entries in  $\tilde{A}_{ii}^{(\mu,\nu)}$  and  $\tilde{A}_{ii}^{(\nu,\mu)}$ .

to the right  $X \rightarrow (\hat{A}_{ii}^{(\sigma)})^{-1}X$  and to the left  $X \rightarrow X(\hat{A}_{ii}^{(\sigma)})^{-1}$ . Given the invertible block matrix

$$B = \begin{bmatrix} B_{11} & B_{12} \\ B_{21} & B_{22} \end{bmatrix}$$

and a right-hand side  $X$ , with the corresponding partitioning, we can apply the inverse to the right by using Algorithm 7.1. We can also

```

Partition  $X = \begin{bmatrix} X_1 \\ X_2 \end{bmatrix}$ , corresponding to the blocks in  $B$ 
 $X_1 \leftarrow B_{11}^{-1}X_1$ 
 $X_2 \leftarrow X_2 - B_{21}X_1$ 
Form the Schur complement  $\tilde{S} = B_{22} - B_{21}B_{11}^{-1}B_{12}$ 
 $X_2 \leftarrow \tilde{S}^{-1}X_2$ 
 $X_1 \leftarrow X_1 - B_{11}^{-1}B_{12}X_2$ 
return  $X = \begin{bmatrix} X_1 \\ X_2 \end{bmatrix}$ 

```

construct a similar algorithm to apply the block inverse  $B^{-1}$  to the left. It is clear that an efficient algorithm requires efficient methods to form the Schur complement  $\tilde{S}$ , as well as efficient methods to apply  $B_{11}$ ,  $B_{22}$  and  $\tilde{S}$ .

As mentioned previously, we assume that the Schur complements  $\hat{S}^{(\mu)}$ ,  $\hat{S}^{(\nu)}$  have the block-structure shown in Figure 7.6, which makes the blocks  $\hat{S}_{ii}^{(\mu)}$  and  $\hat{S}_{ii}^{(\nu)}$  readily available in HSS format. The off-diagonal blocks  $\tilde{A}_{ii}^{(\mu,\nu)}$  and  $\tilde{A}_{ii}^{(\nu,\mu)}$  on the other hand, are submatrices of the original matrix and therefore sparse. They represent the interactions between the two children nodes  $\mu$  and  $\nu$ , restricted to the interior of  $\sigma$ .<sup>5</sup> If these matrices can be represented by hierarchical matrices with low off-diagonal ranks, it will allow us to construct an accelerated algorithm for the block inverse.

For finite element discretizations in two dimensions, these matrices are essentially banded. Figure 7.7 depicts the corresponding connectivity for both discontinuous Galerkin and continuous Galerkin discretizations in two dimensions. Because the interaction of elements is limited to the elements directly opposed, we see that the right reordering of  $B^\mu$  and  $B^\nu$  will ensure that  $\tilde{A}_{ii}^{(\mu,\nu)}$  and  $\tilde{A}_{ii}^{(\nu,\mu)}$  are banded. It is important to keep in mind that  $\tilde{A}_{ii}^{(\nu,\mu)}$  only corresponds to the dotted lines in Figure 7.7. Banded matrices, on the other hand, can be well-approximated with hierarchical matrices, if their bandwidth is small.

**Algorithm 7.1:** Right-apply block inverse of  $B$  to  $X$ .

5: The notation with subscripts  $i$  can be misleading here and one might believe that  $\tilde{A}_{ii}^{(\mu,\nu)}$  should be zero. However,  $i$  refers to the interior of the parent node  $\sigma$  and therefore, it contains the boundary interaction of  $B^\mu$  with  $B^\nu$ .

We let  $\mathcal{T}_r^{(\mu)}, \mathcal{T}_c^{(v)}$  denote the row and column cluster trees of  $\hat{S}_{ii}^{(\mu)}$  and  $\hat{S}_{ii}^{(v)}$  respectively and assume that they share the same tree structure, which we might have to enforce algorithmically by pruning the cluster trees. In two dimension, we can guarantee for all  $I_i^l \in \mathcal{T}_r^{(\mu)}$  and  $J_i^l \in \mathcal{T}_c^{(v)}$  that

$$\text{rank } \tilde{A}_{ii}^{(\mu,v)}(I_i^l, J \setminus J_i^l) \leq \text{nnz } \tilde{A}_{ii}^{(\mu,v)}(I_i^l, J \setminus J_i^l) \leq n_{\text{con}}, \quad (7.15a)$$

$$\text{rank } \tilde{A}_{ii}^{(\mu,v)}(I \setminus I_i^l, J_i^l) \leq \text{nnz } \tilde{A}_{ii}^{(\mu,v)}(I \setminus I_i^l, J_i^l) \leq n_{\text{con}}, \quad (7.15b)$$

where  $n_{\text{con}} \in \mathbb{N}_0$  is a small constant. This constant depends on the connectivity of  $A$ , and therefore on the chosen discretization of the problem. In other words, the ranks of each block row and column are bounded, which makes  $\tilde{A}_{ii}^{(\mu,v)}$  and  $\tilde{A}_{ii}^{(v,\mu)}$  HSS matrices of rank  $n_{\text{con}}$  according to Definition 5.4.1. Figure 7.7a illustrates that for discontinuous Galerkin discretization in two dimensions, this constant can be  $n_{\text{con}} = 0$ , if the block structures conforms to the elements of the discretization. Coincidentally, this makes it a block-diagonal matrix. For continuous Galerkin discretizations in two dimensions, as illustrated in Figure 7.7b, the ideal case is  $n_{\text{con}} = 1$ .

The situation is more complicated in three dimensions. For discontinuous Galerkin discretizations in three dimensions, the connectivity is still similar to the situation depicted in Figure 7.7b, as interactions only occur through opposing faces. As a consequence, (7.15) still holds with  $n_{\text{con}} = 0$ . For continuous Galerkin discretizations however,  $n_{\text{con}}$  depends on the size of the matrix and it is evident, whether a similiar argument could be recovered to guarantee that  $\tilde{A}_{ii}^{(v,\mu)}$  is compressible.

Finally, we must take care in the construction of the nested dissection  $\mathcal{E}$ , in order to guarantee that the degrees of freedom are numbered correctly, such that the argument applies. More precisely, we need to ensure that the nodes are numbered similarly across the interface, such that the interactions illustrated in Figure 7.7 are clustered around the diagonal. In practice, however, this can be achieved by simply extending the nested dissection hierarchy in its depth (see Section 8).<sup>6</sup> To construct  $\tilde{A}_{ii}^{(\mu,v)}$  in HSS format, we can simply extract the HSS block diagonal and check how many non-zero entries remain. We can then compress the HSS matrix using the efficient HSS compression Algorithm 6.8 or a custom compression algorithm adapted to the compression of very sparse matrices.

Hence, for FE Galerkin matrices,  $\tilde{A}_{ii}^{(\mu,v)}$  and  $\tilde{A}_{ii}^{(v,\mu)}$  can indeed be expressed as HSS matrices with low HSS rank. We modify Algorithm 7.1 using HSS arithmetic to form the intermediate Schur complement  $\tilde{S}$  in HSS format. This requires the compression of  $\tilde{A}_{ii}^{(\mu,v)}$  and  $\tilde{A}_{ii}^{(v,\mu)}$  with compatible row and column clusters among  $\hat{S}_{ii}^{(v)}, \tilde{A}_{ii}^{(v,\mu)}, \hat{S}_{ii}^{(\mu)}$  and  $\tilde{A}_{ii}^{(\mu,v)}$ . The procedure is described in Algorithm 7.2. The formation of  $A^{-1}B$  in HSS format is based on the efficient inversion using the ULV factorization [45, 47]. This algorithm has a complexity of  $\mathcal{O}(k^2n)$ , where  $n$  is the corresponding matrix size and  $k$  is the maximum HSS rank encountered. Therefore, the overall complexity of Algorithm 7.2 is  $\mathcal{O}(k^2n)$ .

<sup>6</sup>: For discontinuous Galerkin discretizations, we make sure that the blocks  $I_i^l$  at the leaf level conform to the elements of the discretization. This can be done by choosing a HSS block size  $\beta$ , which corresponds to a multiple of the number of degrees of freedom per element. This also makes the algorithm robust in the case that the numbering is not conforming.

```

Partition  $X = \begin{bmatrix} X_1 \\ X_2 \end{bmatrix}$ , corresponding to the blocks in  $\hat{A}_{ii}^{(\sigma)}$ 
Extract clusters  $\mathcal{T}_r^{(\mu)}, \mathcal{T}_c^{(\mu)}$  and  $\mathcal{T}_r^{(v)}, \mathcal{T}_c^{(v)}$  of  $\hat{S}_{ii}^{(\mu)}$  and  $\hat{S}_{ii}^{(v)}$ 
Compress  $\tilde{A}_{ii}^{(\mu,v)}$  into HSS format, conforming to  $\mathcal{T}_r^{(\mu)}$  and  $\mathcal{T}_c^{(v)}$ 
Compress  $\tilde{A}_{ii}^{(v,\mu)}$  into HSS format, conforming to  $\mathcal{T}_r^{(v)}$  and  $\mathcal{T}_c^{(\mu)}$ 
 $X_1 \leftarrow (\hat{S}_{ii}^{(\mu)})^{-1} X_1$ 
 $X_2 \leftarrow X_2 - \hat{A}_{ii}^{(\mu,v)} X_1$ 
Form  $\tilde{S} \leftarrow \hat{S}_{ii}^{(v)} - \tilde{A}_{ii}^{(v,\mu)} (\hat{S}_{ii}^{(\mu)})^{-1} \tilde{A}_{ii}^{(\mu,v)}$  in HSS form
Compute  $X_2 \leftarrow \tilde{S}^{-1} X_2$ 
 $X_1 \leftarrow X_1 - (\hat{S}_{ii}^{(\mu)})^{-1} \hat{A}_{ii}^{(\mu,v)} X_2$ 
return  $X = \begin{bmatrix} X_1 \\ X_2 \end{bmatrix}$ 

```

**Algorithm 7.2:** Right-apply the block inverse of  $\hat{A}_{ii}^{(\sigma)}$  using HSS arithmetic.

## Computing the left and right transforms

We move on to the computation of the factors  $\hat{L}^{(\sigma)}$  and  $\hat{R}^{(\sigma)}$ . The right transform is given by

$$\hat{R}^{(\sigma)} = (\hat{A}_{ii}^{(\sigma)})^{-1} \hat{A}_{ib}^{(\sigma)} = (\hat{A}_{ii}^{(\sigma)})^{-1} \begin{bmatrix} \hat{S}_{ib}^{(\mu)} & \tilde{A}_{ib}^{(\mu,v)} \\ \tilde{A}_{ib}^{(v,\mu)} & \hat{S}_{ib}^{(v)} \end{bmatrix}. \quad (7.16)$$

A close look at  $\tilde{A}_{ib}^{(\sigma)}$  reveals that the diagonal blocks  $\hat{S}_{ib}^{(\mu)}$  and  $\hat{S}_{ib}^{(v)}$  are low-rank as they are off-diagonal blocks extracted from a HSS matrix. The off-diagonal blocks  $\tilde{A}_{ib}^{(\mu,v)}, \tilde{A}_{ib}^{(v,\mu)}$ , on the other hand, are sparse, as they reflect interactions between the interior degrees of freedom  $I^\sigma \cap B^\mu$ , associated with box  $\mu$  and the boundary degrees of freedom  $B^\sigma \cap B^v$  associated with box  $v$ . In two dimensions, it corresponds to interactions in one point. In three dimensions, this interaction corresponds to interactions along a line. Consequently, we can expect these blocks to have very few non-zero entries.

The rank of the right transform  $\text{rank } \hat{R}^{(\sigma)}$  can therefore be bounded by

$$\begin{aligned} \text{rank } \hat{R}^{(\sigma)} &= \text{rank } \hat{A}_{ib}^{(\sigma)} \\ &\leq \text{rank } \hat{S}_{ib}^{(\mu)} + \text{rank } \hat{S}_{ib}^{(v)} + \text{rank } \tilde{A}_{ib}^{(\mu,v)} + \text{rank } \tilde{A}_{ib}^{(v,\mu)} \\ &\leq \text{hssrank } \hat{S}^{(\mu)} + \text{hssrank } \hat{S}^{(v)} + \text{nnz } \tilde{A}_{ib}^{(\mu,v)} + \text{nnz } \tilde{A}_{ib}^{(v,\mu)}, \end{aligned}$$

which implies that  $\hat{R}^{(\sigma)}$  is of low-rank itself if  $\tilde{A}_{ib}^{(\mu,v)}$  and  $\tilde{A}_{ib}^{(v,\mu)}$  are sufficiently sparse. Thus  $\hat{L}^{(\sigma)}$  and  $\hat{R}^{(\sigma)}$  are themselves low-rank matrices with ranks bounded by double the maximum HSS rank encountered among Schur complements plus a small constant. This low-rank property is well-known for matrices arising from Galerkin discretizations of elliptic problems and has been documented and exploited in the literature [68, 77]. Here, we have shown how this property is related to the approximability of Schur complements using hierarchical matrices, another property which has been proven for these matrices [54–56].

Consequently, we can store the factors  $\hat{L}^{(\sigma)}$  and  $\hat{R}^{(\sigma)}$  in low-rank format

represented by their generators. Moreover, as we can apply both  $(\hat{A}_{ii}^{(\sigma)})^{-1}$  and  $\hat{A}_{ib}^{(\sigma)}$  in linear time, we can use a randomized algorithm to compress and store their low-rank representations efficiently in  $\mathcal{O}(k^3 n)$  time [17]. An alternative is to first form a low-rank representation of  $\hat{A}_{ib}^{(\sigma)}$  and then apply the inverse of  $\hat{A}_{ii}^{(\sigma)}$  to the left generator. This is summarized in Algorithm 7.3, which is more efficient in practice.

```

Extract  $X_1 = \begin{bmatrix} \hat{S}_{ib}^{(\mu)} & \mathbf{0} \\ \mathbf{0} & \hat{S}_{ib}^{(v)} \end{bmatrix}$  in low-rank format
Compress  $X_2 = \begin{bmatrix} \mathbf{0} & \hat{A}_{ib}^{(\mu,v)} \\ \tilde{A}_{ib}^{(v,\mu)} & \mathbf{0} \end{bmatrix}$  into low-rank format
Add  $\hat{A}_{ib}^{(\sigma)} \leftarrow X_1 + X_2$  and recompress
Form  $\hat{R}^{(\sigma)} \leftarrow (\hat{A}_{ii}^{(\sigma)})^{-1} \hat{A}_{ib}^{(\sigma)}$  by applying  $(\hat{A}_{ii}^{(\sigma)})^{-1}$  to the generator of
 $\hat{A}_{ib}^{(\sigma)}$ 

```

**Algorithm 7.3:** Form the right transform  $\hat{R}^{(\sigma)}$  in low-rank format.

## Compressing the Schur complement

The final step to complete the factorization of  $\sigma$  in compressed form is to form the Schur complement (7.13) in HSS format, with a block structure which conforms to the degrees of freedom of the parent node of  $\sigma$ , as shown in Figure 7.6. To compute a HSS representation of the Schur complement (7.13) via random sampling (Algorithm 6.8), we require an efficient way to compute  $x \rightarrow \hat{S}^{(\sigma)}x$ ,  $x \rightarrow (\hat{S}^{(\sigma)})^*x$  and  $\hat{S}^{(\sigma)}(i, j)$ .

The first matrix  $\hat{A}_{bb}^{(\sigma)}$  consists of HSS matrices on the diagonal and sparse matrices on the off-diagonal. Therefore, the cost of performing matrix-vector multiplications  $x \rightarrow \hat{A}_{bb}^{(\sigma)}x$  is  $\mathcal{O}(kn + n_{nz}n)$ , where  $n_{nz}$  is the maximum number of non-zero entries per line and  $k$  the maximum HSS rank. On the other hand, the cost of accessing entries is  $\mathcal{O}(\log n_{nz})$ , if  $(i, j)$  lies in the sparse blocks and  $\mathcal{O}(k \log n)$  if it lies in the HSS blocks. The update matrix  $\hat{U}^{(\sigma)}$  can be written as

$$\hat{U}^{(\sigma)} = -\hat{A}_{bi}^{(\sigma)} (\hat{A}_{ii}^{(\sigma)})^{-1} \hat{A}_{ib}^{(\sigma)} = -\hat{A}_{bi}^{(\sigma)} \hat{R}^{(\sigma)}.$$

Because both  $\hat{A}_{bi}^{(\sigma)}$  and  $\hat{R}^{(\sigma)}$  are low-rank matrices, we can compute their product in  $\mathcal{O}(k^2 n)$  operations. Subsequent matrix-vector products can be computed in  $\mathcal{O}(kn)$  time and individual entries can be accessed in  $\mathcal{O}(k)$  time.<sup>7</sup> Consequently, we can compress the Schur complement  $\hat{S}^{(\sigma)}$  efficiently using the randomized algorithm in  $\mathcal{O}(k^2 n \log n)$  operations.<sup>8</sup> Then, the matrix-vector products  $x \rightarrow \hat{S}^{(\sigma)}x$ ,  $x \rightarrow (\hat{S}^{(\sigma)})^*x$ , as well as the access to individual entries can be adapted to incorporate permutations of the degrees of freedom. This allows us to compress the Schur complement  $S^{(\sigma)}$  with a block structure that exposes the correct submatrices for the next factorization steps as depicted in Figure 7.6. As such, we have completed the factorization of  $\sigma$  and we meet all the requirements to continue with the factorization of the parent node.

7: As the update matrices are low-rank, it is possible that an improved algorithm can be designed, which passes on update matrices instead of Schur complements. The blocks designated for elimination can then be compressed and inverted directly.

8: The increased cost for accessing individual entries is what increases the cost of HSS compression to  $\mathcal{O}(k^2 n \log n)$ .

## Forming the factorization

Just as with the direct solvers presented in Section 3.4, we traverse the elimination tree  $\mathcal{E}$  from the bottom up, factoring the nodes in the order prescribed by the elimination tree.

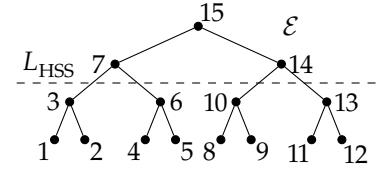
Most of the time, dense arithmetic outperforms HSS arithmetics when the matrices involved are small, as there is typically an overhead associated with the HSS datastructures and algorithms. Therefore, we introduce a switching level  $L_{\text{HSS}}$  and apply regular, structured elimination using dense matrices at all nodes at levels  $l > L_{\text{HSS}}$  below the switching level (see Figure 7.8). At the level at which we switch to HSS arithmetic, we therefore have to compress dense Schur complements to HSS format, which can be handled by Algorithm 6.7.

There is an added benefit to extending the elimination tree below the switching level  $L_{\text{HSS}}$ . The compression of Schur complements to HSS format can be quite sensitive to the order in which the degrees of freedom appear. Extending the elimination tree below  $L_{\text{HSS}}$  acts like a nested-dissection reordering and we observe better compressibility of Schur complements.

We summarize the factorization procedure in Algorithm 7.4. The

```

for all nodes  $\sigma \in \mathcal{E}$  from the bottom up do
  if  $l < L_{\text{HSS}}$  then
    Form  $\hat{\mathbf{A}}_{ii}^{(\sigma)}$  by extracting the corresponding blocks
    Factor  $\hat{\mathbf{A}}_{ii}^{(\sigma)}$  so that 7.2 can be applied
    Compute  $\hat{\mathbf{L}}^{(\sigma)}$  and  $\hat{\mathbf{R}}^{(\sigma)}$  in low-rank format
  else
    Factor  $\hat{\mathbf{A}}_{ii}^{(\sigma)}$  using dense arithmetic
    Form  $\hat{\mathbf{L}}^{(\sigma)}$  and  $\hat{\mathbf{R}}^{(\sigma)}$  using dense arithmetic
  end if
  if  $\sigma$  is the root node then
    Do nothing
  else if  $l = L_{\text{HSS}}$  then
    Form  $\hat{\mathbf{S}}^{(\sigma)}$  densely and compress to HSS form
  else if  $l < L_{\text{HSS}}$  then
    Form  $\hat{\mathbf{S}}^{(\sigma)}$  directly in HSS form via Algorithm 6.8
  else
    Form and store  $\hat{\mathbf{S}}^{(\sigma)}$  as dense matrix
  end if
end for
```



**Figure 7.8:** The switching level  $L_{\text{HSS}}$  determines the nodes  $\mathcal{E}$  which are factored using HSS arithmetic.

**Algorithm 7.4:** Compute approximate factorization  $A \approx LDR = P$ .

factorization  $P = LDR$  is therefore only formed implicitly. Just as we have done for the structured direct solver in Section 3.4, we require an algorithm for applying the inverse  $P^{-1} = R^{-1}D^{-1}L^{-1}$  efficiently to a vector. This can be done by using Algorithm 3.3, switching to HSS arithmetic whenever it is applicable.

## 7.4 Complexity of the algorithm

To determine the overall cost of forming the approximate factorization, let us summarize the computational cost of the various operations encountered for the factorization of each node. Table 7.1 provides an overview of the computational complexity of the individual steps that have to be performed at each node above the switching level. Here,

	operation	operation count
factorization	form $\hat{A}_{bb}^{(\sigma)}, \hat{A}_{bi}^{(\sigma)}, \hat{A}_{ii}^{(\sigma)}$ and $\hat{A}_{ib}^{(\sigma)}$	$\mathcal{O}(kn_l \log n_l)$
	factor $\hat{A}_{ii}^{(\sigma)}$	$\mathcal{O}(k^2 n_l)$
	compute $\hat{L}^{(\sigma)}$ and $\hat{R}^{(\sigma)}$	$\mathcal{O}(k^3 n_l)$
	compress $\hat{S}^{(\sigma)}$	$\mathcal{O}(k^2 n_l \log n_l)$
application	apply $(\hat{A}_{ii}^{(\sigma)})^{-1}$	$\mathcal{O}(kn_l)$
	apply $\hat{L}^{(\sigma)}$ and $\hat{R}^{(\sigma)}$	$\mathcal{O}(kn_l)$
	apply $\hat{S}^{(\sigma)}$	$\mathcal{O}(kn_l)$

**Table 7.1:** Summary of the computational cost for the operations involved in the factorization and application at each node.

we have assumed that the ranks of all low-rank matrices (both  $\hat{L}^{(\sigma)}$  and  $\hat{R}^{(\sigma)}$ , as well as off-diagonal blocks of  $\hat{S}^{(\sigma)}$ ) can be bounded by the maximum HSS rank  $k$ , which is typically the HSS rank of the top-level Schur complement. Then, we use  $n_l$  to denote the size of matrices at level  $l$ . Finally, we have omitted the cost of extracting submatrices from the sparse matrix  $A$ , as this is typically not a main part of the cost of the overall factorization. We also drop any dependency on the number of non-zero entries per column  $n_{nz}$ , which is typically a constant for a given problem. We point out that this may not be the case however, for instance if  $p$ -refinement is considered. The main cost of the factorization is either the compression of the Schur complement  $\hat{S}^{(\sigma)}$  or the formation of the left and right transforms  $\hat{L}^{(\sigma)}$  and  $\hat{R}^{(\sigma)}$ .

Determining the cost of the factorization is mainly a question of inserting the updated cost at each node into the original calculation (3.15). All operations below the switching level  $L_{\text{HSS}}$  are decoupled and therefore, their asymptotic cost is linear in  $n$ . This can be explained by the bottom level of the hierarchy growing proportionally with the overall size of the matrix. As such, we focus on the portion of the elimination tree, which is above the switching level  $L_{\text{HSS}}$ . We recall from (3.14) that the relation between the overall size and the size of matrices at level  $l$  is given by  $n_l \sim 2^{-\frac{d-1}{d}(l-1)} n^{\frac{d-1}{d}}$ , which follows from geometric consideration of the nested dissection.

The overall complexity can be determined by summing over all leaves. For the factorization, the number of operations at each node is  $\mathcal{O}(k^2 n_l \log n_l + k^3 n_l)$ . Therefore, the total number of operations required to form  $P$  is

$$\begin{aligned}
 W &\sim \sum_{l=1}^{L_{\text{HSS}}} 2^{l-1} (k^3 n_l + k^2 n_l \log n_l) \\
 &\lesssim n^{\frac{d-1}{d}} (k^3 + k^2 \log n) \sum_{l=1}^{L_{\text{HSS}}} 2^{\frac{1}{d}(l-1)} \sim n^{\frac{d-1}{d}} (k^3 + k^2 \log n) 2^{\frac{1}{d}L_{\text{HSS}}} \\
 &\sim k^2 n \log n + k^3 n,
 \end{aligned} \tag{7.17}$$

where we have used that  $L_{\text{HSS}} \sim \log n$ . We conclude that the approximate factorization  $\mathbf{P}$  can be formed in quasilinear time  $\mathcal{O}(k^2 n \log n + k^3 n)$ , under the assumptions that we have made in Section 7.3 and assuming a constant rank  $k$ .<sup>9</sup> A similar calculation can be carried out for the cost of applying the preconditioner, which results in the linear complexity of  $\mathcal{O}(kn_l)$ .

<sup>9</sup>: In practice, the off-diagonal ranks  $k$  have a non-trivial dependency on the overall size of the matrix  $n$ . This dependency is problem-dependant and therefore omitted in this discussion. For realistic cost estimates however, this needs to be taken into consideration.

# 8

## Numerical Experiments

We present numerical results obtained with the hierarchical preconditioner/direct solver. We are mainly interested in investigating three questions. First of all, we wish to investigate the performance of our method as a preconditioner, especially with respect to  $h$ - and  $p$ -refinement as these are typically valid questions that arise in the practical use of finite elements. For wave problems, we are also interested in how this method behaves with increasing wavenumbers, as this is known to be a challenging problem for preconditioning due to the indefinite nature of the problem. We are also interested in verifying the computational complexity of forming and applying the factorization. Finally, we recall that the computational cost is also a function of the maximum off-diagonal rank  $k$ , encountered in the factorization. As such, we wish to investigate the scaling behavior of the ranks in the aforementioned scenarios.

As the main goal is to consider the use as a preconditioner, performance will be mainly measured in terms of GMRES performance. To this end, we use the restarted version of the GMRES Algorithm 4.2, where the Arnoldi vectors are recomputed every 10 iterations. We consider the number of iterations  $i$ , required for the relative residual of the solution  $x_i$  to meet a specified threshold  $\epsilon_{\text{sol}}$ , such that

$$\|P^{-1}Ax_i - P^{-1}b\|_2 \leq \epsilon_{\text{sol}}\|P^{-1}b\|_2. \quad (8.1)$$

We terminate it once a relative residual smaller than  $\epsilon_{\text{sol}} = 10^{-9}$  is achieved. Alternatively, we terminate the computation if a maximum of 30 GMRES iterations is exceeded.

### 8.1 Parameters

To avoid discussing parameters for each experiment, let us discuss the standard parameters that we use for most experiments. Throughout this chapter we use HSS compression with a tolerance of  $\epsilon_{\text{HSS}} = 10^{-6}$  to compress Schur complements. We use this tolerance both in the absolute and relative sense. This implies that we keep refining until the tolerance is met, either in the absolute or the relative sense. In a similar way, we use  $0.5 \cdot \epsilon_{\text{HSS}}$  as the tolerance for the compression of the low-rank approximations of  $\hat{L}^{(\sigma)}$  and  $\hat{R}^{(\sigma)}$ . The block-size of the HSS compression is adapted to match the number of degrees of freedom in each element. In two dimensions, we use

$$\beta = 10 \frac{(p+1)(p+2)}{2},$$

which corresponds to the number of degrees of freedom in 10 elements if a discontinuous Galerkin approximation of order  $p$  is used. Similarly,

8.1 Parameters . . . . .	81
8.2 Poisson problem . . . . .	82
8.3 Helmholtz problem . . . . .	84
Nonstandard domains . . . . .	87
Heterogeneous problems . . . . .	88
Elastic wave equation . . . . .	89
Frequency-domain . . . . .	90
8.4 Scaling and performance . . . . .	92
8.5 Codes for reproducibility . . . . .	95
8.6 Concluding remarks . . . . .	96

we use

$$\beta = 180 \frac{(p+1)(p+2)(p+3)}{6}$$

in three dimensions, corresponding to 180 elements. These choices are based on our own experiences and are therefore unlikely to be optimal. For optimized implementations, the block size should ideally be adapted using a heuristic to guarantee that blocks are only compressed if there is a performance benefit.

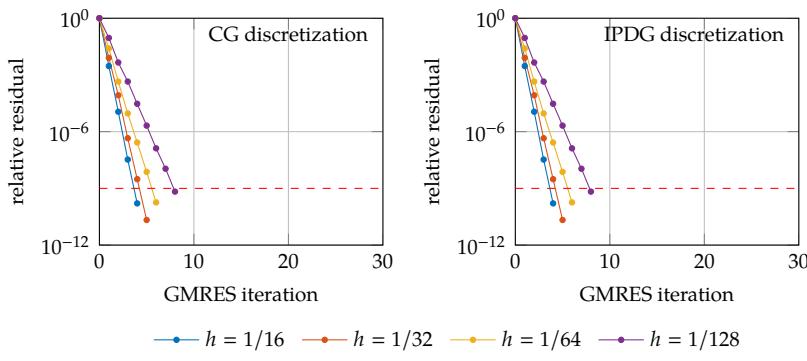
As discretizations, we consider both continuous Galerkin (CG) and interior penalty discontinuous Galerkin (IPDG), discussed in [78]. The domain is typically discretized using a triangular mesh and the nested dissection is generated by recursively bisecting the mesh. For IPDG formulations, we proceed with this until there are less than 10 elements in each box, for CG until there are less than 20 elements per box. The depth of the hierarchy  $L_{\text{HSS}}$ , which determines performance, is typically chosen to start 4 levels from the bottom of the nested dissection hierarchy. In this way, we create an adaptive hierarchy which becomes deeper under  $h$ -refinement.

## 8.2 Poisson problem

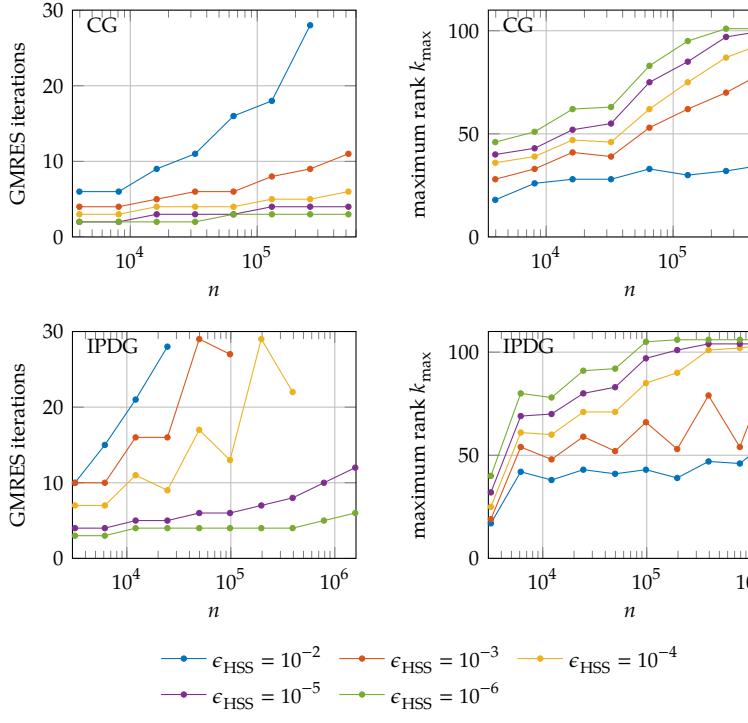
We consider the Poisson problem (1.4) on the unit square  $\Omega \subset \mathbb{R}^2$  in two dimensions, with homogeneous Dirichlet boundary conditions  $g_D = 0$  on the entire boundary and right-hand side  $f = 1$ .

The first thing that we would like to investigate is how the performance of the direct solver is affected by  $h$ - and  $p$ -refinement, as those are the major mechanisms for increasing the fidelity of finite element approximations. Moreover, we seek to understand whether we can expect consistent performance irrespective of whether a continuous or discontinuous Galerkin approximation is chosen. Figure 8.1 depicts the relative residual over each GMRES iteration for various discretizations with a polynomial degree of  $p = 4$ . Clearly, the preconditioner does its job and we observe swift convergence to the solution. From this figure alone, we do not observe any systematic difference between CG and IPDG discretizations.

To study things systematically, we repeat the above experiment for various values of  $1/h$  and  $\epsilon_{\text{HSS}}$  and record the number of GMRES



**Figure 8.1:** Relative residual for each iteration of the preconditioned GMRES applied to the Poisson problem. The figure on the left depicts the case of a CG discretization with  $p = 4$ , the figure on the right shows results obtained with an IPDG discretization of order  $p = 4$ .

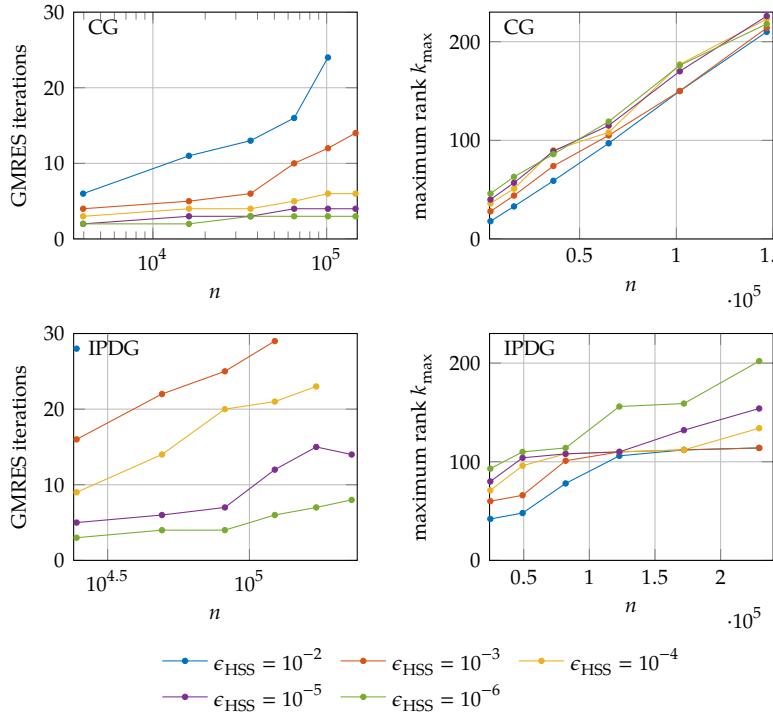


**Figure 8.2:** Preconditioner performance under  $h$ -refinement for the Poisson problem. All experiments use a polynomial order of  $p = 1$  and the values for  $1/h$  range from

iterations required to achieve the prescribed tolerance  $\epsilon_{\text{sol}}$ . In addition, we record the largest rank  $k_{\max}$  encountered in either the off-diagonal blocks of the Schur complements  $\hat{S}^{(\sigma)}$  or the low-rank approximations of  $\hat{\mathbf{L}}^{(\sigma)}, \hat{\mathbf{R}}^{(\sigma)}$ . This usually corresponds to double the HSS rank of the top-level Schur complement. Figure 8.2 depicts the results of this study. Right away, we observe that it is important to control the quality of the compression via  $\epsilon_{\text{HSS}}$  and to ensure that the approximate factorization does a good job of approximating  $\mathbf{A}$ . Moreover, there is an upwards trend in the number of iterations with decreasing  $h$ . This can be explained by the increasing depth of the hierarchy  $L_{\text{HSS}}$ . More precisely, we introduce errors due to the approximative nature of the compressed Schur complements. These cause further errors in the next elimination step and lead to an accumulation of errors. We can expect this effect to be more pronounced with increased depth  $L_{\text{HSS}}$ . Consequently, we have to overcome this effect by adapting the compression tolerance  $\epsilon_{\text{HSS}}$ .

An encouraging trend for the ranks  $k_{\max}$  is that they all seem to converge towards the same value as we increase the compression accuracy. This implies that the singular values decay abruptly once this rank is reached and we therefore do not have to further increase the compression accuracy to guarantee convergence. It is perhaps even more remarkable that this value coincides for CG and IPDG discretizations. This suggests that the nature of the finite element discretization itself does not play an important role for the performance of our method. Moreover, the growth of the ranks seem to slow down with increasing  $n$ .

To have a complete picture, we repeat the experiment, this time with  $p$ -refinement, which is depicted in Figure 8.3. Similar to what we have observed for  $h$ -refinement, it is clear that we need to control the quality of the approximation via  $\epsilon_{\text{HSS}}$ , to ensure that the preconditioned GMRES converges fast. This time around, the ranks grow more substantially, in



**Figure 8.3:** Preconditioner performance under  $p$ -refinement for the Poisson problem. All experiments were performed with  $1/h = 64$  and polynomial orders  $p = 1, 2, \dots, 6$ .

what appears to be linear growth. Moreover, the behavior is comparable for both types of discretizations.

Overall, it seems that the method works better with CG discretizations as with IPDG discretizations. This observation is hardly conclusive, however, as there are many factors which play into this. This includes the actual error of the discretization, the spectrum of off-diagonal blocks in the resulting matrix and the different connectivities in the matrices. We do remark, however, that the trends seem to be quite similar across the two types of discretizations and we therefore conjecture that observations for one type of discretization also have some validity for the other.

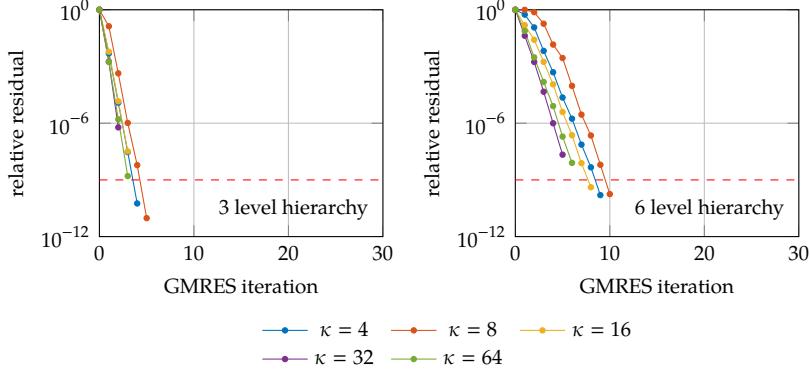
### 8.3 Helmholtz problem

We move onto the Helmholtz problem 1.1.1 and indefinite, elliptic operators. We keep the domain, discretization, boundary conditions and right-hand side the same as before. The discretization of oscillatory problems requires some extra care in the choice of  $p$  and  $h$ . This is mainly due to the pollution effect, where the finite element error is dominated by a so-called pollution term, whose contribution can be understood as the phase difference between the correct solution and its numerical approximation [79, 80]. We use the rule of thumb for standard techniques, which is to have a minimum of 10 grid points per wavelength [81].<sup>1</sup>

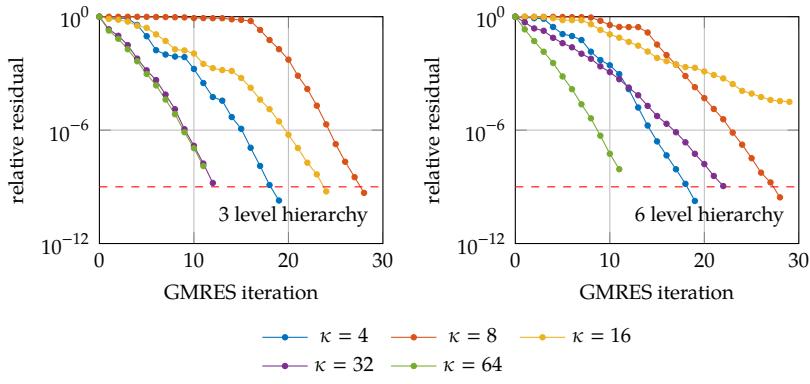
The following results are obtained with preconditioned GMRES for an IPDG discretization of the Helmholtz problem with  $p = 2$  and  $h = 1/64$  for exponentially increasing wave numbers ranging from  $\kappa = 4$  to  $\kappa = 64$ . For all wavenumbers, the preconditioner is able to ensure fast convergence. This includes the higher wavenumbers, which are typically more difficult to handle with conventional preconditioning

1: We remind ourselves that the wavelength is given by

$$\lambda = \frac{2\pi}{\kappa} = \frac{2\pi c}{\omega}.$$



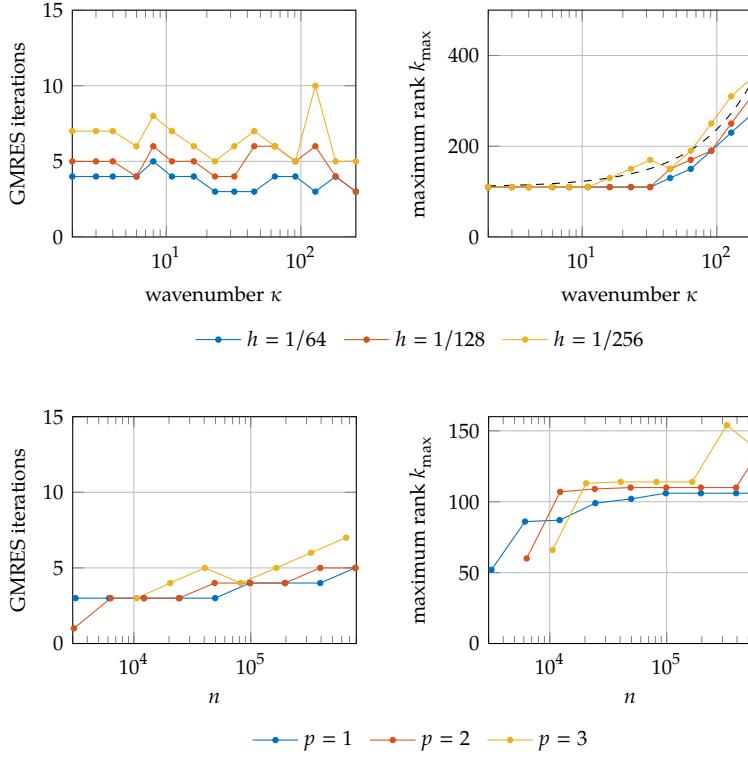
**Figure 8.5:** Relative residual at each iteration of preconditioned GMRES for the IPDG discretization of the Helmholtz problem 1.3 with  $h = 1/64$  and  $p = 2$ . The tests were performed using 3- and 6-level hierarchies and a compression tolerance of  $\epsilon_{\text{HSS}} = 10^{-6}$ .



**Figure 8.6:** Relative residual at each iteration of preconditioned GMRES for the IPDG discretization of the Helmholtz problem 1.3 with  $h = 1/64$  and  $p = 2$ . The figure shows results with a decreased accuracy of  $\epsilon_{\text{HSS}} = 10^{-4}$ .

techniques. We make the observation that the performance decreases as the depth of the hierarchy increases. We have already observed this indirectly with the Poisson problem, where we saw an upwards trend when performing  $h$ -refinement and the associated increase in the depth of the nested dissection hierarchies. This is most likely due to the increased accumulation of errors that comes with deeper hierarchies. We can amplify this effect by decreasing the compression tolerance to  $\epsilon_{\text{HSS}} = 10^{-4}$ . The results are depicted in Figure 8.6. These results confirm the notion that the compression tolerance has to be carefully chosen to ensure an accurate approximation by the preconditioner. This also depends on the depth of the hierarchy and therefore on the size of the matrix, as we have already observed for the Poisson problem. Curiously, it seems that the preconditioner fares better with higher wavenumbers when it comes to increasingly deep hierarchies.

One of the most pressing questions for wave problems is how the method performs in the limit of high wavenumbers. Figure 8.7 depicts the number of GMRES iterations required to achieve a tolerance of  $10^{-9}$ , as well as the maximal rank  $k_{\max}$  for exponentially increasing wavenumbers ranging from  $\kappa = 2$  to  $\kappa = 256$ . The encouraging part is that the number of iterations remain constant, even for problems with very high wavenumbers. As for the ranks, we observe that they slowly increase as the wavenumbers grow. This relation is roughly linear<sup>2</sup> and the maximum HSS rank of the Schur complements  $k_{\max}/2$  can be



**Figure 8.7:** Preconditioner performance at exponentially increasing wave numbers and  $p = 2$ . On the left, the number of GMRES iterations to reach a tolerance of  $10^{-9}$  are depicted. On the right, the maximum among HSS ranks of Schur complements and ranks of the Gauss transforms  $k_{\max}$  is shown.

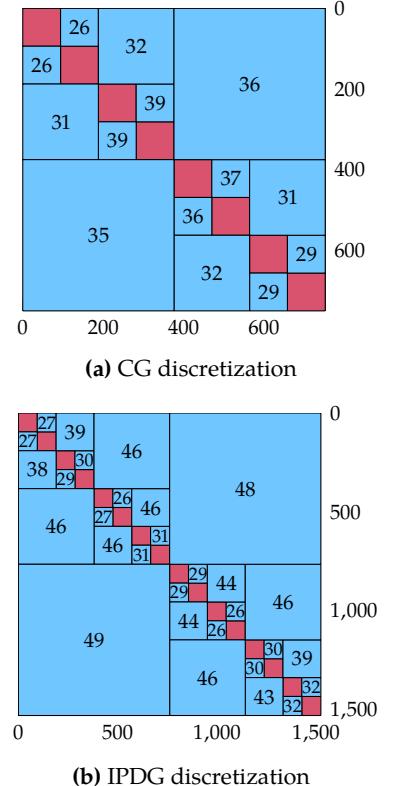
approximated by the relation

$$\max_{\sigma \in \mathcal{E}} \text{hssrank}(\hat{\mathcal{S}}^{(\sigma)}) = \frac{k_{\max}}{2} \approx 55 + 2 \frac{l}{\lambda} = 55 + 2 \frac{l\kappa}{2\pi}, \quad (8.2)$$

where  $l$  is the width of the domain. This result is remarkably consistent with results stated in [25, 63]. This estimate is also shown in Figure 8.7 as dashed line. An intuitive explanation for this increase in ranks is that the dissipative nature of the elliptic operator only applies to high-frequency components of the solution which are above the frequency of the solution. An increase in the wavenumber therefore corresponds to less information which is dissipated and therefore, higher ranks [25].

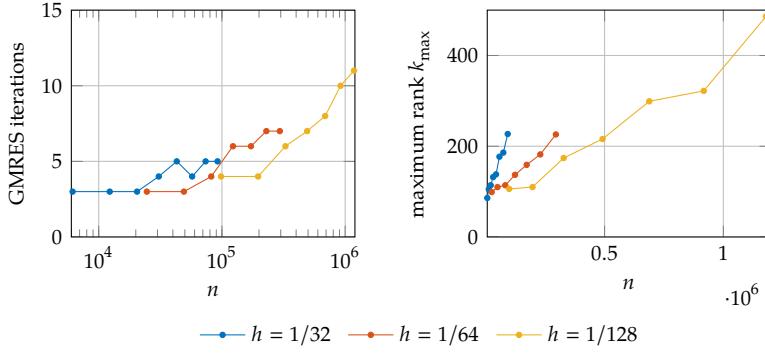
We proceed to examine the performance for Helmholtz problems under  $h$ - and  $p$ -refinement. To this end, we run our experiments with varying discretizations with a fixed wavenumber of  $\kappa = 19.5$ . Figure 8.8 shows the number of iterations and the maximal ranks under  $h$ -refinement. As with the Poisson problems, we note that there is only a slight increase in the number of iterations, which again can be attributed to the increasing depth of the nested dissection hierarchy. Similarly, we observe only a slight increase of the ranks, which is roughly logarithmic in relation to the overall number of degrees of freedom. These results suggest that the method remains efficient as both ranks and iterations grow slowly compared to the problem size. This hypothesis is further investigated in Section 8.4. Figure 8.9 shows results for  $p$ -refinement obtained with varying mesh widths of  $h = 1/32$ ,  $h = 1/64$  and  $h = 1/128$ . We increase the polynomial degree from 1 to 8 for  $h = 1/32$ , and from 1 to 7 for  $h = 1/64$  and  $h = 1/128$ . Again, we observe that the rank growth is approximately linear in the degrees of freedom.

**Figure 8.8:** Preconditioner performance for the Helmholtz problem under  $h$ -refinement.  $n$  is the number of degrees of freedom.



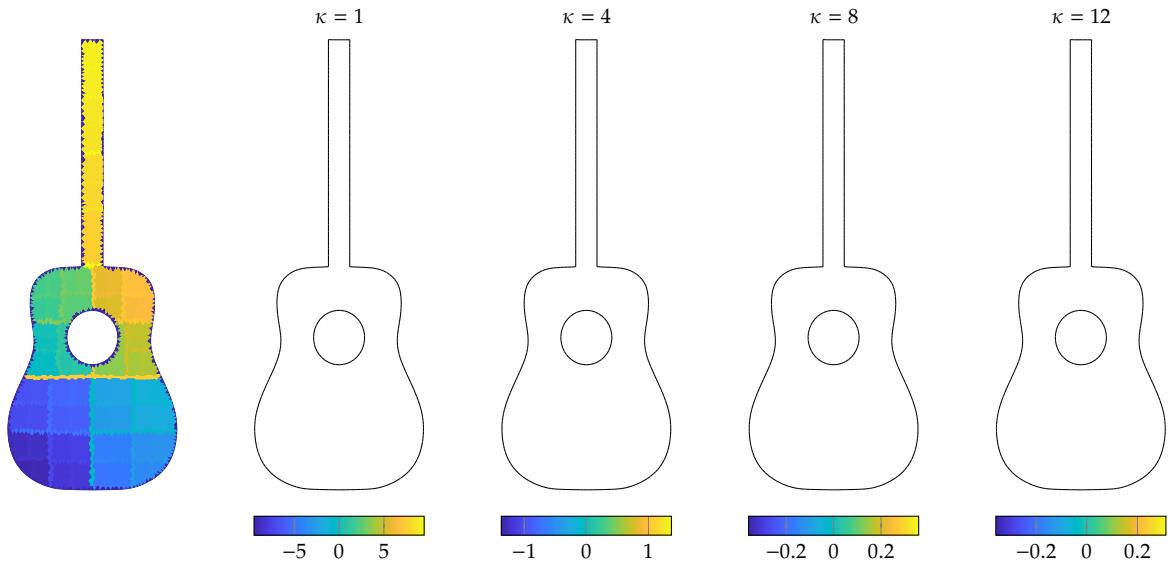
**Figure 8.4:** Top-level Schur complements for the Helmholtz problem with  $h = 1/64$ ,  $p = 2$ ,  $\kappa = 32$  and  $\epsilon_{\text{HSS}} = 10^{-6}$ .

2: Note that the axis for  $\kappa$  is logarithmic



**Figure 8.9:** Preconditioner performance for the Helmholtz problem under  $p$ -refinement.  $n$  is the number of degrees of freedom.

## Nonstandard domains

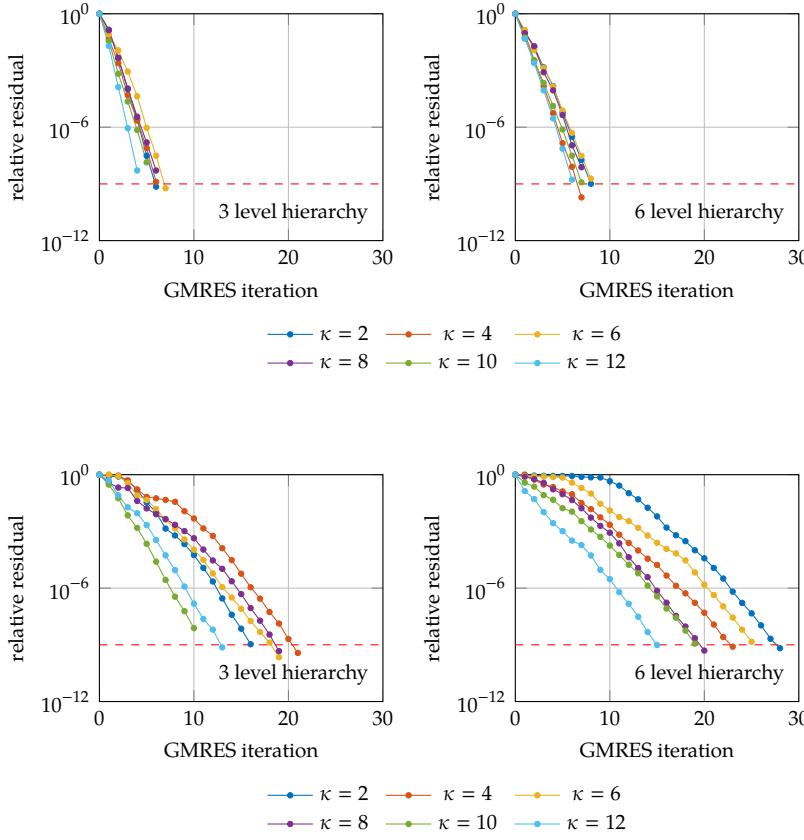


**Figure 8.10:** Solution of the Helmholtz problem 1.3 on a guitar shaped domain. The figure on the left shows the nested dissection reordering. The remaining figures illustrate solutions obtained with the wavenumbers 1, 4 and 8.

So far, we only considered problems formulated on the square domain  $\Omega = [-1, 1]^2$ . To understand how our method performs on non-standard domains, we apply it to the Helmholtz problem 1.1.1 formulated on a guitar shaped domain, shown in Figure 8.10. This domain has a height of  $H = 40$  and a width of 16. We select an average mesh width of  $h_0 = H/100$  and a polynomial degree of 4. This amounts to a total of 71760 degrees of freedom. We choose zero Dirichlet boundary conditions  $g_D = 0$  on the exterior of the guitar and zero Neumann boundaries  $g_N = 0$  at the sound hole. The constant right-hand side is kept constant  $f = 1$ , which corresponds to a uniform excitation on the whole domain.

The nested dissection of the domain is depicted on the left of Figure 8.10. Different colors imply different supernodes in the elimination tree. On the right, solutions for the wavenumbers  $\kappa = 1, 4, 8, 12$  are depicted.

We investigate how the method performs with varying wavenumbers. Figure 8.11 shows the residual history of preconditioned GMRES for various wavenumbers and for various depths of the preconditioner hierarchy (the HSS switching level). Figure 8.12 depicts the situation for a reduced accuracy of  $\epsilon_{\text{HSS}} = 10^{-5}$ . This time, the effect observed in



**Figure 8.11:** Relative residual for various wave numbers on the guitar shaped domain with  $h = 1/100$  and  $p = 4$  and a compression tolerance of  $10^{-6}$ . In this example we only control the compression in the relative sense.

**Figure 8.12:** Relative residual at for various wave numbers on the guitar shaped domain with  $h = 1/100$  and  $p = 4$  and a compression tolerance of  $10^{-5}$ . In this example we only control the compression in the relative sense.

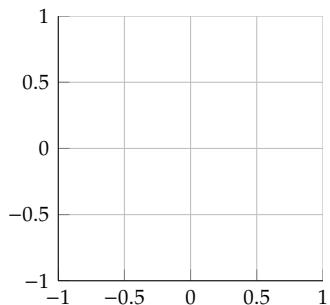
Figure 8.5 is more pronounced. This emphasizes the need to control the compression tolerance in order to guarantee good results. An interesting observation is that this effect seems to affect lower frequencies more strongly than high frequencies.

## Heterogeneous problems

We consider heterogeneous problems in which material coefficients vary in space. One such example is the heterogeneous Helmholtz problem

$$-\nabla \cdot (\mu \nabla u) - \omega^2 u = f, \quad (8.3)$$

subject to suitable boundary condition and right-hand side. Here  $\mu : \Omega \rightarrow \mathbb{R}$  is a material distribution governing the wavespeed  $c = \sqrt{\mu}$ . This problem can be particularly challenging if there are high contrasts in the material distribution [82]. One approach to overcome this is to adapt the nested dissection so that separators conform to the high contrast interfaces. To test this hypothesis, we formulate the heterogeneous Helmholtz problem on a square domain, with piecewise constant  $\mu$ . We choose  $\mu$  to conform to the elements and generate disparate zones with different material coefficients. To ensure that these form large coherent zones, we generate them by picking random points on the unit square and then growing them outwards like a crystal until they touch. Figure 8.13 depicts such a material distribution with 5 different zones.



**Figure 8.13:** Randomly generated heterogeneous zones in the case of five different values for  $\mu$ .

$\mu \in$	regular dissection $k_{\max}$	iters	conforming dissection $k_{\max}$	iters
{0.6861, 0.7533, 1.2181}	77	3	108	4
{ $10^{-1}$ , 1, $10^1$ }	78	4	103	4
{ $10^{-2}$ , 1, $10^2$ }	77	5	115	4
{0.5585, 1.2581, 1.3635, 1.3395, 0.6693}	77	3	102	4
{ $10^{-2}$ , $10^{-1}$ , 1, $10^1$ , $10^2$ }	81	5	101	6
{ $10^{-4}$ , $10^{-2}$ , 1, $10^2$ , $10^4$ }	64	-	76	25

To test both variants of the nested dissection, we use an IPDG discretization of  $p = 2$  and  $1/h = 32$  with a constant wavenumber of  $\kappa = 16$ . We use homogeneous boundary conditions and a constant right-hand side. Figure 8.14 depicts the numerical solutions with 3 and 5 zones respectively, where  $\mu$  takes the values  $\{0.6861, 0.7533, 1.2181\}$  and  $\{0.5585, 1.2581, 1.3635, 1.3395, 0.6693\}$ .

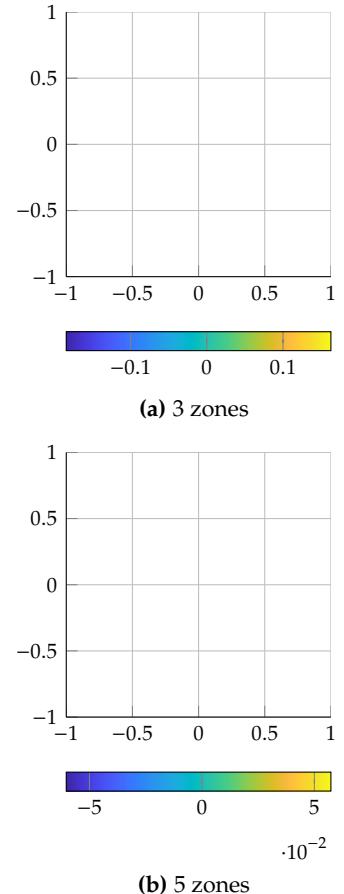
We compare both methods for generating a nested dissection. To test the methods, we compare GMRES performance across a range of contrast ratios for  $\mu$ . Table 8.1 reports our results for a wavenumber of  $\kappa = 16$ . We observe that for moderate to high contrast ratios, there is no benefit of using the conforming dissection method. Conversely, it seems that the conforming dissection leads to an increase in ranks, which can be attributed to the separators not being regular in shape. By this we mean that the separators are far away from being circles, which is known to reduce the approximability using hierarchical matrices. This is related to the approximate separability of the Green's function [59], and its relation to geometric properties of the underlying domain as discussed in Chapter 5. As a consequence, we can regard the problem as finding a balance between shape-regularity of the separators and conforming to the high-contrast interfaces. The results presented in Table 8.1 clearly indicate that there is little to no payoff for the latter, unless contrasts are very high. However it is questionable whether such a discretization and problem formulation makes sense in these situations. Similar experiments were carried out at various other wavenumbers up to  $\kappa = 64$ , with similar outcome.

## Elastic wave equation

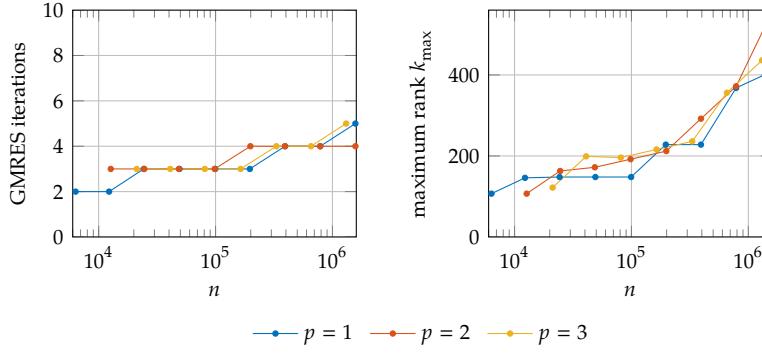
To test our method on systems of partial differential equations, we apply it to an IPDG discretization of the static elastic wave equations. The specific type of the discretization can be found in [83, 84]. We consider a heterogeneous problem on the square domain  $\Omega = \Omega_1 \cup \Omega_2$ , with  $\Omega_1 = [-1, 0] \times [-1, 1]$  and  $\Omega_2 = (0, 1] \times [-1, 1]$ , where the material parameters are constant in each of the respective subdomains:  $\mu = \mu_1$ ,  $\lambda = \lambda_1$  in  $\Omega_1$  and  $\mu = \mu_2$ ,  $\lambda = \lambda_2$  in  $\Omega_2$ . We choose  $\mu_1 = 1$ ,  $\mu_2 = 2$ ,  $\lambda_1 = 1$  and  $\lambda_2 = 2$ . As the source term we choose the constant  $f = [0, 1]^T$  and we set zero Dirichlet boundary conditions on all sides. The mesh is a regular mesh and conforms to the interface in the center of the domain.

We test the performance of the preconditioner on the described problem under  $h$ - and  $p$ -refinement. Figure 8.15 and Figure 8.16 depict the respective results which are very similar to the results obtained for the Helmholtz problem, however with roughly double the ranks. Because the system has two components, the problem is double the size and we

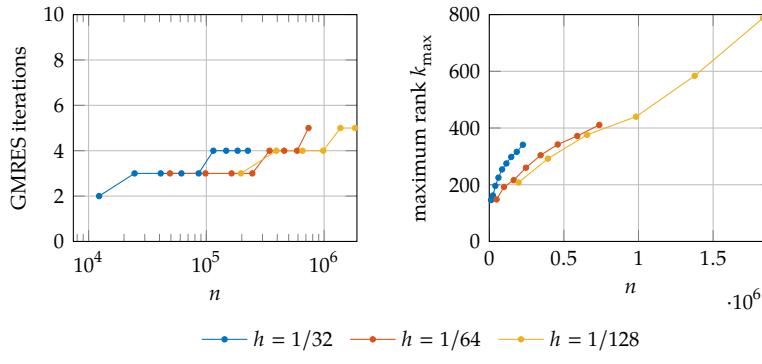
**Table 8.1:** Comparison of both dissection methods for the heterogeneous Helmholtz problem with  $\kappa = 16$ .



**Figure 8.14:** Solution of the Helmholtz problem with heterogeneous material coefficients. Both figures depict solutions for  $\kappa = 16$  with 3 and 5 zones.



**Figure 8.15:** Preconditioner performance for the elastic wave equations (1.7) under  $h$ -refinement.



**Figure 8.16:** Preconditioner performance for the elastic wave equations (1.7) under  $p$ -refinement.

have roughly the same relative ranks as in the case of the Helmholtz problem.

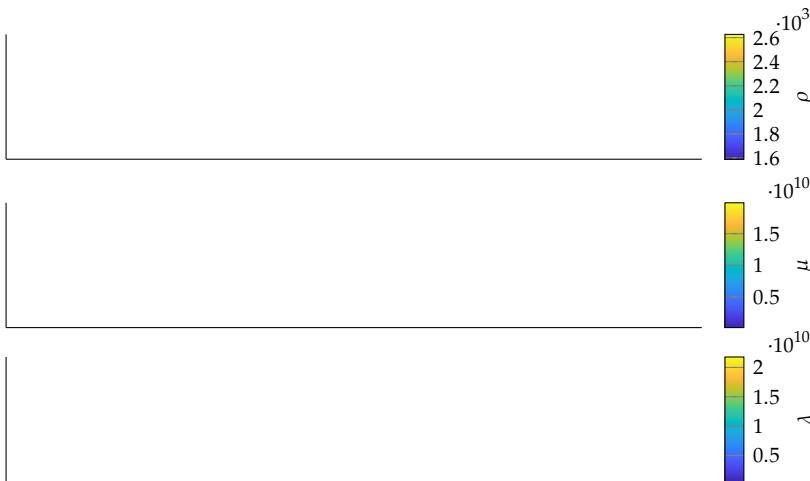
### Frequency-domain elastic wave equation

To test our method on more application-oriented problems, we apply it to IPDG discretizations of the frequency-domain elastic wave equation (1.8). It corresponds to the eigenvalue problem of the elastic wave equations. Problems such as these arise in the context of subsurface modeling and direct waveform inversion [5]. We consider the Marmousi II velocity model [85], which is a common benchmark test. It models soil deposits off the coast of Madagascar, measured using seismic imaging techniques akin to waveform inversion. The original model is 17km wide and 3.5km deep. We use the soil portion of the model and represent it in the computational domain  $\Omega = [0, 17000] \times [-3500, -450]$ . As boundary conditions, we choose zero Dirichlet boundary conditions  $g_D = 0$  on the bottom and zero Neumann boundary conditions  $g_N = 0$  on the remaining three sides. As source term we use a dipole of the form

$$f = (\mathbf{r} - \mathbf{r}_s) \exp \frac{\|\mathbf{r} - \mathbf{r}_s\|^2}{2R^2},$$

where  $\mathbf{r} = [x, y]^T$  is the radius vector,  $\mathbf{r}_s = [-1250, 8500]^T$  the source location and  $R = 100$  is the width of the dipole.

For our experiments we use two meshes. The first mesh is generated using a segmentation of the material distributions, to conform to the sharp interfaces in the material parameters. The other mesh is a simple, regular mesh. We use meshes with approximately 40 or 80 elements



**Figure 8.17:** Material distribution of  $\rho$ ,  $\mu$  and  $\lambda$  for the Marmousi II test case. The domain is discretized using the IPDG method with  $h = 3050/40$  and  $p = 1$ .

in the vertical direction, which amounts to a total of 54642 or 186868 elements in the case of the conforming mesh. The elimination tree is again generated by hierarchical subdivision, while keeping the aspect ratio of the boxes close to one.

For the conforming mesh, the material distributions  $\rho$ ,  $\mu$  and  $\lambda$  are approximated by piecewise constant functions in each element. Figure 8.17 depicts these material distributions on the conforming mesh. For the regular mesh, the material distributions are approximated within the discontinuous Galerkin function space on the mesh.

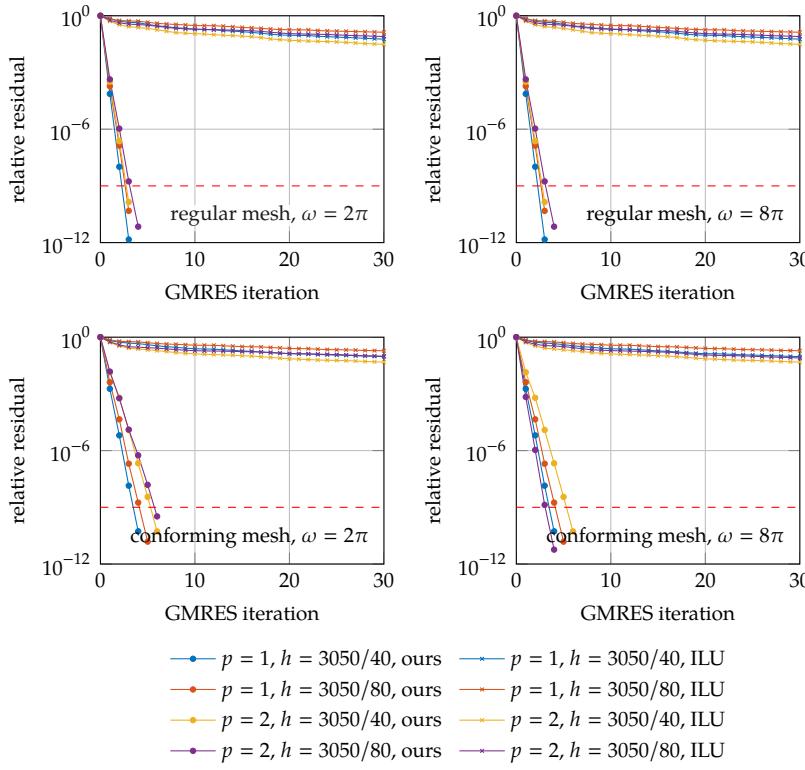
We solve the problem for two frequencies,  $\omega = 2\pi$  and  $\omega = 8\pi$ . The components  $u_x$  and  $u_y$  of  $\mathbf{u}$  are shown for  $\omega = 2\pi$  in Figure 8.18. Solving such



**Figure 8.18:**  $x$ - and  $y$ -components of the solution at a frequency of  $\omega = 2\pi$ .

problems can be challenging as it includes many elongated high-contrast interfaces which are typically hard to precondition. The encouraging results in Section 2 indicate that we can expect good results without having to conform to material interfaces with our nested dissection.

Figure 8.19 shows the relative residual at each GMRES iteration for both frequencies. For the purpose of comparison, we apply the incomplete LU factorization (ILU) as a preconditioner, as it represents a popular, general-purpose preconditioning technique. We observe that using our method, we are able to precondition the problem and achieve a satisfying convergence rate, while the ILU preconditioner fails. This illustrates the robustness of the hierarchical preconditioner. The only problem-specific knowledge that is required is a hierarchical partitioning of the domain. If this partitioning is badly chosen, the method might fail or deteriorate due to Schur complements not being compressible. As we have computed the nested dissection based purely on the aspect ratio of the bounding



**Figure 8.19:** Performance of the preconditioner compared to ILU. Number of GMRES iterations for  $\omega = 2\pi$  and  $\omega = 8\pi$  with non-conforming and conforming discretizations.

boxes, we have demonstrated that it is quite robust, even if there are high-contrast interfaces in the domain.

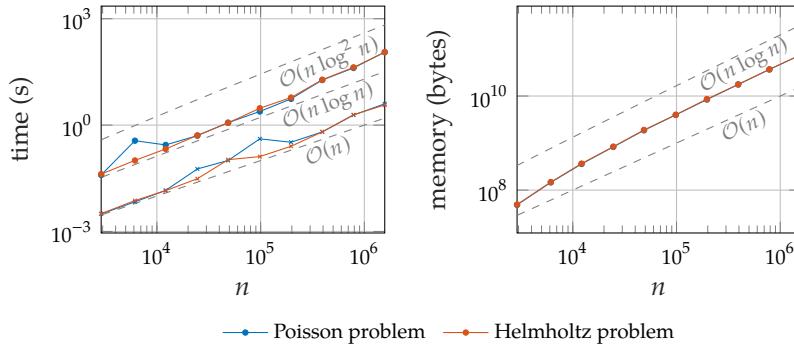
## 8.4 Scaling and performance

One of the claims that we have made is that the approximate factorization can be formed in quasilinear time, assuming that the ranks stay more or less constant. We seek to validate these claims through timings of our reasonably optimized, single-threaded JULIA code.<sup>3</sup> Table 8.2 and Table 8.3 list timings, as well as memory requirements for forming and applying the factorization. The experiments were carried out for the two-dimensional Poisson and Helmholtz problems respectively, on meshes ranging from  $h = 1/22$  to  $h = 1/512$  with  $p = 1$ . The corresponding data is also depicted in Figure 8.20. We observe that the cost of application, construction and memory requirements roughly double from row to row. This is consistent with the (quasi-)linear complexity postulated in Section 7.4. With increasing problem size, a bigger portion of the

3: All of the following performance figures were obtained on a 2019 MacBook Pro with an Intel i9 Processor clocked at 2.3 GHz and 32GB of RAM.

$1/h$	$n$	$L_{\text{HSS}}$	application	factorization	memory	iters	$k_{\max}$
22	2904	4	0.00311 s	0.03921 s	$4.8530e7$ B	3	44
32	6144	5	0.00684 s	0.36351 s	$1.4575e8$ B	3	79
45	12150	7	0.01442 s	0.27612 s	$3.5953e8$ B	3	81
64	24576	7	0.05801 s	0.50404 s	$8.2917e8$ B	4	93
90	48600	9	0.10054 s	1.17373 s	$1.8655e9$ B	4	100
128	98304	9	0.41032 s	2.42477 s	$3.9659e9$ B	4	106
181	196566	11	0.32823 s	5.50095 s	$8.4243e9$ B	4	106
256	393216	11	0.64399 s	18.6995 s	$1.7571e10$ B	4	106
362	786264	13	1.91412 s	40.9561 s	$3.6662e10$ B	6	106
512	1572864	14	4.00240 s	114.050 s	$7.5003e10$ B	6	106

**Table 8.2:** Factorization and application times, as well as memory consumption for the Poisson problem under  $h$ -refinement in two dimensions.



**Figure 8.20:** Timings and memory requirements of the preconditioner under  $h$ -refinement in two dimensions. The figure on the left shows factorization times on the top and application times on the bottom, while the figure on the right shows memory requirements.

$1/h$	$n$	$L_{\text{HSS}}$	application	factorization	memory	iters	$k_{\max}$
22	2904	4	0.00318 s	0.04152 s	$4.9294e7$ B	3	48
32	6144	5	0.00744 s	0.10067 s	$1.4819e8$ B	3	81
45	12150	7	0.01418 s	0.21265 s	$3.6590e8$ B	3	84
64	24576	7	0.03072 s	0.51274 s	$8.3719e8$ B	4	96
90	48600	9	0.10588 s	1.15436 s	$1.9002e9$ B	4	102
128	98304	9	0.12941 s	2.98963 s	$3.9888e9$ B	4	106
181	196566	11	0.25293 s	5.99976 s	$8.6168e9$ B	5	106
256	393216	11	0.64557 s	19.0612 s	$1.7669e10$ B	5	106
362	786264	13	1.96164 s	42.4397 s	$3.6981e10$ B	7	106
512	1572864	13	3.69466 s	114.259 s	$7.5168e10$ B	7	106

hierarchy is processed using compressed arithmetic using both HSS and low-rank matrices, which keeps the cost quasilinear. This comes at the cost of potentially larger errors due to increased error accumulation, such that GMRES iterations should also be taken into account. Moreover, the efficiency of the method is determined by the rank growth, which we observe to be moderate. For the last three values of  $h$ , we observe a slight increase in the factorization time. This is likely caused by the high memory requirement, which exceeds the memory of the machine on which the experiments were done. As a consequence, virtual memory is used, which comes at a performance penalty. To put matters into perspective, Table 8.4 lists performance figures for the solver without any compression, which corresponds to a direct solve exploiting the nested dissection structure. We observe better performance with the approximate solver, once matrices are large enough.

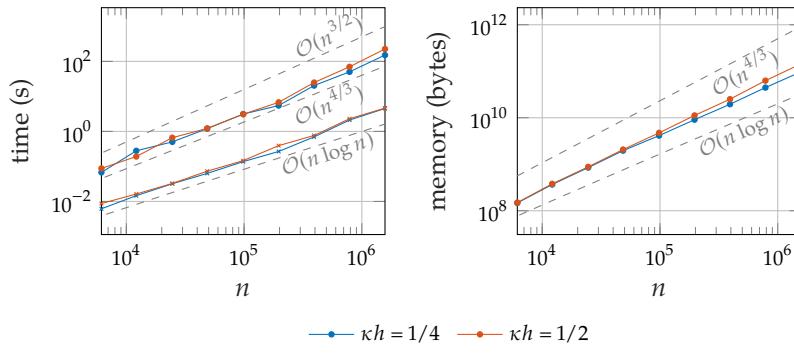
For wave problems it is rarely practical to just increase the wavenumber  $\kappa$ , without also refining the mesh to ensure that the problems are well-resolved. As we have observed in Section 8.3, we expect the growth of the off-diagonal ranks to eventually increase the cost beyond the quasilinear complexity. This would also be in line with what other authors have observed with similar methods [25, 62].

Table 8.5 and Table 8.6 list performance figures for the Helmholtz

**Table 8.3:** Factorization and application times, as well as memory consumption for the Helmholtz problem under  $h$ -refinement in two dimensions.

$1/h$	$n$	application	factorization	memory
22	2904	0.01844 s	0.02356 s	$3.9407e7$ B
32	6144	0.00600 s	0.06504 s	$1.0078e8$ B
45	12150	0.01295 s	0.14313 s	$2.2164e8$ B
64	24576	0.06674 s	0.32718 s	$5.1301e8$ B
90	48600	0.06614 s	0.80438 s	$1.1045e9$ B
128	98304	0.15046 s	1.88804 s	$2.4966e9$ B
181	196566	0.35345 s	4.23004 s	$5.3735e9$ B
256	393216	1.25417 s	18.3061 s	$1.1774e10$ B
362	786264	2.80405 s	52.2744 s	$2.5054e10$ B
512	1572864	7.44162 s	159.893 s	$5.4262e10$ B

**Table 8.4:** Factorization and application times, as well as memory consumption using a direct solver for the Helmholtz problem under  $h$ -refinement in two dimensions.



**Figure 8.21:** Timings and memory requirements of the preconditioner under simultaneous  $\kappa$ - and  $h$ -refinement, keeping  $\kappa h$  constant.

$1/h$	$n$	$L_{\text{HSS}}$	application	factorization	memory	iters	$k_{\max}$
32	6144	5	0.00616 s	0.06750 s	1.4707e8 B	3	80
45	12150	7	0.01464 s	0.27935 s	3.6654e8 B	4	81
64	24576	7	0.03196 s	0.50203 s	8.4778e8 B	3	94
90	48600	9	0.06308 s	1.20064 s	1.9628e9 B	4	105
128	98304	9	0.13711 s	3.12739 s	4.1334e9 B	4	106
181	196566	11	0.26719 s	5.46857 s	9.0829e9 B	5	106
256	393216	11	0.69663 s	20.3759 s	1.9513e10 B	5	146
362	786264	13	2.11069 s	50.1623 s	4.4535e10 B	7	186
512	1572864	13	4.51857 s	151.391 s	9.7229e10 B	6	266

equation under  $h$ -refinement, where we also adapt  $\kappa$  to keep  $\kappa h$  constant. The corresponding data is also shown in Figure 8.21 and illustrates that the cost of factorization is roughly  $\mathcal{O}(n^{4/3})$ . Memory requirements appear to grow as  $\mathcal{O}(n \log n)$ , with  $\kappa h = 1/2$  requiring more memory than  $\kappa h = 1/4$ , as expected.

Another important question concerning performance is how these methods fare for three dimensional problem. To shed some light on this, we revisit the Poisson problem (1.4), this time on  $\Omega = [0, 1]^3$ . Problems in three dimensions are challenging for a number of reasons. The cost of direct methods is increased, based on the lower sparsity in three dimensions and increase memory requirements. For rank-based methods, it is known that the compressibility of Schur complements decreases considerably due to the two-dimensional nature of the separators [25]. We observed this already in Section 7.1, where we illustrated the same admissibility condition results in more complex block-structures in three dimensions. This means that compressed matrices have to be much larger for the HSS compression to be effective. Consequently, we can expect that the size of problems to be considered has to be large enough to truly evaluate the effectiveness of such methods.

We use the adapted compression parameters  $\epsilon_{\text{HSS}} = 10^{-2}$  and  $\beta = 180(p+1)(p+2)(p+3)/6$ . Moreover, we switch to compressed arithmetic only once matrices are four times larger than the HSS blocksize  $\beta$ . The dissection strategy is kept the same as in two dimensions, where the

$1/h$	$n$	$L_{\text{HSS}}$	application	factorization	memory	iters	$k_{\max}$
32	6144	5	0.00870 s	0.08765 s	1.4983e8 B	3	82
45	12150	7	0.01629 s	0.19147 s	3.7940e8 B	3	89
64	24576	7	0.03264 s	0.66125 s	8.7951e8 B	3	104
90	48600	9	0.07302 s	1.21613 s	2.0727e9 B	4	106
128	98304	9	0.14631 s	3.08951 s	4.7327e9 B	4	146
181	196566	11	0.39223 s	6.76212 s	1.1205e10 B	5	186
256	393216	11	0.76545 s	24.8398 s	2.5064e10 B	4	226
362	786264	13	2.29401 s	69.9011 s	6.2828e10 B	6	306
512	1572864	13	4.63122 s	225.629 s	1.4390e11 B	6	426

**Table 8.6:** Factorization and application times, as well as memory consumption for the Helmholtz problem under  $h$ -refinement, while keeping  $\kappa h = 1/2$  constant.

$1/h$	$n$	$L_{\text{HSS}}$	application	factorization	memory	iters	$k_{\max}$
12	41472	1	0.52092 s	23.6715 s	$1.4570e10$ B	10	780
16	98304	2	1.89304 s	107.020 s	$4.5474e10$ B	12	971
20	192000	4	4.41101 s	415.628 s	$1.3949e11$ B	17	1204

**Table 8.7:** Factorization and application times, as well as memory consumption for the Poisson problem in three dimensions. These results were obtained with an IPDG formulation.

separator is drawn in a way which keeps the aspect ratios of the resulting meshes as balanced as possible. Table 8.7 lists our findings for the three-dimensional Poisson problem. As expected, we notice that ranks are much larger, and consequently, memory requirements quickly becomes an issue for our sequential code. The overall cost to form the approximate factorization seems to scale as  $\mathcal{O}(n^2)$ , whereas the cost of applying it seems to scale as  $\mathcal{O}(n^{3/2})$ . This would make the factorization as expensive as classical structured elimination. These findings are hardly conclusive however, due to memory limitations and the small size of the matrices that we consider. Other, similar approaches that combine rank-based approached with the structured multifrontal elimination report runtimes that scale as  $\mathcal{O}(n^{5/3})$  [65]. The overall trend is consistent with similar rank-based approaches, which also report a decrease in the performance when three-dimensional problems are considered [25].

## 8.5 Codes for reproducibility

The experiments that we have shown here have been made available online, to allow easy reproducibility. The functionality of these codes is two-fold. The first part is to generate the problem itself. Thus we require a code to generate the Galerkin matrix  $A$ , together with a suitable right-hand side  $b$  and an elimination tree  $\mathcal{E}$  derived from a nested dissection of the computational domain. This functionality has been implemented in Matlab using the discontinuous Galerkin library NODAL-DG and our custom extension NODAL-DG-EXTENSIONS, which extends the capabilities to formulate continuous Galerkin problems, generate elimination trees using nested dissection and more. The second part is the approximate solver itself, which requires a library for dealing with HSS matrices, as well as the implementation of the solver. We have developed both MATLAB and JULIA implementations for this. Our MATLAB implementation HPRECON uses the hierarchical matrix library HM-TOOLBOX for HSS arithmetic [45]. The JULIA implementation of the preconditioner, HIERARCHICALSOLVERS.JL is reasonably optimized and makes use of the library HssMATRICES.JL, for HSS matrix arithmetic.

To reproduce the experiments, problems can be generated by using the respective MATLAB codes. To generate a Helmholtz problem with  $1/h = 64$ ,  $p = 1$  and a wavenumber of  $\kappa = 32$  on the square domain, we can run:

```
GenMatrixHelmholtz2D("test.mat", 64, 1, 32, "square", 10 )
```

This will generate the Galerkin matrix  $A$ , a suitable right-hand side  $b$  and a nested dissection elimination tree  $\mathcal{E}$  with leaves not bigger than 10 elements.  $A$ ,  $b$  and  $\mathcal{E}$  are then stored in "test.mat". To solve the problem using the approximate solver we can open it in JULIA using the routine read\_problem:

An overview of the aforementioned software and their availability online:  
[github.com/tcew/nodal-dg](https://github.com/tcew/nodal-dg)  
[github.com/bonevbs/nodal-dg-extension](https://github.com/bonevbs/nodal-dg-extension)  
[github.com/bonevbs/HssMatrices.jl](https://github.com/bonevbs/HssMatrices.jl)  
[github.com/bonevbs/HierarchicalSolvers.jl](https://github.com/bonevbs/HierarchicalSolvers.jl)  
[github.com/numpi/hm-toolbox](https://github.com/numpi/hm-toolbox)  
[github.com/bonevbs/hprecon](https://github.com/bonevbs/hprecon)

```
using HierarchicalSolvers
A, b, nd = read_problem("test.mat")
```

Forming the factorization is then as straight-forward as running:

```
nd, nd_loc = symfact!(nd)
F = factor(A, nd, nd_loc, swlevel=-4, atol=1e-6, rtol=1e-6);
```

This constitutes a symbolic factorization step and a second step in which we form the actual, approximate factorization. Here, `swlevel` determines the depth of the hierarchy  $L_{\text{HSS}}$ . A negative value indicates a depth determined relative to the overall depth of the nested dissection hierarchy. To do an approximate solve using the factorization, we can then run:

```
x = F\b
```

Of course we are more interested in applying it as a preconditioner. Using the `IterativeSolvers` package, we can use it as a right preconditioner for GMRES:

```
using IterativeSolvers
x = gmres(A, b; Pr=F, reltol=1e-9, restart=30, maxiter=30)
```

## 8.6 Concluding remarks

We have presented a method for computing an approximate factorization of Galerkin matrices arising from finite element type discretizations of elliptic PDEs. To make this approach efficient, rank-structured matrices, and in particular HSS matrices were used to compress the dense fill-in that occurs during factorization. We were particularly interested in the performance of such techniques when used as a preconditioner.

We were able to verify the quasilinear complexity of the preconditioner for two-dimensional problems using numerical experiments. We have observed that three-dimensional problems still pose significant challenges to these methods due to the increase in ranks which adversely affects the performance of rank-based methods. For two-dimensional problems we were able to investigate the scaling behavior for both  $h$ - and  $p$ -refinement. We saw that ranks grow roughly logarithmically with respect to the number of degrees of freedom when  $h$ -refinement is performed. For  $p$ -refinement this growth is roughly linear. Both of these observations are consistent with theoretical results. Moreover, we have investigated the effect of highly oscillatory problems. While increasing the wavenumber typically leads to an increase in the ranks, we observe that the method is robust in the sense that solutions are still computed accurately and fast. Finally we have tested our method on heterogeneous problems, where it performed equally robustly, without the need of adapting it to the geometry.

As with any work there are still open questions. The main questions here are whether the performance in three dimensions improves if large enough problems are considered. This might require more efficient implementations and, in particular, parallel codes. Another important question regarding three-dimensional problems is whether other rank-structured formats offer performance benefits over HODLR and HSS formats.

**DISCONTINUOUS GALERKIN METHODS FOR  
THE SHALLOW WATER EQUATIONS**

# 9

## Motivation

In the first part of this work we already saw the introduction of the linear wave equation (1.1). In this second part, we are concerned with numerical methods for solving the shallow water equations (SWE). Both the linear wave equation and the shallow water equations belong to the broader class of problems known as conservation laws and hyperbolic partial differential equations [7, 86]. As the name implies, conservation laws are the mathematical expression of phenomena involving conserved quantities and their transport [87–89]. This includes a wide range of real-world phenomena ranging from fluid dynamics to traffic flow, with many applications in engineering and science.

Conservation laws can generally be expressed as initial value problems in the following way:

**Problem 9.0.1** (Initial value problem) For a spatial domain  $x \in \Omega$  and temporal domain  $t \in (0, \infty)$ , find a suitable solution  $q = q(x, t) : \Omega \times (0, \infty) \rightarrow \mathbb{R}^m$  which satisfies

$$\partial_t q + \nabla \cdot F(q) = S(x, q) \quad \text{in } \Omega \times (0, \infty) \quad (9.1a)$$

$$q = q_0 \quad \text{on } \Omega \times \{t = 0\}, \quad (9.1b)$$

comprising  $m$  conservation laws, an initial condition  $q_0 : \Omega \rightarrow \mathbb{R}^m$  and suitable boundary conditions on  $\partial\Omega$ , which will be specified when necessary.  $F(q)$  and  $S(x, q)$  are suitable flux and source functions, and are specified independently for each problem.

The scalar wave equation (1.1) can be expressed in this form by setting  $q = u$ ,  $F(q) = \pm cu$  and  $S = f$ .

The development of methods for solving these PDEs numerically has been a field of intense study. Various numerical methods such as finite difference methods, spectral methods and finite volume methods have been studied extensively with the purpose of efficiently solving these problems [86, 88, 89]. Our focus is on developing discontinuous Galerkin (DG) methods for the shallow water equations. These methods are an interesting option as they combine high-order accuracy with the geometric flexibility that typically comes from lower-order methods [78, 89, 90]. As such, they offer favorable properties to create computational models based on the shallow water equations. Such models are invaluable to study flooding, extreme weather events, and ocean phenomena such as tsunamis. We aim to create a model capable of simulating large-scale tsunami events accurately and efficiently, so that it may be used as an early warning system.

With this in mind, we formulate the shallow water equations on the surface of the sphere and account for the effects of curvature and rotation. Designing a numerical scheme for these equations is a challenging task in itself [91]. Physical accuracy further demands the scheme to be well-balanced, so that it conserves a certain stationary solution where the

9.1 The shallow water equations	99
In one dimension	99
On the sphere	101
9.2 A simple scheme	102
Finite volume scheme	103
The numerical flux	104



**Figure 9.1:** The Great Wave off Kanagawa. Depiction of a Tsunami by Hokusai. Picture taken from [Wikimedia commons](#).

water is at rest [92–94]. In the context of tsunami simulations, this is particularly important, as initial conditions are in general a perturbation of this steady-state solution. The final challenge is wetting/drying, which refers to the appearance of dry areas in the solution and associated numerical problems. This requires the development of methods which are capable of handling wet-dry transitions in a robust manner, which does not compromise our well-balanced scheme.

The solutions to the aforementioned challenges are not necessarily restricted to our treatment of the discontinuous Galerkin method for the spherical shallow water equations. It does however serve as a good model problem as we will see later on. To lay the foundations for this, we introduce the shallow water equations in Section 9.1 and construct a simple finite volume scheme to solve them numerically in Section 9.2.

## 9.1 The shallow water equations

### In one dimension

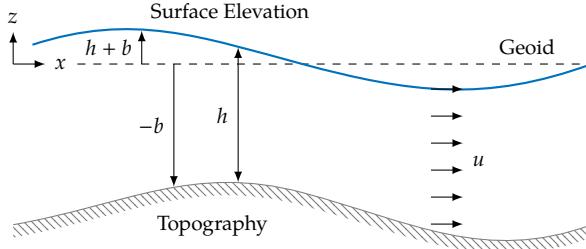
Before we move on to a discussion on methods, let us introduce the shallow water equations. This name can often be misleading as they are frequently used for tsunami modelling, which involves the simulation of ocean waves in open waters at depths of up to 11km. At the same time, they are often used to model weather events, modelling the large-scale dynamics of atmospheric layers as opposed to modelling water. The name “shallow water equations” comes from the assumption that the vertical depth of the fluid is small compared to the horizontal dimensions and the wavelengths being studied. Under these assumptions, one can assume the vertical dynamics to be negligible. This is also referred to as *hydrostatic balance*, as vertical pressure gradients are balanced by gravity. We can then eliminate the vertical velocity component of the Euler equations by integrating them in the vertical direction [95]. In one dimension, the result are the shallow water equations

$$\partial_t h + \partial_x(hu) = 0 \quad (9.2a)$$

$$\partial_t(hu) + \partial_x(hu^2 + \frac{1}{2}gh^2) = -gh\partial_x b \quad (9.2b)$$

for a suitable domain  $\Omega = [x_0, x_1] \subset \mathbb{R}$ . The unknown solution consists of the total column height  $h$ , measured from the bottom of the seabed, and the so-called discharge  $hu$  which is the product of the water column height and the average horizontal velocity of the fluid.  $b$  is the height of the bottom topography measured with respect to an equipotential surface of the gravitation field (geoid). The sea surface level with respect to the geoid is therefore given by  $h + b$ . Finally,  $g$  denotes the vertical acceleration due to gravity. The terminology of the shallow water problem is depicted in Figure 9.2.

It is no surprise that the shallow water equations (9.2a) are functionally equivalent to the isothermal compressible Euler equations. As previously mentioned, we can derive them by depth-integration. In the resulting equations, the water depth acts as a density and the discharge can therefore be understood as the momentum. In this view, the bottom



**Figure 9.2:** Illustration of the assumptions for the shallow water equations in one dimension.

topography acts as pressure field and we can identify the source term to act like the pressure gradient.

To write the shallow water equations as a conservation law, we notice that we could have identically written (9.2a) as

$$\partial_t \begin{bmatrix} h \\ hu \end{bmatrix} + \partial_x \begin{bmatrix} hu \\ hu^2 + \frac{1}{2}gh^2 \end{bmatrix} = \begin{bmatrix} 0 \\ -gh\partial_x b \end{bmatrix}, \quad (9.3)$$

and equivalently (assuming sufficient regularity), we can write

$$\partial_t \begin{bmatrix} h \\ hu \end{bmatrix} + \underbrace{\begin{bmatrix} 0 & 1 \\ -u^2 + gh & 2u \end{bmatrix}}_{=J_F(q)} \partial_x \begin{bmatrix} h \\ hu \end{bmatrix} = \begin{bmatrix} 0 \\ -gh\partial_x b \end{bmatrix}. \quad (9.4)$$

In the latter, we have introduced the Jacobian  $J_F(q)$  of the flux function  $F(q)$  to rewrite the equations in a form that is similar to the linear wave equations. We will see that linearizing the equations indeed yields the eigenvalues of the flux jacobian as wave speeds.<sup>1</sup>

To make the equations dimensionless, we multiply (9.3) by  $g$  and introduce the geopotential water column height  $\varphi = gh$ , as well as the geopotential topography  $\tau = gb$ . We can then write the unknown solution as

$$q = \begin{bmatrix} \varphi \\ \varphi u \end{bmatrix} \quad (9.5)$$

and specify the flux  $F : \mathbb{R}^2 \rightarrow \mathbb{R}^2$  and source  $S : \Omega \times \mathbb{R}^2 \rightarrow \mathbb{R}^2$  as

$$F(q) = \begin{bmatrix} \varphi u \\ \varphi u^2 + \frac{1}{2}\varphi^2 \end{bmatrix} \quad (9.6)$$

and

$$S(x, q) = \begin{bmatrix} 0 \\ -\varphi\partial_x \tau \end{bmatrix}. \quad (9.7)$$

It should be clear that physical solutions only permit positive water heights  $\varphi \geq 0$ . Moreover, the shallow water equations are a system of non-linear equations, which allows for shock-formation [87].<sup>2</sup> As we have assumed that we can describe the layer of water by a function, we expect the shallow water equations to fail in the description of breaking waves as depicted in Figure 9.1.

To develop numerical schemes for solving the shallow water equations, it is useful to study their characteristics. To do so, we look for a curve

1: We loosely speak of wave speeds throughout this chapter. By this we refer to the phase velocity.

2: We omit any discussion regarding appropriate solution spaces and refer the reader to [7, 87].

$x_c(t)$  such that the solution is constant along this curve, i.e.

$$\begin{aligned}\frac{d}{dt} \mathbf{q}(x_c(t), t) &= \mathbf{0} \\ \implies \mathbf{q}(x_c(t), t) &= \mathbf{q}_0(x_c(0)).\end{aligned}$$

This yields

$$\partial_t \mathbf{q}(x_c(t), t) + \partial_t x_c(t) \partial_x \mathbf{q}(x_c(t), t) = \mathbf{0},$$

and by comparing it to (9.4), we notice that for a vanishing source term, this equation is satisfied if

$$J_F(\mathbf{q}(x_c(t), t)) \partial_x \mathbf{q}(x_c(t), t) = \partial_t x_c(t) \partial_x \mathbf{q}(x_c(t), t). \quad (9.8)$$

In other words,  $x_c(t)$  is a characteristic curve, i.e., the solution is constant along  $x_c(t)$ , if  $\partial_t x_c(t)$  is an eigenvalue of the flux Jacobian  $J_F(\mathbf{q})$ . It is easy to show that the eigenvalues of  $J_F(\mathbf{q})$  are given by

$$\alpha_{\pm} = u \pm \sqrt{\varphi}. \quad (9.9)$$

$\alpha_{\pm}$  is also called the wave speed of the system as small perturbations of the solution travel at these speeds across the physical domain. Unlike elliptic PDEs, hyperbolic PDEs are characterized by the finite speeds at which information propagates.

As previously noted, many conservation laws permit the formation and propagation of shock waves, that is, solutions that are not continuously differentiable. Properly defined, such solutions are called *weak solutions* and need to be allowed to make the problem well-posed [7]. Without going into the details, we remark that this is an important property to keep in mind and is often a prime factor in the development of numerical schemes [87].

## On the sphere

We are interested in modelling large-scale atmospheric and oceanic phenomena, which can be described by formulating the shallow water equations on the sphere. We skip the discussion of the two-dimensional shallow water equations and formulate them directly on the two-dimensional sphere  $\mathbf{x} \in S^2(R) \subset \mathbb{R}^3$  of radius  $R$ . To do so, we are presented with the choice of a suitable coordinate system. While the natural choice seems to be spherical coordinates, it is well-known that the coordinate singularities that are introduced at the poles pose a challenge for construction of numerical schemes [91]. An alternative way of using coordinates that conform to the sphere is to use a covariant formulation, directly on local coordinate systems introduced by the discretization [96]. Finally, there is also the possibility of avoiding conforming coordinates altogether and instead use Cartesian coordinates. This requires us to constrain the velocity vectors to the surface of the sphere, which can be done by using a Lagrangian forcing term.

We adopt the latter approach as presented in [91]. This approach comes at the cost of an additional state variable as we have to take three-dimensional velocities  $\mathbf{u} = [u, v, w]^T$ . The state vector is therefore

four-dimensional:

$$\mathbf{q} = \begin{bmatrix} \varphi \\ \varphi u \\ \varphi v \\ \varphi w \end{bmatrix} = \begin{bmatrix} \varphi \\ \varphi \mathbf{u} \end{bmatrix}. \quad (9.10)$$

To write the spherical shallow water equations as a conservation law

$$\partial_t \mathbf{q} + \nabla \cdot \mathbf{F}(\mathbf{q}) = \mathbf{S}(\mathbf{x}, \mathbf{q}), \quad (9.11)$$

we introduce the flux function

$$\begin{aligned} \mathbf{F}(\mathbf{q}) &= f_x(\mathbf{q}) \hat{\mathbf{e}}_x + f_y(\mathbf{q}) \hat{\mathbf{e}}_y + f_z(\mathbf{q}) \hat{\mathbf{e}}_z \\ &= \begin{bmatrix} \varphi u \\ \varphi u^2 + \frac{1}{2}\varphi^2 \\ \varphi uv \\ \varphi uw \end{bmatrix} \hat{\mathbf{e}}_x + \begin{bmatrix} \varphi v \\ \varphi uv \\ \varphi v^2 + \frac{1}{2}\varphi^2 \\ \varphi vw \end{bmatrix} \hat{\mathbf{e}}_y + \begin{bmatrix} \varphi w \\ \varphi uw \\ \varphi vw \\ \varphi w^2 + \frac{1}{2}\varphi^2 \end{bmatrix} \hat{\mathbf{e}}_z, \end{aligned} \quad (9.12)$$

where  $\hat{\mathbf{e}}_x$ ,  $\hat{\mathbf{e}}_y$  and  $\hat{\mathbf{e}}_z$  are the unit vectors along the coordinate axes in  $\mathbb{R}^3$ . The above notation has to be understood in the sense that scalar-products with three-dimensional, Cartesian vectors act on the unit vectors. In particular, this implies

$$\nabla \cdot \mathbf{F}(\mathbf{q}) = \partial_x f_x + \partial_y f_y + \partial_z f_z. \quad (9.13)$$

The source term

$$\mathbf{S} = \begin{bmatrix} 0 \\ \tilde{\mathbf{S}} \end{bmatrix}$$

only acts on the momentum equations via

$$\tilde{\mathbf{S}}(\mathbf{x}, \mathbf{q}) = \mathbf{C}(\mathbf{x}, \mathbf{u}) - \varphi \nabla \tau(\mathbf{x}) + \mu \mathbf{x}. \quad (9.14)$$

The first term is the Coriolis force

$$\mathbf{C}(\mathbf{x}, \mathbf{u}) = -\frac{2\omega z \varphi}{R^2} \mathbf{x} \times \mathbf{u} \quad (9.15)$$

induced by the rotation of the sphere, where  $\omega$  is the angular velocity of the rotation. Here we have chosen the  $z$ -axis to be the axis of rotation for the sphere. The second term in (9.14) accounts for the pressure gradient caused by the slope of the bottom topography  $\tau = g b(\mathbf{x})$ . Finally, to ensure that the fluid does not escape into space, we need to enforce  $\mathbf{u}$  to remain tangential to the surface of the sphere. This is done via the Lagrange multiplier

$$\mu(\mathbf{x}, \mathbf{q}) = \frac{1}{R^2} \mathbf{x} \cdot (\varphi \nabla \tau + \nabla \cdot \tilde{\mathbf{F}}), \quad (9.16)$$

which projects out any non-tangential change in  $\varphi \mathbf{u}$ . Here,  $\tilde{\mathbf{F}}$  denotes the three last components of  $\mathbf{F}$  acting on the momentum equations.

## 9.2 A simple scheme

Let us return to the one-dimensional shallow water equations, with the goal of constructing a simple scheme for solving them. We already

mentioned that one of the main challenges for the numerical solution of conservation laws is the formation of shocks and the resulting lack of regularity in the solution. A natural way of dealing with this problem is by considering an integral formulation of the problem. To do so, we identify a function space  $V_h$  in which we look for numerical approximations to the solution  $\mathbf{q}_h$ . We require the residual of the numerical solution

$$\partial_t \mathbf{q}_h + \nabla \cdot \mathbf{F}(\mathbf{q}_h) - S(\mathbf{x}, \mathbf{q}_h) \quad (9.17)$$

to be orthogonal to  $V_h$ , which yields the formulation

$$\forall v_h \in V_h : \int_{\Omega} (\partial_t \mathbf{q}_h + \nabla \cdot \mathbf{F}(\mathbf{q}_h) - S(\mathbf{x}, \mathbf{q}_h)) v_h \, d\mathbf{x} = 0. \quad (9.18)$$

This formulation is almost the weak formulation of the problem, with the important difference that there is no time-dependence in the test functions  $v_h$ . Instead, we separate the temporal discretization from the discretization in space. This kind of approach is known as *method of lines*.

## Finite volume scheme

The important class of finite volume schemes can be derived by choosing  $V_h$  to be space of piecewise constant functions. To do this, we divide the spatial domain  $\Omega = [x_0, x_1]$  into  $K$  non-overlapping cells

$$D^k = [x_{k-1/2}, x_{k+1/2}], \quad (9.19)$$

such that

$$\Omega \approx \Omega_h = \bigcup_{k=1}^K D^k. \quad (9.20)$$

Here we have introduced the computational domain  $\Omega_h$  which approximates the spatial domain  $\Omega$ . The index  $h$ , which refers to the dependence of the numerical solution on the spatial discretization is defined as

$$h = \sup_k \text{diam } D^k, \quad (9.21)$$

and indicates the accuracy of the approximation. We construct the space of piecewise constant functions as

$$V_h = \left\{ v \in L^2(\Omega) : \forall D^k, v|_{D^k} = \text{const.} \right\}. \quad (9.22)$$

By inserting this into (9.18) and assuming that the source term is  $\mathbf{0}$ , we get

$$\forall k : \partial_t \int_{D^k} \mathbf{q} \, d\mathbf{x} = -[\mathbf{F}(\mathbf{q})]_{x_{k-1/2}}^{x_{k+1/2}} \quad (9.23)$$

after integration by parts. This expresses that mass and momentum are conserved across elements. To obtain the finite volumes scheme, we approximate  $\mathbf{q}$  using cell averages in each cell. The piecewise constant functions

$$\varphi^k(x) = \begin{cases} 1 & x_{k-1/2} \leq x < x_{k+1/2} \\ 0 & \text{otherwise} \end{cases} \quad (9.24)$$

serve as a basis for  $V_h$ . This allows us to write the numerical solution as

$$\mathbf{q}_h(x, t) = \sum_{k=1}^K \hat{\mathbf{q}}^k(t) \varphi^k(x). \quad (9.25)$$

The unknown coefficients  $\hat{\mathbf{q}}^k(t)$  then represent the cell-averages of mass and momentum in each cell. Inserting the numerical solution  $\mathbf{q}_h$  into (9.23) yields

$$\forall k : \partial_t \hat{\mathbf{q}}^k = -\frac{1}{|D^k|} [F(\mathbf{q}_h(x, t))]_{x_{k-1/2}}^{x_{k+1/2}} \quad (9.26)$$

Using a forward Euler discretization in time, we get

$$\forall k : \hat{\mathbf{q}}^{k,n+1} = \hat{\mathbf{q}}^{k,n} - \frac{\Delta t}{|D^k|} [F(\mathbf{q}_h(x, t^n))]_{x_{k-1/2}}^{x_{k+1/2}} \quad (9.27)$$

where  $\hat{\mathbf{q}}^{k,n} = \hat{\mathbf{q}}(t^n)$ .

## The numerical flux

We notice that the evaluation of the flux function in (9.27) is ambiguous due to potentially different value of  $\mathbf{q}_h$  at the interfaces  $x_{k+1/2}, x_{k-1/2}$ . The important observation by Godunov [97] was that solving this problem is akin to solving the *Riemann problem*

$$\partial_t \mathbf{q} + \nabla \cdot F(\mathbf{q}) = \mathbf{S}(x, \mathbf{q}) \quad (9.28a)$$

$$\mathbf{q}(x, t^n) = \begin{cases} \hat{\mathbf{q}}^k(t^n) & x < x_{k+1/2} \\ \hat{\mathbf{q}}^{k+1}(t^n) & x > x_{k+1/2} \end{cases} \quad (9.28b)$$

for  $t \in (t^n, t^{n+1})$  [98]. Schemes that solve this problem analytically at each interface are called Godunov schemes. Unfortunately, doing this is a nonlinear operation and can be quite costly depending on the considered conservation law. For the shallow water equations for instance, the Riemann problem can have three types of elementary waves, which have to be taken into consideration [98].

An alternative approach is to introduce an approximate numerical flux function

$$F(\mathbf{q}_h)|_{x_{k+1/2}} \approx F^*(\hat{\mathbf{q}}^k, \hat{\mathbf{q}}^{k+1}), \quad (9.29)$$

which takes both solutions at the interface into account. In this way, the numerical flux will be communicating the information between cells. We can for instance use the local Lax-Friedrichs flux

$$F^*(\hat{\mathbf{q}}^k, \hat{\mathbf{q}}^{k+1}) = \frac{1}{2} \left( F(\hat{\mathbf{q}}^k) + F(\hat{\mathbf{q}}^{k+1}) \right) - \frac{\alpha}{2} \left( \hat{\mathbf{q}}^{k+1} - \hat{\mathbf{q}}^k \right), \quad (9.30)$$

which uses the maximum local eigenvalue

$$\alpha = \max\{|\hat{u}_k| + \sqrt{\hat{\varphi}_k}, |\hat{u}_{k+1}| + \sqrt{\hat{\varphi}_{k+1}}\} \quad (9.31)$$

of the flux Jacobian. It is worth noting that the numerical flux is consistent with the exact flux function, i.e. the numerical flux becomes exact as the

discontinuity at the interface vanishes:

$$F^*(q, q) = F(q). \quad (9.32)$$

The Lax-Friedrichs flux greatly simplifies the evaluation of the flux at interfaces. The disadvantage of such methods is that they introduce additional dissipation into the scheme [87, 88]. Inserting the numerical flux into (9.27) gives us the Rusanov scheme<sup>3</sup> for the one-dimensional shallow water equations:

$$\hat{q}^k(t^{n+1}) = \hat{q}^k(t^n) - \frac{\Delta t}{|D^k|} \left( F^*(\hat{q}^k, \hat{q}^{k+1}) - F^*(\hat{q}^{k-1}, \hat{q}^k) \right). \quad (9.33)$$

Due to the added dissipation, this scheme is inferior to Godunov schemes. However, it serves as a baseline for the schemes that we are going to construct in latter chapters. Before we move on, we remark that the waves of nearby Riemann problems could interact if the timestep  $\Delta t$  is too large. We avoid this by imposing the Courant-Friedrichs-Levi (CFL) condition

$$\max_k \alpha \frac{\Delta t}{|D^k|} \leq \frac{1}{2}. \quad (9.34)$$

<sup>3</sup>: The difference between the Rusanov and Lax-Friedrichs scheme is the choice of wave speeds. If the wave speeds are chosen locally, based on the solution at both sides, we call the flux a Rusanov or local Lax-Friedrichs flux. If instead the global maximum is considered, the scheme is called the Lax-Friedrichs scheme. The use of the globally maximal wavespeed introduces more dissipation into the scheme.

The main motivation for developing schemes that differ from finite volume schemes is to acquire better higher-order convergence rates. This is possible because solutions are mostly regular apart from some localized areas in which they have discontinuities. This has lead to the development of higher order finite difference and finite volume schemes such as the essentially non-oscillatory (ENO) schemes [87, 89], as well as the discontinuous Galerkin method, among others. The discontinuous Galerkin method offers a number of advantages over other methods, i.e. geometric flexibility and improved parallel efficiency [89, 90, 99].

In this chapter we introduce the discontinuous Galerkin model for solving the spherical shallow water equations (9.11). In order to introduce the discontinuous Galerkin method and techniques related to well-balancing and wetting/drying, we focus on the one-dimensional version of the scheme before moving on to the spherical case.

## 10.1 In one dimension

As for the finite volume discretization of the shallow water equations, we proceed by separately discretizing the scheme in space and in time. To do so, we reuse the discretization of the finite volume method and use  $D^k = [x_{k-1/2}, x_{k+1/2}]$ . Unlike the finite volume solution, which was represented by a piecewise constant solution, we seek higher-order accuracy by representing the numerical solution  $q_h$  as a piecewise polynomial of order  $p$ . Formally we write

$$q_h \in V_h(\Omega) = \{v \in L^2(\Omega) : \forall D^k, v|_{D^k} \in \mathcal{P}_p(D^k)\}, \quad (10.1)$$

with  $V_h(\Omega)$  being the finite element space in which we seek solutions. Then, we can express the numerical solution as the direct sum

$$q(x, t) \approx q_h(x, t) = \bigoplus_{k=1}^K q_h^k(x, t), \quad (10.2)$$

where  $q_h^k(x, t) \in \mathcal{P}_p(D^k)$  is the local polynomial approximation to the solution in each element  $D^k$ . Theoretically, it is possible to choose a different polynomial degree  $p$  in each element. This is referred to  $p$ -adaptivity. For our applications, we stick to a single polynomial degree for all cells.

To represent the numerical solution  $q_h^k(x, t)$ , we require a polynomial basis which spans  $V_h$ . Here, we can choose between modal and nodal representations. The former offer some advantages for the evaluation of the integrals and associated matrices. Nodal approaches on the other hand, offer geometrical flexibility and are usually easier to construct and evaluate.

10.1 In one dimension . . . . .	106
10.2 On the Sphere . . . . .	109
10.3 A few words on meshes . . . .	113
10.4 Time integration . . . . .	115

We choose the latter and construct a basis on the reference element  $I = [-1, 1]$ . We consider the Lagrange basis functions

$$l_i(\xi) = \prod_{j=1, j \neq i}^{p+1} \frac{\xi - \xi_j}{\xi_i - \xi_j}, \quad (10.3)$$

which are defined on a set of interpolation points  $\{\xi_i\}_{i=1}^{p+1} \subset [-1, 1]$ . The nodal basis has the useful property

$$l_i(\xi_j) = \delta_{ij}, \quad (10.4)$$

which allows easy evaluation of the function on the interpolation points. Thus we require a set of points  $\{\xi_j\}_{j=1}^{p+1} \subset I$  with favorable properties for interpolation [78]. A common choice are the Legendre-Gauss-Lobatto (LGL) points. We omit the construction of the LGL points and instead refer the reader to [78]. To construct a basis in the element  $D^k$ , we map the points  $\{\xi_j\}_{j=1}^{p+1}$  via an affine transformation to  $D^k$ . This yields the local points  $\{x_j^k\}_{j=1}^{p+1}$ , from which we can construct a local basis in the sense that

$$L_i^k(x) = l_i(\xi(x)), \quad (10.5)$$

where  $\xi(x)$  is the affine transformation from  $D^k$  into the  $I$ .<sup>1</sup>

Having constructed the local polynomial bases, we can now represent the numerical solution locally as

$$\mathbf{q}_h^k(x) = \sum_{i=1}^{p+1} \hat{q}_i^k(t) L_i^k(x), \quad (10.6)$$

where  $\hat{q}_i^k$  are the unknown coefficients associated with the solution at each point  $x_i$ . In a similar fashion, we express numerical approximations of the flux  $\mathbf{F}_h(\mathbf{q}(x))$  and the bottom topography  $\tau_h(x)$ :

$$\mathbf{F}_h(\mathbf{q}(x, t))|_{D^k} = \mathbf{F}_h^k(x) = \sum_{i=1}^{p+1} \mathbf{F}(\hat{q}_i^k(t)) L_i^k(x), \quad (10.7)$$

$$\tau_h(x)|_{D^k} = \tau_h^k(x) = \sum_{i=1}^{p+1} \tau(x_i^k) L_i^k(x). \quad (10.8)$$

We are now ready to re-consider the variational formulation (9.2a). By restricting the test function space to  $V_h(\Omega)$ , we obtain

$$\begin{aligned} \forall k, i : & \int_{D^k} (\partial_t \mathbf{q}_h - \mathbf{F}(\mathbf{q}_h) \partial_x) L_i^k(x) dx \\ &= - \left[ \mathbf{F}(\mathbf{q}_h) L_i^k(x) \right]_{x_{k-1/2}}^{x_{k+1/2}} + \int_{D^k} \mathbf{S}(x, \mathbf{q}_h) L_i^k(x) dx \end{aligned}$$

within each cell. This brings us back to the problem of evaluating the ambiguous flux  $[\mathbf{F}(\mathbf{q}_h) v(x)]_{x_{k-1/2}}^{x_{k+1/2}}$  at each cell interface. As for the finite volume scheme, we overcome this by inserting a numerical flux function

$$\mathbf{F}(\mathbf{q})|_{x_{k+1/2}} \approx \mathbf{F}^*(\mathbf{q}_h^k(x_{k+1/2}), \mathbf{q}_h^{k+1}(x_{k+1/2})), \quad (10.9)$$

which takes both solutions at the interface into account. To keep things

1: We use the tall letter  $L_i^k(x)$ , as well as the variable  $x$  to distinguish between the local basis function in  $D^k$  and  $l_i(\xi)$  in the reference element.

simple, we stick to the Rusanov flux (9.30). This gives us the so-called weak form of the discontinuous Galerkin formulation

$$\begin{aligned} \forall k, i : & \int_{D^k} (\partial_t \mathbf{q}_h - \mathbf{F}(\mathbf{q}_h) \partial_x) L_i^k(x) dx \\ &= - \left[ \mathbf{F}^*(\mathbf{q}_h^-, \mathbf{q}_h^+) L_i^k(x) \right]_{x_{k-1/2}}^{x_{k+1/2}} + \int_{D^k} S(x, \mathbf{q}_h) L_i^k(x) dx. \end{aligned} \quad (10.10)$$

The short-hand notation  $\mathbf{q}_h^-$  refers to the solution on the left side of the interface, whereas  $\mathbf{q}_h^+$  refers to the solution at the right side of the respective interface. An alternative, albeit equivalent formulation is the strong form, which is obtained by integrating the volume integral by parts:

$$\begin{aligned} \forall k, i : & \int_{D^k} (\partial_t \mathbf{q}_h - \partial_x \mathbf{F}(\mathbf{q}_h)) L_i^k(x) dx \\ &= \left[ (\mathbf{F}(\mathbf{q}_h^-) - \mathbf{F}^*(\mathbf{q}_h^-, \mathbf{q}_h^+)) L_i^k(x) \right]_{x_{k-1/2}}^{x_{k+1/2}} + \int_{D^k} S(x, \mathbf{q}_h) L_i^k(x) dx. \end{aligned} \quad (10.11)$$

Although these formulations are equivalent up to this point, they are not equivalent once further terms are replaced by approximations and integrals are solved by quadrature. This plays a key role in the construction of the well-balanced scheme, which is discussed in Chapter 11.

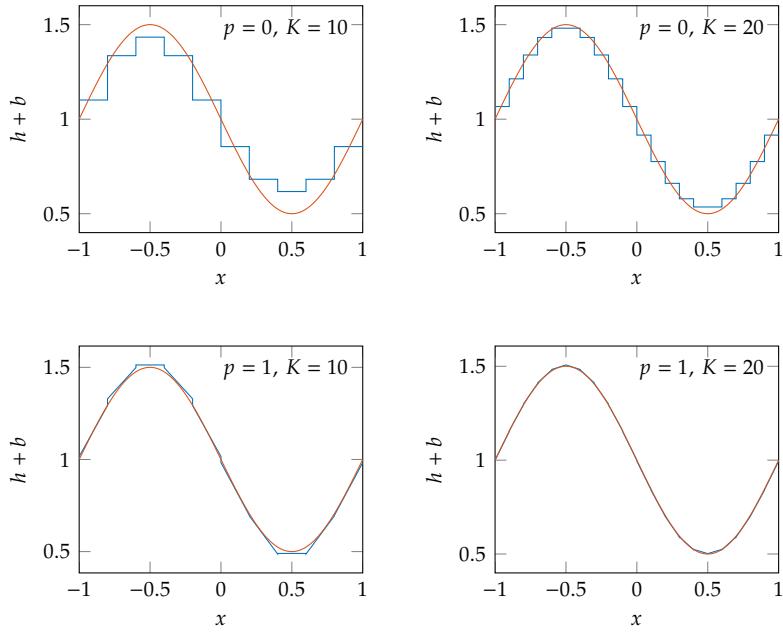
We replace the solution, flux and source terms by their numerical approximations (10.6), (10.7) and (10.8), which are effectively interpolated representations. Replacing them yields

$$\begin{aligned} \forall k, i : & \sum_{j=1}^{p+1} \partial_t \hat{\mathbf{q}}_i^k \int_{D^k} L_i^k(x) L_j^k(x) dx - \sum_{j=1}^{p+1} \mathbf{F}(\hat{\mathbf{q}}_j^k) \int_{D^k} L_j^k(x) \partial_x L_i^k(x) dx \\ &= - \left[ \mathbf{F}^*(\mathbf{q}_h^-, \mathbf{q}_h^+) L_i^k(x) \right]_{x_{k-1/2}}^{x_{k+1/2}} + \sum_{j=1}^{p+1} \hat{\mathbf{q}}_j^k \tau(x_j^k) \hat{\mathbf{e}}_2 \int_{D^k} \partial_x L_j^k(x) L_i^k(x) dx. \end{aligned} \quad (10.12)$$

for the weak form and alternatively, for the strong form

$$\begin{aligned} \forall k, i : & \sum_{j=1}^{p+1} \partial_t \hat{\mathbf{q}}_i^k \int_{D^k} L_i^k(x) L_j^k(x) dx + \sum_{j=1}^{p+1} \mathbf{F}(\hat{\mathbf{q}}_j^k) \int_{D^k} L_j^k(x) \partial_x L_i^k(x) dx \\ &= \left[ (\mathbf{F}(\mathbf{q}_h^-) - \mathbf{F}^*(\mathbf{q}_h^-, \mathbf{q}_h^+)) L_i^k(x) \right]_{x_{k-1/2}}^{x_{k+1/2}} \\ &\quad + \sum_{j=1}^{p+1} \hat{\mathbf{q}}_j^k \tau(x_j^k) \hat{\mathbf{e}}_2 \int_{D^k} \partial_x L_j^k(x) L_i^k(x) dx. \end{aligned} \quad (10.13)$$

At this point, we have obtained semi-discrete formulations which are systems of ordinary differential equations that need to be integrated in time. It is also common to introduce notations for the matrices that arise in this formulation. We forego this step as we are mostly interested in integrating these equations explicitly and a formulation involving matrices is therefore unnecessary for our discussion. Instead, we simply remark that the final form of the schemes can be obtained by using the explicit time-integration techniques which we introduced in Section 10.4.



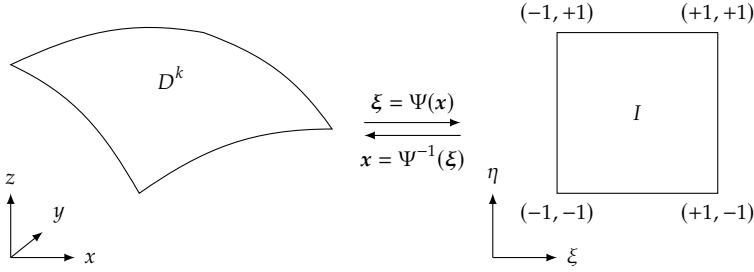
**Figure 10.1:** Comparison of the finite volume scheme to the discontinuous Galerkin scheme for the standing wave solution of the shallow water equations. The red curve is the analytical solution, and the curves in blue represent the numerical solutions.

Finally, the evaluation of the volume integrals requires either the use of analytic integration or numerical quadrature formula [78, 89, 90]. In one dimension, this is straight-forward as we are integrating polynomials on a compact interval of the real axis. To do this numerically, we can use quadrature formulae for the LGL points, which are exact for integrants of polynomial order up to  $2p - 1$  [100]. For our one-dimensional example, this is the case, considering that the bottom topography is represented in the chosen finite element space.

We make a simple comparison of the discontinuous Galerkin scheme ( $p = 1$ ) to the finite volumes scheme ( $p = 0$ ). For a standing wave solution, depicted in Figure 10.1, we compare the results after one oscillation for meshes with  $K = 10$  and  $K = 20$  elements. As expected, we observe that the discontinuous Galerkin scheme yields much more accurate solutions. Moreover, we note that the amplitude is underestimated with the finite volume scheme. This can be explained with the larger jumps at the cell interfaces and the associated increase in dissipation from the numerical flux. This situation is considerably improved with the piecewise linear approximations of the discontinuous Galerkin solution. Another important observation is that we can achieve a higher accuracy with a lower number of degrees of freedom, due to the smoothness of the solution. The cases  $p = 1, K = 10$  and  $p = 0, K = 20$  have the same number of degrees of freedom but the former leads to a better approximation of the analytical solution.

## 10.2 On the Sphere

We consider again the formulation of the discontinuous Galerkin scheme, this time in two-dimensions and for the shallow water equations on the sphere. Ultimately, this will result in the Runge-Kutta discontinuous Galerkin scheme as presented in [91], which will serve as the basis for our model for large-scale geophysical flows [2].



**Figure 10.2:** Transformation into the reference element.

As done above, we express the physical domain  $\Omega$  by a union of smaller elements. To do so, we discretize the sphere into a collection  $\mathcal{M}$  of non-overlapping quadrilateral, curvilinear elements  $D^k \in \mathcal{M}$  such that  $\Omega \approx \Omega_h \cup_{k=1}^K D^k$ . How this can be achieved in practice is explained in Section 10.3. We represent the numerical solution as  $\mathbf{q}_h(\mathbf{x}, t) = \bigoplus_{k=1}^K \mathbf{q}_h^k(\mathbf{x}, t)$ . To construct a basis for each element, we introduce the two-dimensional reference element  $I = [-1, 1]^2$ . Then, for each element, there exists a bijective map  $\Psi : D^k \rightarrow I$  which maps coordinates  $\mathbf{x} \in D^k$  in the physical domain to coordinates  $\xi = [\xi, \eta]^\top = \Psi(\mathbf{x}) \in I$  on the reference element. Figure 10.2 shows an illustration of both elements and the mapping between them. Using this map we can construct a basis on  $I$  and then map it to  $D^k$ . Using the one-dimensional Lagrange polynomials (10.3) defined on the LGL points on  $[-1, 1]$ , we form a tensor product-basis of the form

$$L_m(\xi) = l_i(\xi)l_j(\eta), \quad (10.14)$$

where  $m = 1, 2, \dots, (p+1)^2$  is a unique multiindex associated to each node  $\xi_m = (\xi_i, \eta_j)$  with indices  $i, j = 1, 2, \dots, p+1$ . As we can represent any two polynomials of order up to  $p$  in each variable, we have formed a polynomial basis of mixed order up to  $2p$ . This allows us to represent the numerical solution as

$$\begin{aligned} \mathbf{q}_h^k(\mathbf{x}, t) &= \sum_{m=1}^{(p+1)^2} \hat{\mathbf{q}}_m^k(t) L_m(\mathbf{x}) \\ &= \sum_{i=1}^{p+1} \sum_{j=1}^{p+1} \mathbf{q}_h^k(\mathbf{x}(\xi_i, \eta_j), t) l_i(\xi_i) l_j(\eta_j) \end{aligned} \quad (10.15)$$

using the nodal values  $\hat{\mathbf{q}}_j^k(t) = \mathbf{q}_h^k(\mathbf{x}_j^k, t)$ . With a slight abuse of notation, we will use  $L_i(\mathbf{x})$  to denote  $L_i(\xi(\mathbf{x}))$  and similarly  $\mathbf{x}_i = \mathbf{x}(\xi_i)$ .<sup>2</sup> In this formulation, we do not explicitly define the finite element space  $V_h(\Omega, \mathcal{M})$ , as it is implicitly defined through the construction of the basis functions. Due to the transformation onto the sphere, the functions in  $V_h$  are only piecewise polynomials on the reference elements.

2: For the sake of clarity, we drop the superscript “ $k$ ”. Typically it is clear from the context that we refer to the current element.

With the polynomial representation in place, we consider the problem of satisfying the conservation law (9.1a). We write the weak form of the discontinuous Galerkin formulation

$$\int_D (\partial_t \mathbf{q}_h - \mathbf{F}_h \cdot \nabla - \mathbf{S}_h) L_i \, d\mathbf{x} = - \int_{\partial D} \hat{\mathbf{n}} \cdot \mathbf{F}_h^* L_i \, d\mathbf{x}, \quad (10.16)$$

which should be satisfied by  $\mathbf{q}_h$  in each element  $D \in \mathcal{M}$  and for every basis function  $L_i$ . Alternatively, we can use the strong form of the

Galerkin formulation, as

$$\int_D (\partial_t \mathbf{q}_h + \nabla \cdot \mathbf{F}_h - \mathbf{S}_h) L_i \, dx = \int_{\partial D} \hat{\mathbf{n}} \cdot (\mathbf{F}_h - \mathbf{F}_h^*) L_i \, dx \quad (10.17)$$

to be satisfied by for each element  $D \in \mathcal{M}$  and for each  $L_i$ . As in the one-dimensional case, (10.17) is obtained by integrating (10.16) by parts.

<sup>3</sup> In both equations, we introduced polynomial approximations for the flux

$$\mathbf{F}_h(\mathbf{q}_h) = \sum_{j=1}^{(p+1)^2} \mathbf{F}(\mathbf{q}_h(\mathbf{x}_j)) L_j(\mathbf{x}), \quad (10.18)$$

and source term

$$\mathbf{S}_h(\mathbf{x}, \mathbf{q}_h) = \sum_{j=1}^{(p+1)^2} \mathbf{S}(\mathbf{x}_j, \mathbf{q}_h(\mathbf{x}_j)) L_j(\mathbf{x}). \quad (10.19)$$

As in the one-dimensional case, our solution space  $V_h$  is discontinuous and we require a suitable numerical flux  $\mathbf{F}^*$ , which connects the solutions in the individual elements through a single-valued flux. We do this by evaluating the Rusanov flux

$$\mathbf{F}^*(\mathbf{q}_h^-, \mathbf{q}_h^+) = \frac{1}{2} (\mathbf{F}_h(\mathbf{q}_h^-) + \mathbf{F}_h(\mathbf{q}_h^+)) - \frac{\alpha}{2} (\mathbf{q}_h^+ - \mathbf{q}_h^-), \quad (10.20)$$

on each boundary node, with  $\mathbf{q}_h^-$  representing the local solution within the element and  $\mathbf{q}_h^+$  the solution in the neighboring element. To construct the numerical representation  $\mathbf{F}_h^*(\mathbf{q}_h^-, \mathbf{q}_h^+) \approx \mathbf{F}^*(\mathbf{q}_h^-, \mathbf{q}_h^+)$ , we interpolate these values using the basis function on the boundary nodes. The wavespeed  $\alpha$  represents the maximum local wave speed across the element boundary and it is obtained as

$$\alpha = \max \left\{ |\hat{\mathbf{n}} \cdot \mathbf{u}_h^-| + \sqrt{\varphi_h^-}, |\hat{\mathbf{n}} \cdot \mathbf{u}_h^+| + \sqrt{\varphi_h^+} \right\}. \quad (10.21)$$

We project the numerical flux onto the normals of the element edges  $\hat{\mathbf{n}}$ , effectively evaluating the flux through the edge  $\hat{\mathbf{n}} \cdot \mathbf{F}_h^*$ .

To complete the scheme, we require numerical integration techniques to evaluate the integrals over curved elements in (10.16) and (10.17). To do so, we construct quadrature rules  $\mathcal{Q}_D$  and  $\mathcal{Q}_{\partial D}$  which approximate the volume and surface integrals on  $D$  and  $\partial D$ . We approximate volume integrals of an integrable function  $g \in L^1(D)$  on the curved element with the quadrature formula

$$\begin{aligned} \int_D g(x) \, dx &= \int_I g(\xi) J_D(\xi) \, d\xi \\ &\approx \sum_{i=1}^{p+1} \sum_{j=1}^{p+1} g(\xi_i, \eta_j) J_D(\xi_i, \eta_j) \omega_i \omega_j =: \mathcal{Q}_D [g(x)], \end{aligned} \quad (10.22)$$

where  $J_D(\xi)$  is the determinant of the Jacobian of  $\Psi^{-1}(\xi)$ . Here,  $\omega_i$  and  $\omega_j$  are the quadrature weights associated to the Legendre-Gauss-Lobatto nodes  $\xi_i, \eta_j$  [100]. We remark that the scheme in two dimensions also requires a quadrature formula for the surface integrals. LGL nodes provide quadrature points at the edges of the reference element, which

3: Due to the operators arising in two dimensions, the strong form is also often referred to as the *divergence form*. Similarly, the weak form is referred to as *Green's form*.

simplifies the construction of a quadrature rule for these integrals:

$$\begin{aligned} \int_{\partial D} g(\mathbf{x}) d\mathbf{x} &= \int_{\partial I} g(\boldsymbol{\xi}) J_{\partial D}(\boldsymbol{\xi}) d\boldsymbol{\xi} \\ &\approx \sum_{i=1}^{p+1} g(\xi_i, -1) J_{\partial D}(\xi_i, -1) \omega_i + \sum_{j=1}^{p+1} g(1, \eta_j) J_{\partial D}(1, \eta_j) \omega_j \\ &\quad + \sum_{i=1}^{p+1} g(\xi_i, 1) J_{\partial D}(\xi_i, 1) \omega_i + \sum_{j=1}^{p+1} g(-1, \eta_j) J_{\partial D}(-1, \eta_j) \omega_j \\ &=: \mathcal{Q}_{\partial D} [g(\mathbf{x})]. \end{aligned} \quad (10.23)$$

Here we have introduced  $J_{\partial D}$ , which is the determinant of the Jacobian arising from the mapping of edges in the reference element to the edges of  $D$ . As previously noted, the quadrature rules (10.22) and (10.23) provide exact integration of integrands with polynomial degrees up to  $2p - 1$  [100]. Due to the transformation onto the curved elements, we cannot obtain exact integration due to the non-polynomial nature of the Jacobian determinants.

In what is to follow, we are especially concerned with the practical differences of using the strong form over the weak form. It is therefore useful to discuss the discretization of flux and source terms (10.18), (10.19) in both formulations in some detail. For the weak form, we write out the flux term as

$$\mathbf{F}_h \cdot \nabla L_i = \sum_{j=1}^{(p+1)^2} \left( f_x(\hat{\mathbf{q}}_j) L_j \partial_x L_i + f_y(\hat{\mathbf{q}}_j) L_j \partial_y L_i + f_z(\hat{\mathbf{q}}_j) L_j \partial_z L_i \right) \quad (10.24)$$

and for the strong form as

$$(\nabla \cdot \mathbf{F}_h) L_i = \sum_{j=1}^{(p+1)^2} \left( f_x(\hat{\mathbf{q}}_j) (\partial_x L_j) L_i + f_y(\hat{\mathbf{q}}_j) (\partial_y L_j) L_i + f_z(\hat{\mathbf{q}}_j) (\partial_z L_j) L_i \right). \quad (10.25)$$

The difference between the two lies in the derivative acting on either the test function or on the Lagrange function over which the sum is formed. This will become important later, when we discuss how to achieve the well-balanced property in both formulations.

The source term is discretized by first representing the bottom topography in the finite element space

$$\tau_h(\mathbf{x}) = \sum_{i=1}^{(p+1)^2} \tau(\mathbf{x}_i) L_i(\mathbf{x}) \quad (10.26)$$

by interpolating the topography within each element  $D$ . Using this piecewise polynomial approximation, we construct the numerical approximation of the source term as

$$\tilde{s}_h = \sum_{j=1}^{(p+1)^2} \left( C(\mathbf{x}_j, \mathbf{u}(\mathbf{x}_j)) L_j - \varphi(\mathbf{x}_j) L_j \sum_{k=1}^{(p+1)^2} \tau(\mathbf{x}_k) \nabla L_k(\mathbf{x}_j) \right). \quad (10.27)$$

In this formulation, we have omitted the Lagrange multiplier as we can enforce it directly by projecting the change in velocity onto the sphere

[91].

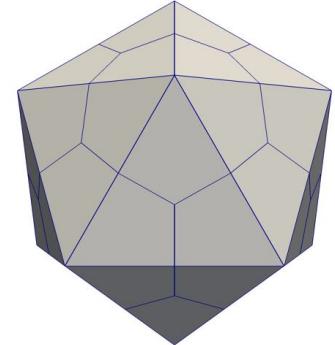
### 10.3 A few words on meshes and adaptivity

So far, we have conveniently ignored how meshes are generated in two dimensions and in particular on the sphere. A popular choice on the sphere are *cubed sphere meshes*, which can be generated by creating a mesh on the sides of the unit cube and then projecting it onto the sphere [101]. Related approaches use icosahedra to generate triangle meshes and hexahedral meshes [102, 103]. In this work we use an icosahedron to generate the mesh. As all its sides are equilateral triangles, we can generate “nice” quadrilateral elements by subdividing them at the barycenter [91]. The elements are then projected onto the sphere and LGL nodes are generated on each curved element. The procedure is illustrated in Figure 10.3. To refine the mesh, we can either subdivide triangular elements further before quadrilaterals are formed or subdivide the quadrilateral elements directly. This allows us to refine the mesh either globally or locally. The latter is particularly useful as we can adjust the mesh to areas of particular interest. We can do this either statically, refining regions of interests from the start, or dynamically to better resolve features of the solution such as shocks.

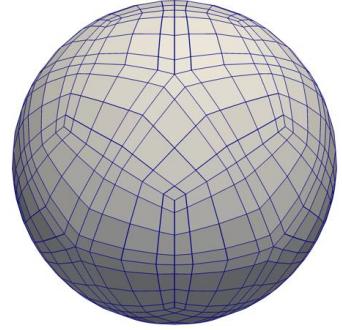
Locally refining the mesh through subdivisions has the drawback that it generally results in non-conforming meshes. This means that the edges of neighboring elements might not be aligned anymore. As a consequence, hanging nodes, i.e. vertices of the elements located on the edge of another element but not coinciding with any other vertex, are created. We simplify our discussion by only considering balanced non-conforming meshes, by which we mean that the difference in refinement is only one across an interface. An example of such a mesh is given in Figure 10.4. This has the consequence that a maximum of three elements can communicate at each interface. Let us formalize this and explain how we can deal with such interfaces at which three elements coincide.

Let  $\mathcal{M}^0$  denote our initial, conforming, non-overlapping partition of  $\Omega$  into quadrilaterals. Subdividing each element into four children generates a series of conforming partitions  $\mathcal{M}^0, \mathcal{M}^1, \mathcal{M}^2, \dots$ . For each of these grids, the superscript refers to the refinement level. We arrange the elements into a quadtree forest  $\mathcal{F}$ , such that each element is connected to its parent from which it was created. At any time, the active mesh  $\mathcal{M}$  is represented as a subforest  $\mathcal{M} \subset \mathcal{F}$ , such that  $\mathcal{M}$  is a balanced, non-overlapping partition of  $\Omega$ . The mesh hierarchy and the active subforest corresponding to the mesh in Figure 10.4 are shown in Figure 10.5.

To express the interaction of three elements over one edge, we need a way of handling the numerical flux and the associated surface integrals over these edges. The principal challenge here is that the high order nodes do not align on these edges. To resolve this, an intermediate solution with matching high order points is computed, which allows the calculation of the numerical flux. We adopt the methods employed in [104, 105], which are also otherwise known as mortar element methods [106].

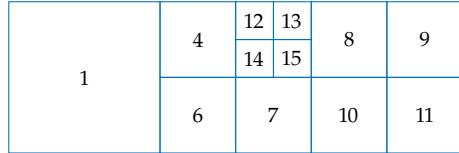


(a) Icosahedron with inscribed quadrilaterals.

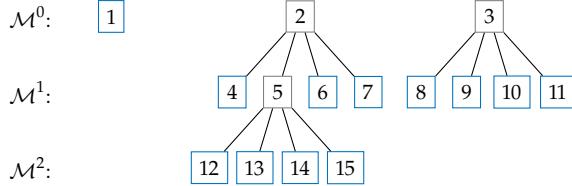


(b) Resulting mesh on the sphere with high-order nodes.

**Figure 10.3:** Generation of icosahedral meshes on the sphere. The top figure depicts the initial icosahedron, where each of the faces is broken up to generate quadrilateral elements. The mesh on the bottom shows the final mesh including the LGL nodes on each element.



**Figure 10.4:** Schematic of a mesh containing various refinement levels. The grid is balanced in the sense that the difference in refinement levels of any two neighboring cells is at most one.



**Figure 10.5:** Illustration of the mesh hierarchy corresponding to Figure 10.4. Elements in blue represent elements that are currently active in the mesh.

Let  $D^0 \in \mathcal{M}^0$  and  $D^1, D^2 \in \mathcal{M}^1$  be elements interfacing at the common edge  $E^0 = E^1 \cup E^2$ , where  $E^1 = \partial D^1 \cap \partial D^0$  and  $E^2 = \partial D^2 \cap \partial D^0$  are the edges of the children elements. The situation is illustrated in Figure 10.6. For this edge, we have to specify the computation of the three fluxes

$$F_h^{*,0}(q_h^0, q_h^1 \oplus q_h^2), F_h^{*,1}(q_h^1, q_h^0), F_h^{*,2}(q_h^2, q_h^0).$$

The superscripts for the fluxes indicate the edge and the corresponding high order nodes on which the numerical flux is represented (See Figure 10.6). Then, we begin by evaluating the fluxes on the high order points of the children elements

$$F_h^{*,1}(q_h^1, q_h^0) = F_h^*(q_h^1, P_0^1 q_h^0), \quad (10.28)$$

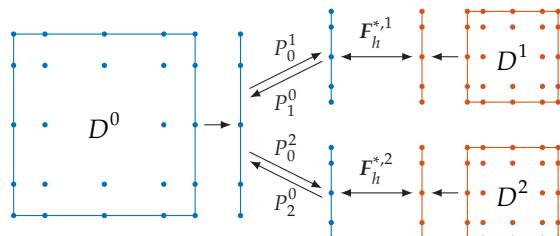
$$F_h^{*,2}(q_h^2, q_h^0) = F_h^*(q_h^2, P_0^2 q_h^0), \quad (10.29)$$

where we have introduced the “scatter” projection operators  $P_0^1, P_0^2$ , which project the polynomial on  $E^0$  onto the nodal bases on  $E^1$  and  $E^2$ , respectively. Because the polynomial bases are of the same degree, these projection operators do not change the polynomial itself. We write out the projection operators to indicate the change of basis. This makes the computation of the numerical flux (10.20) unambiguous, as both polynomials in the argument are defined on the same set of nodes.

Similarly, we define the “gather” projection operators  $P_1^0, P_2^0$ , which project polynomials defined on the nodes of the children edges  $E^1$  and  $E^2$  onto the nodes of the parent edge  $E^0$ . Using these projections, we define the numerical flux on the parent node as

$$F_h^{*,0}(q_h^0, q_h^1 \oplus q_h^2) = \frac{1}{2} \left( P_1^0 F_h^*(P_0^1 q_h^0, q_h^1) + P_2^0 F_h^*(P_0^2 q_h^0, q_h^2) \right). \quad (10.30)$$

As mentioned before, the flux is first evaluated on the nodes of the children elements before it is projected back onto the parent edge. Consequently, it is a piecewise polynomial with  $2(p+1)$  degrees of freedom. The  $L^2$



**Figure 10.6:** Treatment of the hanging node in a non-conforming discretization. The solution on the edge of the parent element  $D^0$  is projected onto the edges of the children elements for the computation of the numerical fluxes. This solution is then projected back to the parent edge.

projection onto the polynomial basis of the parent edge can be broken into two individual  $L^2$  projections  $P_1^0, P_2^0$  from the children elements. As a result, there is some loss of information as the flux is projected onto a basis of dimension  $p + 1$ . In the context of the mortar element method, this approach of performing the flux computation on the edge with the higher approximation order is known as the *maximum rule* [107]. For more information on adaptive mesh refinement in the context of discontinuous Galerkin methods, we refer the reader to [104, 105, 108].

## 10.4 Time integration

By putting everything together and by replacing the integrals with the quadrature rules (10.22) and (10.23), we obtain the semi-discrete scheme in the form of a nonlinear system of ordinary differential equations

$$\partial_t \hat{\mathbf{q}}_h(t) = \mathbf{R}_h(\hat{\mathbf{q}}_h(t)), \quad (10.31)$$

for the vector of unknowns

$$\hat{\mathbf{q}}_h = \begin{bmatrix} \hat{q}_1^1 \\ \vdots \\ \hat{q}_{(p+1)^2}^1 \\ \hat{q}_1^2 \\ \vdots \\ \hat{q}_{(p+1)^2}^K \end{bmatrix} \quad (10.32)$$

and the right-hand side  $\mathbf{R}_h(\hat{\mathbf{q}}_h(t))$  defined by the volume integrals of flux and source terms, as well as surface integrals of flux terms. We have therefore a problem of the form

$$\frac{d}{dt} \mathbf{y} = f(t, \mathbf{y}), \quad (10.33)$$

where  $\mathbf{y}(t)$  is a vector-valued function in time, with the derivative prescribed by  $f(t, \mathbf{y})$ . This problem is typically solved numerically by a multi-stage, multi-step linear method [109]. A popular class among them are the explicit Runge-Kutta methods. These methods update the solution  $\mathbf{y}^n$  at time  $t^n$  to the solution  $\mathbf{y}^{n+1}$  at time  $t^{n+1} = t^n + \Delta t$ , by taking the weighted average

$$\mathbf{y}^{n+1} = \mathbf{y}^n + \Delta t \sum_{i=1}^s b_i \mathbf{k}_i, \quad (10.34)$$

where  $\mathbf{k}_i$  denote the evaluation of  $f$  at different stages, given by

$$\mathbf{k}_1 = f(t^n, \mathbf{y}^n), \quad (10.35a)$$

$$\mathbf{k}_2 = f(t^n + c_2 \Delta t, \mathbf{y}^n + \Delta t(a_{21} \mathbf{k}_1)), \quad (10.35b)$$

$$\vdots \quad (10.35c)$$

$$\mathbf{k}_s = f(t^n + c_s \Delta t, \mathbf{y}^n + \Delta t(a_{s1} \mathbf{k}_1 + a_{s2} \mathbf{k}_2 + \cdots + a_{s,s-1} \mathbf{k}_{s-1})). \quad (10.35d)$$

The coefficients that characterize the Runge-Kutta schemes can therefore be summarized into the so-called Butcher tableau 10.1 [109].

Unsurprisingly, the coefficients  $b_i$  must satisfy

$$\sum_{i=1}^s b_i = 1,$$

for the scheme to be consistent. We are also interested in the scheme converging quickly, such that the truncation error is of order  $\mathcal{O}(\Delta t^{r+1})$ , where  $r$  is called the order. We call the scheme an explicit  $s$ -stage, order- $r$  Runge-Kutta scheme. It is known that for explicit schemes, the order is a lower bound for the number of stages. In other words  $s \geq r$  and  $s \geq r+1$  for  $r > 4$  [109, p. 187].

Apart from high order convergence, we are also interested in the properties of the scheme when errors are introduced. This is generally referred to as the stability of the scheme. The study of numerical schemes for solving the ODE system (10.33) is a well-established field and their stability properties are well-studied [109, 110]. To introduce the concept of stability, we analyze the explicit Euler scheme, given by

$$\mathbf{y}^{n+1} = \mathbf{y}^n + f(t^n, \mathbf{y}^n). \quad (10.36)$$

By linearizing the ODE (10.33), we obtain the system

$$\frac{d}{dt} \mathbf{y} = A \mathbf{y}, \quad (10.37)$$

where  $A$  is the Jacobian of  $f$  with respect to  $\mathbf{y}$ . As we can transform this equation using a similarity transform, we can also instead consider the system in which  $A$  is replaced by its Jordan normal form. As this effectively decouples eigenvectors, we can instead consider the stability for the method applied to  $\frac{d}{dt} \mathbf{y} = \lambda \mathbf{y}$ , where  $\lambda$  is an eigenvalues of  $A$ . If we perform  $n$  steps of the explicit Euler method, we obtain

$$\mathbf{y}_n = (1 + \Delta t \lambda)^n \mathbf{y}_0. \quad (10.38)$$

Simultaneously, we know that the exact solution is  $\exp(n\Delta t \lambda) \mathbf{y}_0$ . Requiring that the numerical approximation stays bounded for solutions which are bounded, yields the linear stability condition

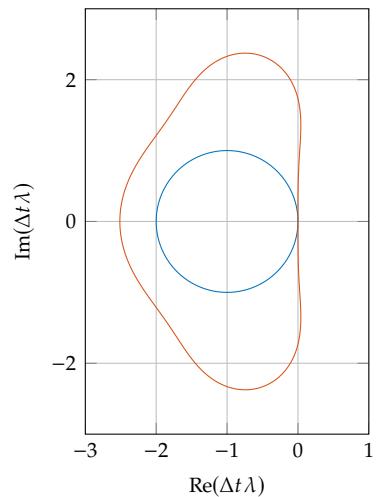
$$|1 + \Delta t \lambda| < 1 \quad (10.39)$$

for eigenvalues  $\lambda \leq 0$ . In other words, these are the scaled eigenvalues  $\Delta t \lambda$ , for which we can expect the numerical scheme give a bounded result if the solution itself is bounded. These regions are also called the regions of absolute stability. Figure 10.7 depicts the stability region of the explicit Euler method (10.39), which is simply the unit circle centered at  $-1$ .

In the context of non-linear partial differential equations and their semi-discrete forms, the linear stability conditions are often insufficient. Especially for hyperbolic problems, where solutions may become discontinuous, we require methods which guarantees stability in some non-oscillatory quantity of the solution, such as the maximum norm or the total variation of the solution [111]. For this reason, a tremendous

**Table 10.1:** Butcher Tableaus for a general, explicit Runge-Kutta scheme.

0				
$c_2$	$a_{21}$			
$\vdots$	$\vdots$	$\ddots$		
$c_s$	$a_{s1}$	$\dots$	$a_{s,s-1}$	
	$b_1$	$\dots$		$b_s$



**Figure 10.7:** Linear stability regions of the explicit Euler method in blue and SSPRK(3,3) method in red.

amount of effort has been undertaken to develop spatial discretizations of the flux function  $\nabla \cdot F(q)$ , which guarantee these stability properties when coupled with the explicit Euler scheme (10.36). However, for practical applications, we require higher order methods and the Runge-Kutta methods (10.34) presented previously, do not retain these stability properties in general.

This is where the development of strong stability preserving Runge-Kutta (SSPRK) methods come in [111]. Given the semi-discrete form (10.33) of our scheme, assume that there exists a maximum time step  $\Delta t_{\max}$ , such that

$$\|\mathbf{y} + \Delta t f(\mathbf{y})\| \leq \|\mathbf{y}\| \quad (10.40)$$

holds for all  $\mathbf{y}$ , some norm  $\|\cdot\|$  and if  $0 \leq \Delta t \leq \Delta t_{\max}$ . Loosely speaking, we say that a scheme is strong stability preserving with coefficient  $c$ , if the solutions satisfy

$$\|\mathbf{y}^{n+1}\| \leq \|\mathbf{y}^n\|, \quad (10.41)$$

whenever (10.40) holds and  $\Delta t \leq c\Delta t_{\max}$  [111]. In particular, we are interested high-order time-discretization methods with this property. Shu and Osher were the first to introduce such methods [112]. The core idea of these methods is to search for high-order Runge-Kutta schemes which can be written as a convex combination of Euler time steps [111, 112]. In latter chapters we will see that such methods benefit us in that they help us guarantee other properties such as preserving positivity of the waterheight.

For our purposes we use the popular three-step, third-order strong stability preserving Runge-Kutta method (SSPRK(3,3)), as presented in [111]. The coefficients which characterize the scheme are listed in Table 10.2. There are other methods, specifically adapted to discontinuous Galerkin discretizations such as [113]. For our discussion however, it will be sufficient to stick to the simple explicit Euler scheme, knowing full well that any relevant time discretization is a convex combination of Euler steps. As such, we consider the fully-discrete form

$$\hat{\mathbf{q}}_h(t + \Delta t) = \hat{\mathbf{q}}_h(t) + \Delta t \mathbf{R}_h(\hat{\mathbf{q}}_h(t)). \quad (10.42)$$

**Table 10.2:** Butcher Tableau for the strong-stability preserving Runge-Kutta 3,3 scheme.

0			
1		1	
1/2	1/4	1/4	
	1/6	1/6	2/3

# Well-balanced schemes

In this chapter we are concerned with the physical consistency of our models. A first step towards this goal is the development of *well-balanced* schemes. In the context of modelling tsunamis or storm surges, one is often confronted with situations in which the initial condition can be regarded as a perturbation of a stationary solution of (9.1a). For water waves and tsunamis, we can imagine throwing a stone into a lake which is perfectly at rest. Mathematically, the lake at rest is characterized by the solution

$$\bar{\varphi} = \max(\varphi_0 - \tau, 0), \quad (11.1a)$$

$$\bar{\varphi}\bar{\mathbf{u}} = \mathbf{0}. \quad (11.1b)$$

We can easily verify that the lake at rest solution, which we denote by  $\bar{\mathbf{q}} = [\bar{\varphi}, \bar{\varphi}\bar{\mathbf{u}}]^\top$ , satisfies

$$\nabla \cdot \mathbf{F}(\bar{\mathbf{q}}) = \mathbf{S}(x, \bar{\mathbf{q}}), \quad (11.2)$$

both for the one-dimensional and the spherical shallow water equations.

<sup>1</sup> To have a physically consistent scheme, it is therefore important to construct schemes that are able to numerically preserve this steady-state [92, 93, 114]. Otherwise we can expect spurious waves to be generated, which pollute the solution that we are interested in. We call schemes that can preserve this steady-state numerically *well-balanced*.

11.1 The well-balanced property	118
11.2 Hydrostatic reconstruction	. 119
11.3 Well-balanced DG schemes	121
11.4 Non-conforming meshes	. . 122

1: As the velocity  $\bar{\varphi}\bar{\mathbf{u}} = \mathbf{0}$  disappears, pressure gradients induced by the bottom topography balance the gradients due to the gravity potential. This condition is equivalent to the condition of hydrostatic balance [93].

## 11.1 The well-balanced property

We formalize the notion of well-balanced schemes. For nodal schemes, we can do this in the following way:

**Definition 11.1.1** Let  $\bar{\mathbf{q}}_h = [\bar{\varphi}_h, \bar{\varphi}\bar{\mathbf{u}}_h]^\top$  denote the numerical representation of the lake at rest solution, such that

$$\bar{\varphi}_h(x_i) = \max\{\varphi_0 - \tau_h(x_i), 0\}, \quad (11.3)$$

$$\bar{\varphi}\bar{\mathbf{u}}_h(x_i) = \mathbf{0}, \quad (11.4)$$

holds on all nodes  $x_i$  in the computational domain. We call a scheme with right-hand side  $\mathbf{R}_h(\mathbf{q}_h)$  well-balanced, if it exactly preserves the lake at rest solution, i.e.

$$\mathbf{R}_h(\bar{\mathbf{q}}_h) = \mathbf{0}. \quad (11.5)$$

Exactness in this context refers to machine-precision, as an exact 0 can hardly be guaranteed with floating-point arithmetic. Therefore, we require  $\mathbf{R}_h(\bar{\mathbf{q}}_h)$  to be of order  $\mathcal{O}(\epsilon\varphi)$ , where  $\epsilon$  is the relative rounding error of floating point arithmetic. To simplify our analysis, we assume that floating point arithmetic is performed exactly. This does not hinder

the accuracy of our analysis as catastrophic cancellation would have to take place to significantly disturb the well-balanced scheme.

**Example 11.1.1** (Finite volume scheme in one dimension) We return to the simple finite volume scheme in one dimension and check whether it is well-balanced. To represent the bottom topography in the finite volume scheme, we represent it as a piecewise constant function:

$$\tau_h(x^k) = \tau(x^k) = \hat{\tau}^k.$$

We can then approximate the gradient of the bottom topography  $\partial_x \tau$  using centered finite differences

$$(\partial_x \tau)_h^k \approx \frac{\hat{\tau}^{k+1} - \hat{\tau}^{k-1}}{2|D^k|},$$

which makes it piecewise constant as well. We borrow the notation from the discontinuous Galerkin schemes and write the  $k$ -th component of the right-hand side as

$$\begin{aligned} R_h^k = & -\frac{1}{|D^k|} \left( \frac{1}{2}(F(\hat{q}^{k+1}) - F(\hat{q}^{k-1})) - \frac{\alpha}{2}(\hat{q}^{k+1} - 2\hat{q}^k + \hat{q}^{k-1}) \right) \\ & - \hat{e}_2 \hat{\varphi}^k (\partial_x \tau)_h^k. \end{aligned}$$

For the scheme to be well-balanced it must satisfy  $R_h^k = \mathbf{0}$ , which yields the discrete momentum balance

$$\frac{1}{2|D^k|} \left( \frac{1}{2}(\hat{\varphi}^{k+1})^2 - \frac{1}{2}(\hat{\varphi}^{k-1})^2 \right) = -\hat{\varphi}^k \frac{\hat{\tau}^{k+1} - \hat{\tau}^{k-1}}{2|D^k|} \quad (11.6)$$

if  $u_h = 0$  is assumed. The lake at rest solution is further characterized by  $\varphi + \tau = \varphi_0 = \text{const}$ . By representing this discretely as  $\hat{\varphi}^k = \varphi_0 - \hat{\tau}^k$ , we can see that the scheme is not well-balanced in general unless  $\tau_h$  is constant everywhere (i.e. no source term).

## 11.2 Hydrostatic reconstruction

The preceding example shows that it is not straight-forward to obtain a well-balanced scheme. For finite volume schemes, we observe that the main problem lies in the different approaches for discretizing fluxes and source terms. This can be seen by considering the one-dimensional hydrostatic balance

$$\partial_x \left( \frac{1}{2} \varphi^2 \right) = -\varphi \partial_x \tau \quad (11.7a)$$

$$\iff \varphi \partial_x (\varphi + \tau) = 0. \quad (11.7b)$$

The discrete momentum balance (11.6) corresponds to the discretization of (11.7a) using central finite differences. This confirms that the problems stem from the difference in discretizations of the source and flux terms, respectively.

A possible way of overcoming this is a technique called *hydrostatic*

*reconstruction*, introduced by Audusse et al. [93]. The core idea is to reconstruct the water surface at the cell interface and compute the numerical fluxes based on the difference in water surface height as opposed to water column height. To understand this, let us imagine that we are at the interface  $x_{k+1/2}$ . We compute the maximum of the bottom topographies

$$\hat{\tau}^{k+1/2} = \max \{ \hat{\tau}^{k+1}, \hat{\tau}^k \}, \quad (11.8)$$

and construct the hydrostatic variables

$$\hat{\mathbf{q}}^{k+1/2-} = \begin{bmatrix} \hat{\varphi}^{k+1/2-} \\ (\hat{\varphi}u)^{k+1/2-} \end{bmatrix} = \begin{bmatrix} \max \{ \hat{\varphi}^k + \hat{\tau}^k - \hat{\tau}^{k+1/2}, 0 \} \\ \hat{\varphi}^{k+1/2-} \hat{u}^k \end{bmatrix} \quad (11.9)$$

and similarly,

$$\hat{\mathbf{q}}^{k-1/2+} = \begin{bmatrix} \hat{\varphi}^{k-1/2+} \\ (\hat{\varphi}u)^{k-1/2+} \end{bmatrix} = \begin{bmatrix} \max \{ \hat{\varphi}^k + \hat{\tau}^k - \hat{\tau}^{k-1/2}, 0 \} \\ \hat{\varphi}^{k-1/2+} \hat{u}^k \end{bmatrix} \quad (11.10)$$

where the “-” and “+” at the end of the superscript distinguish between the left and right sides of the interface. In other words, the hydrostatic variables reconstruct the water column height related to the closest point in the bottom topography, while making sure that they do not become negative. We then replace the numerical flux and source terms with the hydrostatically corrected numerical fluxes

$$\mathbf{F}_l^\bullet(\hat{\mathbf{q}}^k, \hat{\mathbf{q}}_h^{k+1}) = \mathbf{F}^*(\hat{\mathbf{q}}^{k+1/2-}, \hat{\mathbf{q}}^{k+1/2+}) + \frac{1}{2} \left( (\hat{\varphi}^k)^2 - (\hat{\varphi}^{k+1/2-})^2 \right) \hat{e}_2, \quad (11.11a)$$

$$\mathbf{F}_r^\bullet(\hat{\mathbf{q}}^{k-1}, \hat{\mathbf{q}}_h^k) = \mathbf{F}^*(\hat{\mathbf{q}}^{k-1/2-}, \hat{\mathbf{q}}^{k-1/2+}) + \frac{1}{2} \left( (\hat{\varphi}^k)^2 - (\hat{\varphi}^{k-1/2+})^2 \right) \hat{e}_2. \quad (11.11b)$$

Here, the source term has been distributed across the interfaces of the cell. The hydrostatically corrected scheme is then characterized by the right-hand side

$$\mathbf{R}_h^k = -\frac{1}{|D^k|} \left( \mathbf{F}_l^\bullet(\hat{\mathbf{q}}^k, \hat{\mathbf{q}}_h^{k+1}) - \mathbf{F}_r^\bullet(\hat{\mathbf{q}}^{k-1}, \hat{\mathbf{q}}_h^k) \right). \quad (11.12)$$

By inserting in the lake at rest solution  $\bar{\mathbf{q}}_h$ , we verify that the scheme is well-balanced. In particular, the momentum balance yields

$$-\frac{1}{2|D^k|} \left( \frac{1}{2} (\hat{\varphi}^{k+1/2+})^2 - \frac{1}{2} (\hat{\varphi}^{k-1/2-})^2 \right) \quad (11.13)$$

$$-\frac{1}{2} (\hat{\varphi}^{k+1/2-})^2 + \frac{1}{2} (\hat{\varphi}^{k-1/2+})^2 \Big) = 0, \quad (11.14)$$

due to  $\hat{\varphi}^{k+1/2-} = \hat{\varphi}^{k+1/2+}$  and  $\hat{\varphi}^{k-1/2-} = \hat{\varphi}^{k-1/2+}$  for the lake at rest solution. The core idea of the hydrostatic reconstruction is to perform an upwind evaluation of the bottom topography (11.8), to reconstruct the physically meaningful water column heights at the interface. While we have shown that this scheme is well-balanced, we have not done any convergence analysis to prove that it actually solves the shallow water system (9.2a). We refer the reader to [93] for the convergence analysis

and further details on the hydrostatic reconstruction.

### 11.3 Well-balanced discontinuous Galerkin schemes

The core observation for finite volume schemes is that the discretization of the discontinuous bottom topography and resulting source terms requires extra care to guarantee well-balancedness. For the discontinuous Galerkin scheme, we do not consider discontinuities of the bottom topography to be an issue, as we can apply hydrostatic reconstruction. For the moment, we therefore assume that the bottom topography is discretized in a continuous fashion. We consider the weak form. By replacing the integrals with the quadrature rules, the right-hand side for the weak form becomes

$$\begin{aligned} R(\mathbf{q}_h) &= \int_D \mathbf{F}_h \cdot \nabla L_i - S_h L_i \, dx + \int_{\partial D} \hat{\mathbf{n}} \cdot \mathbf{F}_h^* \, dx \\ &\approx \mathcal{Q}_D [\mathbf{F}_h \cdot \nabla L_i - S_h L_i] + \mathcal{Q}_{\partial D} [\hat{\mathbf{n}} \cdot \mathbf{F}_h^* L_i] =: R_h(\mathbf{q}_h). \end{aligned} \quad (11.15)$$

By inserting the numerical representation of the steady-state solution  $\bar{\mathbf{q}}_h$ , we find that the formulation will not be well-balanced in general. Unless both integrands evaluate to 0 exactly, we need to perform integration by parts in order to use the balanced property (11.5) of  $\bar{\mathbf{q}}_h$ . This relies on exact numerical integration, which is a property that the curvilinear discretization does not possess. The strong form

$$\begin{aligned} R(\mathbf{q}_h) &= \int_D (\nabla \cdot \mathbf{F}_h - S_h) L_i \, dx - \int_{\partial D} \hat{\mathbf{n}} \cdot (\mathbf{F}_h - \mathbf{F}_h^*) L_i \, dx \\ &\approx \mathcal{Q}_D [(\nabla \cdot \mathbf{F}_h - S_h)L_i] - \mathcal{Q}_{\partial D} [\hat{\mathbf{n}} \cdot (\mathbf{F}_h - \mathbf{F}_h^*) L_i] =: R_h(\mathbf{q}_h) \end{aligned} \quad (11.16)$$

on the other hand, is obtained through integration by parts. Inserting in the lake at rest solution yields the well-balanced property (11.2) without relying on exact quadrature. The property that divergence-free fields remain constant in the strong form is related to the discrete version of the metric identities. These are a necessary condition for conserving the divergence-free property of a vector field when it is formulated on a curved grid. Under a change of basis, it is always the case, however, discrete representations do not necessarily conserve this property. It can be shown it is automatically satisfied by the strong form of the discontinuous Galerkin method [115].<sup>2</sup>

**Proposition 11.3.1** Let  $\bar{\mathbf{q}}_h$  satisfy the discrete, nodal balance condition  $\nabla \cdot \mathbf{F}_h(\bar{\mathbf{q}}_h) - S_h(\bar{\mathbf{q}}_h) = 0$ . Moreover, let the numerical flux be consistent for this solution, such that  $\mathbf{F}_h^*(\bar{\mathbf{q}}_h, \bar{\mathbf{q}}_h^+) = \mathbf{F}_h(\bar{\mathbf{q}}_h)$ . Then, the strong form of the discontinuous Galerkin scheme  $R_h(\mathbf{q}_h)$  is well-balanced, regardless of the exactness of the quadrature rule.

*Proof.* Inserting  $\bar{\mathbf{q}}_h$  yields  $R_h(\bar{\mathbf{q}}_h) = \mathcal{Q}_D [0] - \mathcal{Q}_{\partial D} [0] = 0$ .  $\square$

The advantages of the strong form lie in the conditions for Proposition 11.3.1. In other words, to obtain a well-balanced scheme, it is sufficient to

2: A problem similar to well-balancing in shallow water equations appears in aeroacoustics, where not conserving the metric identities discretely results in spurious waves on the order of magnitude of the solution [115].

guarantee  $\nabla \cdot F_h(\bar{q}_h) = S_h(\bar{q}_h)$  and  $F_h(\bar{q}_h) = F_h^*(\bar{q}_h^-, \bar{q}_h^+)$  individually. As such, we do not need to pay attention to the volume discretization when we construct a discretization of the surface integrals. This property is particularly useful in the construction of a well-balanced wetting/drying method and non-conforming flux-discretizations, as we will see later on.

**Proposition 11.3.2** *Let  $\mathcal{M}$  be a conforming mesh of  $\Omega$  and let the approximation of the bathymetry  $\tau_h \in V_h(\Omega, \mathcal{M})$  be continuous on  $\Omega$ . Moreover let  $\varphi_0 > \tau_h$ , i.e. there are no dry areas in  $\Omega$ . The strong form of the scheme, as presented in Section 10.2 is well-balanced under these assumptions.*

*Proof.* We start by showing  $F_h^*(\bar{q}_h^-, \bar{q}_h^+) = F_h(\bar{q}_h)$ . We have  $\mathbf{u}_h = \mathbf{0}$  and  $\varphi_h = \varphi_0 - \tau_h$ , which is continuous due to the continuous bathymetry  $\tau_h$ . This implies

$$F_h^*(\bar{q}_h^-, \bar{q}_h^+) = F_h^*(\bar{q}_h, \bar{q}_h) = F_h(\bar{q}_h)$$

due to the consistency of the Lax-Friedrichs flux.

In addition, we have to establish  $\nabla \cdot F_h(\bar{q}_h) = S_h(\bar{q}_h)$ . We can ignore the Lagrange multiplier as it only projects the change in velocity to the surface of the sphere. Inserting  $\mathbf{u}_h = 0$  then reduces the equations to

$$\varphi_h \nabla \varphi_h = -\varphi_h \nabla \tau_h,$$

which is trivially satisfied by  $\bar{q}_h$ . □

## 11.4 Well-balanced non-conforming meshes

We have shown that the strong form of the discontinuous Galerkin scheme has inherent advantages over the weak form. If integration is exact however, this is irrelevant. Consequently, many well-balanced discontinuous Galerkin schemes in the literature use the weak form [116–120]. However, the advantages of the strong form go beyond this as we will see in the following chapters. One particular advantage lies in the fact that surface and volume integrals do not have to balance each other. Exactly this makes handling non-conforming meshes simpler and we only require a few changes to the method presented in Section 10.3. In particular, the introduction of non-conforming edges leaves volume terms unchanged and therefore, we only have to ensure that  $F_h(\bar{q}_h) = F_h^*(\bar{q}_h^-, \bar{q}_h^+)$ .

Let us recall the flux computation across non-conforming edges as illustrated in Figure 10.6. The computation involves projections onto the children elements, and back onto the parent edge, as well as the evaluation of the flux itself. The projection of the polynomial on the parent edge  $E^0$  to the edges of the children elements  $E^1$  and  $E^2$  is exact as we are projecting from one polynomial basis of  $p$ -th order to another. Unless there are dry areas, this preserves the lake at rest condition  $\varphi_h + \tau_h = \varphi_0$ . Once dry nodes are introduced, we cannot guarantee that the water surface  $\varphi_h + \tau_h$  remains constant as we cannot allow negative nodal values.<sup>3</sup> This implies that the nodal lake at rest property (11.3) is not sustained by the projected solutions  $P_0^1 \bar{q}_h^0$  and  $P_0^2 \bar{q}_h^0$ . This is due to the non-constant water surface and the choice of new nodes to represent

3: Dry nodes refer to nodes where the waterheight  $\varphi$  is below a certain threshold  $\varphi_0$ . We formalize this in Chapter 12, which is concerned with the treatment of such situations. Figure 12.1 illustrates that the numerical approximation of the water surface cannot remain constant in the presence of dry areas unless negative values are permitted.

$\varphi_h$  and  $\tau_h$ . To circumvent this issue, we choose to reconstruct the water surface using

$$\sigma_h(x_i) = \begin{cases} \varphi_h(x_i) + \tau_h(x_i) & \varphi_h(x_i) > \varphi_{\text{tol}} \\ \varphi_0 & \text{otherwise,} \end{cases} \quad (11.17)$$

where  $\varphi_{\text{tol}}$  is a minimum water height, below which the solution is considered dry. The projection is then performed on the reconstructed variable  $\sigma_h$ :

$$P_0^{1,*}\varphi_h^0(x_i^1) := \max \{P_0^1\sigma_h^0(x_i^1) - P_0^1\tau_h^0(x_i^1), 0\}, \quad (11.18)$$

$$P_0^{1,*}(\varphi u)_h^0(x_i^1) := P_0^1(\varphi u)_h^0(x_i^1). \quad (11.19)$$

This ensures that the projection carries over the nodal lake at rest property to the children nodes  $x_i^1$ .

Next, we need to ensure that the analytical and numerical fluxes evaluate to the same value on the children nodes. The challenge here is that the bathymetry generally is discontinuous due to the non-conforming mesh. This implies that the lake at rest solution  $q_h$  itself is also discontinuous due to  $\varphi_h = \varphi_0 - \tau_h$ . Because the numerical flux is single-valued it cannot match the analytic fluxes on both sides simultaneously. We have encountered this situation previously in the construction of well-balanced finite volumes schemes. Again, we use the hydrostatic reconstruction as introduced in [93]. This technique has been applied by [116] to form well-balanced discontinuous Galerkin schemes with discontinuous bottom topographies [116, 121]. We form the hydrostatic reconstruction of  $q_h^\pm$  at the cell interfaces:

$$q_h^{\bullet,\pm} = \begin{bmatrix} \varphi_h^{\bullet,\pm} \\ (\varphi u)_h^{\bullet,\pm} \end{bmatrix} = \begin{bmatrix} \max \{\varphi_h^\pm + \tau_h^\pm - \tau_h^\bullet, 0\} \\ \varphi_h^{\bullet,\pm} u_h^\pm \end{bmatrix}, \quad (11.20)$$

where  $\tau_h^\bullet$  is the reconstructed cell interface height

$$\tau_h^\bullet = \max \{\tau_h^-, \tau_h^+\}. \quad (11.21)$$

We replace the numerical flux (10.20) with the hydrostatically reconstructed flux

$$F_h^\bullet(q_h^-, q_h^+) := F_h^*(q_h^{\bullet,-}, q_h^{\bullet,+}) + \frac{1}{2}((\varphi^-)^2 - (\varphi^{\bullet,-})^2) \begin{bmatrix} 0 \\ \hat{e}_x \\ \hat{e}_y \\ \hat{e}_z \end{bmatrix}, \quad (11.22)$$

with source term contributions in the components related to the momentum balance. This flux is not single-valued, as is necessary for it to be well-balanced. If we insert the lake at rest solution, we recover the analytical fluxes  $F(q^\pm)$  on both sides of the interface.

The last projection step in (10.30) also requires special treatment, as it is inexact. Again, we make use of the strong form and evaluate the difference between numerical and exact fluxes directly on the children nodes. As such, we perform gather projections on the difference of the

numerical and analytical fluxes:

$$\frac{1}{2}P_1^0 \left( F_h^\bullet \left( P_0^1 \mathbf{q}_h^0, \mathbf{q}_h^1 \right) - F_h \left( P_0^1 \mathbf{q}_h^0 \right) \right) + \frac{1}{2}P_2^0 \left( F_h^\bullet \left( P_0^2 \mathbf{q}_h^0, \mathbf{q}_h^2 \right) - F_h \left( P_0^2 \mathbf{q}_h^0 \right) \right). \quad (11.23)$$

For the lake at rest solution, this difference becomes  $\mathbf{0}$  and as such, the projected flux is also  $\mathbf{0}$ , regardless of the accuracy of the projection.

**Proposition 11.4.1** *Let  $\mathcal{M}$  be a balanced, non-conforming mesh in  $\Omega$  and let  $\bar{\varphi}_h$  be a lake at rest solution in the sense of (11.3). Then the modifications (11.17)-(11.23) yield a well-balanced evaluation of flux terms at the non-conforming interfaces.*

*Proof.* Because the non-conformity of the discretization does not affect the evaluation of the volume terms, we only have to prove the well-balanced property of the fluxes.

We have already established that the lake at rest property carries over to the nodes of the children elements, when the first projection step is applied on the reconstructed water surface height. As such, the evaluation of  $F_h^\bullet(\bar{\mathbf{q}}_h, \bar{\mathbf{q}}_h^+)$  gives the same result as  $F_h(\bar{\mathbf{q}}_h)$ . Hydrostatic reconstruction gives us  $\varphi_h^{\bullet,-} = \varphi_h^{\bullet,+}$  due to  $\bar{\varphi}_h^\pm + \tau_h^\pm = \varphi_0$ . Consequently, evaluation of (11.22) yields

$$F_h^\bullet(\bar{\mathbf{q}}_h, \bar{\mathbf{q}}_h^+) = F_h^*(\mathbf{q}_h^{\bullet,-}, \mathbf{q}_h^{\bullet,+}) + \frac{1}{2} \left( (\varphi^+)^2 - (\varphi^{\bullet,+})^2 \right) \begin{bmatrix} 0 \\ \hat{\mathbf{e}}_x \\ \hat{\mathbf{e}}_y \\ \hat{\mathbf{e}}_z \end{bmatrix} = F_h(\bar{\mathbf{q}}_h),$$

where we have utilized the identity of the flux (9.12). For the final step we perform the gather projection steps on the flux differences  $F_h^\bullet(\bar{\mathbf{q}}_h, \bar{\mathbf{q}}_h^+) - F_h(\bar{\mathbf{q}}_h)$  which are  $\mathbf{0}$ , as just demonstrated.  $\square$

We remark that adaptive mesh refinement requires refinement and coarsening operations, which compute the representation of the solution on the updated mesh. For the method to be well-balanced, we require these operations to conserve the lake at rest property of  $\bar{\mathbf{q}}_h$  as well. As these are usually projection operators, we can adapt the methods presented here. This implies that projections are only applied to the reconstructed water surface variable  $\sigma_h$  before converting it back.

Numerical models for tsunami simulations, storm surge prediction and inundation modelling all require some way of handling wet-dry transitions. Loosely speaking, this refers to areas of the physical domain in which the water column height  $\varphi$  goes from 0 to  $\varphi > 0$  or vice-versa. While this is intuitive to imagine, it poses significant challenges to numerical schemes. Mathematically speaking, we should consider a time-dependant domain  $\Omega(t)$ , which encompasses all the areas which have a non-zero water column height. Doing this numerically would require constant remeshing, robust handling of topological changes of  $\Omega(t)$ , as well as the construction of rules and algorithms for evolving the boundaries of  $\Omega(t)$ . Consequently, most practical models choose to avoid this by formulating heuristics for dealing with the wet-dry interface. These result in a number of problems:

**Negative waterheights** Once the water column heights become negative, the character of the shallow water equations changes dramatically. The wavespeeds become complex, which is an indication for the system becoming ill-posed. We can therefore expect unphysical results and the scheme to become unstable if negative values are introduced.

**Dealing with small  $\varphi$**  To evaluate the flux, we need to compute  $(\varphi u)^2/\varphi$ , which can lead to loss of accuracy if  $\varphi$  is close to 0.

**Partly dry cells** In the context of finite volume methods, wetting/drying simply means that there will be some cells which can be considered “dead”. With discontinuous Galerkin methods, this is not the case anymore and we need a robust way of handling these elements.

**Well-balanced property** When dealing with these issues, we have to preserve the lake-at rest solution.

**Stability** Finally, the methods should result in a stable scheme, which does not amplify errors indefinitely.

12.1 A survey of existing methods	125
12.2 Maintaining positivity . . . . .	126
12.3 Flux discretization . . . . .	129
12.4 A few notes on stability . . . . .	130

## 12.1 A survey of existing methods

We give a short survey of existing methods for dealing with these challenges. Bokhove uses the straight-forward approach of adapting the mesh to the moving shorelines [122]. The advantage of this method is physical accuracy as the shallow water equations are only solved where it is appropriate. As previously noted, the drawback is that constant remeshing is required and while this is straight-forward for one-dimensional problems as presented, it is considerably more difficult in two dimensions. A large portion of algorithms proposed in the literature therefore treat the shoreline as an immersed boundary within the elements. Various methods exist to ensure positivity of the approximate solution [116, 119, 123]. For discontinuous Galerkin methods Xing et al. propose a method for maintaining the positivity of cell-averages by using a restriction on the timesteps [116]. Positivity on the nodes is then ensured by using

a positivity-preserving limiter, which rescales the polynomial around the positive average. However these methods [116, 121] are not unconditionally well-balanced as partly dry cells are neglected. The problem with these cells is that they introduce artificial gradients and generate unphysical waves at the wet-dry interface. This effect has also been observed by others, and is sometimes referred to as numerical storms [124]. To overcome this Kesserwani et al. propose a reconstruction of nodal values such that the pressure gradients vanish for the lake at rest solution [120]. The authors present this method for a piecewise linear method in one dimension and it is unclear how this approach performs for higher-order methods. Other approaches cancel gravity in these cells to eliminate the problem of artificial pressure gradients [118, 125]. This in turn requires the introduction of dual-valued fluxes to make the scheme well-balanced. While these methods make the schemes well-balanced, they are not consistent with the physical model and appear to be restricted to piecewise linear polynomials. Other approaches use artificial porosity and introduce a fraction indicator to represent how much of the cell is wet and how much is dry [126, 127]. This allows for implicit time integration with large timesteps but introduces other problems such as higher wave speeds in the wet-dry region and a modified shallow water model. Finally, there is the issue of stability at the wet-dry interface. Most of the aforementioned algorithms reduce the order of the solution to linear polynomials and apply a slope limiter to prevent unphysical discharges [116, 125]. Meister and Ortobue use an implicit scheme with a modal filter and a shock indicator to stabilize the scheme in the nearly dry regions [119].

The method that we propose uses the approach presented in [116] to maintain positivity and combines this with a modified discretization for partly dry cells [2]. The advantage is that it does not modify the physics and it is valid for any polynomial degree.

## 12.2 Maintaining positivity

Before we continue, we require a distinction between wet and dry areas. We call a node  $x_i$  dry, if the water height  $\varphi_h(x_i)$  is smaller than a certain tolerance  $\varphi_{\text{tol}}$ . If the opposite is true we call it a wet node. In the context of discontinuous Galerkin methods, three situations can occur. If an element  $D$  contains only dry nodes, we call it a *dry* element. If it contains only wet nodes we call it a *wet* element. In the case that the element contains both wet and dry nodes, we call the element *semi-dry* or *partly dry*. As previously noted, these elements require special attention.

We adopt the approach presented in [116] for maintaining the positivity of the solution. This method uses timestep restrictions, which is a well-established approach in the context of finite volume schemes to guarantees positivity of the cell average  $\varphi_h^{\text{avg}}$ . In the following, we adapt it to the spherical shallow-water equations.

**Proposition 12.2.1** Let  $q_h(x, t_n)$  denote the numerical solution at time  $t_n$  with positive water height on all nodes  $x_i$ . Assuming exact integration of the

integrals in (10.17), the cell-averaged water column height

$$\varphi_h^{\text{avg}} = \int_D \varphi_h(\mathbf{x}) \, d\mathbf{x} \quad (12.1)$$

remains positive after one Euler timestep (10.42), provided the timestep meets the CFL-like requirement

$$\frac{J_{\partial D}}{J_D} \alpha \Delta t \leq \frac{\omega_1}{2} \quad (12.2)$$

everywhere.

*Proof.* Under the assumption of exact numerical integration, we recover the evolution of cell averages by inserting  $L_i = 1$  and the Euler timestep discretization in (10.17):

$$\int_D \mathbf{q}_h(\mathbf{x}, t_{n+1}) \, d\mathbf{x} = \int_D \mathbf{q}_h(\mathbf{x}, t_n) \, d\mathbf{x} + \Delta t \left( \int_D \mathbf{s}_h \, d\mathbf{x} - \int_{\partial D} \hat{\mathbf{n}} \cdot \mathbf{F}_h^* \, d\mathbf{x} \right).$$

As we assume exact numerical integration, the strong form and the weak form are equivalent. We choose the weak form, which reduces the evolution of cell-averages to the boundary of the domain  $D$ . We introduce  $\mathbf{F}_\varphi^*$ , which denotes the component of the numerical flux acting on the water height. Then, by replacing the integrals with the quadrature rules, we obtain

$$\begin{aligned} \mathcal{Q}_D[\varphi_h(\mathbf{x}, t_{n+1})] &= \mathcal{Q}_D[\varphi_h(\mathbf{x}, t_n)] \\ &\quad - \Delta t \mathcal{Q}_{\partial D}[\hat{\mathbf{n}} \cdot \mathbf{F}_\varphi^*(\mathbf{q}_h^-(\mathbf{x}, t_n), \mathbf{q}_h^+(\mathbf{x}, t_n))] \end{aligned}$$

for the cell-averaged water height. By splitting the quadrature into sums over the edges and the sum over the interior points, we have

$$\begin{aligned} &\mathcal{Q}_D[\varphi_h(\mathbf{x}, t_{n+1})] \\ &= \sum_{i,j=2}^p \varphi_h(\xi_i, \eta_j, t_n) J_D(\xi_i, \eta_j) \omega_i \omega_j \\ &\quad + \sum_{i=1}^{p+1} \varphi_h(\xi_i, -1, t_n) J_D(\xi_i, -1) \gamma_i \omega_i \omega_1 \\ &\quad - \Delta t \hat{\mathbf{n}} \cdot \mathbf{F}_\varphi^*(\mathbf{q}_h^-(\xi_i, -1, t_n), \mathbf{q}_h^+(\xi_i, -1, t_n)) J_{\partial D}(\xi_i, -1) \omega_i \\ &\quad + \sum_{i=1}^{p+1} \varphi_h(\xi_i, 1, t_n) J_D(\xi_i, 1) \gamma_i \omega_i \omega_h \\ &\quad - \Delta t \hat{\mathbf{n}} \cdot \mathbf{F}_\varphi^*(\mathbf{q}_h^-(\xi_i, 1, t_n), \mathbf{q}_h^+(\xi_i, 1, t_n)) J_{\partial D}(\xi_i, 1) \omega_i \\ &\quad + \sum_{j=1}^{p+1} \varphi_h(-1, \eta_j, t_n) J_D(-1, \eta_j) \gamma_j \omega_1 \omega_j \\ &\quad - \Delta t \hat{\mathbf{n}} \cdot \mathbf{F}_\varphi^*(\mathbf{q}_h^(-1, \eta_j, t_n), \mathbf{q}_h^+(1, \eta_j, t_n)) J_{\partial D}(-1, \eta_j) \omega_j \\ &\quad + \sum_{j=1}^{p+1} \varphi_h(1, \eta_j, t_n) J_D(1, \eta_j) \gamma_j \omega_h \omega_j \\ &\quad - \Delta t \hat{\mathbf{n}} \cdot \mathbf{F}_\varphi^*(\mathbf{q}_h^-(1, \eta_j, t_n), \mathbf{q}_h^+(1, \eta_j, t_n)) J_{\partial D}(1, \eta_j) \omega_j \end{aligned}$$

where

$$\gamma_i = \begin{cases} \frac{1}{2} & \text{if } i = 1 \text{ or } i = p + 1, \\ 1 & \text{otherwise.} \end{cases}$$

As we assume the solution at  $t_n$  to have a positive water height everywhere, the first term remains positive. Thus, it is sufficient to show that the four boundary sums stay positive. By inserting the Lax-Friedrichs flux (10.20), we rewrite the boundary terms as

$$\varphi_h^- \left[ J_D \gamma \omega_1 - \frac{1}{2} \Delta t J_{\partial D} (\hat{\mathbf{n}} \cdot \mathbf{u}_h^- + \alpha) \right] + \varphi_h^+ \Delta t J_{\partial D} \left[ \frac{\alpha}{2} - \frac{1}{2} \hat{\mathbf{n}} \cdot \mathbf{u}_h^+ \right],$$

where we have dropped the indices for simplicity. Due to the definition of  $\alpha$  (10.21), we have  $|\hat{\mathbf{n}} \cdot \mathbf{u}_h^\pm| < \alpha$ . It follows that the second term is always positive and that the first term is positive under the condition

$$J_D \gamma \omega_1 \geq J_{\partial D} \alpha \Delta t,$$

which is the sufficient condition to guarantee positive averages  $\varphi_h^{\text{avg}}$ .  $\square$

Even though we have assumed exact numerical integration, this condition is sufficient to ensure positivity of cell averages in practice. Apart from their impacts on well-balancedness, the difference between the weak and strong form are negligible and this condition gives us an idea of how big the timesteps can be while maintaining positive cell-averages. Positive cell averages are also retained with higher-order time integration methods if we utilize convex combinations of Euler timesteps. This is one of the principle motivators behind Strong stability preserving Runge-Kutta methods as presented in Section 10.4. Using these time integrators will therefore preserve the positivity of the cell-averaged water heights [111, 116].

In the context of a discontinuous Galerkin discretization, we need to ensure that each nodal value remains positive in addition to the cell average. To achieve this, a limiter is applied in a post-processing step, which rescales the solution around the positive cell-averages [116].

As we have established the positivity of the cell-averages, we correct the nodal values using the positivity limiter presented in [116]. After each timestep, we rescale the solution in each cell according to

$$\varphi_h^* = \theta \left( \varphi_h - \varphi_h^{\text{avg}} \right) + \varphi_h^{\text{avg}}, \quad (12.3a)$$

$$\mathbf{u}_h^* = \theta \left( \mathbf{u}_h - \mathbf{u}_h^{\text{avg}} \right) + \mathbf{u}_h^{\text{avg}}, \quad (12.3b)$$

where

$$\theta = \min \left\{ 1, \frac{\varphi_h^{\text{avg}}}{\varphi_h^{\text{avg}} - \varphi_h^{\text{min}}} \right\}, \quad (12.4)$$

$$\varphi_h^{\text{min}} = \min_{x_i \in D} \{ \varphi_h(x_i) \}. \quad (12.5)$$

The cell-averages  $\varphi_h^{\text{avg}}$  and  $\mathbf{u}_h^{\text{avg}}$  can be computed using (10.22). We observe that the positivity limiter (12.3a) rescales the solution around the average water height. Finally, as we consider any node with water height below the threshold  $\varphi_{\text{tol}}$  as a dry node, we need to make sure that

the velocity remains  $\mathbf{0}$  at these nodes. For this reason, we set  $\varphi \mathbf{u} = \mathbf{0}$  at all nodes with water heights  $\varphi \leq \varphi_{\text{tol}}$ . This process conserves mass, but it does not conserve momentum. However it is necessary to maintain stability and the physical plausibility of the solution.

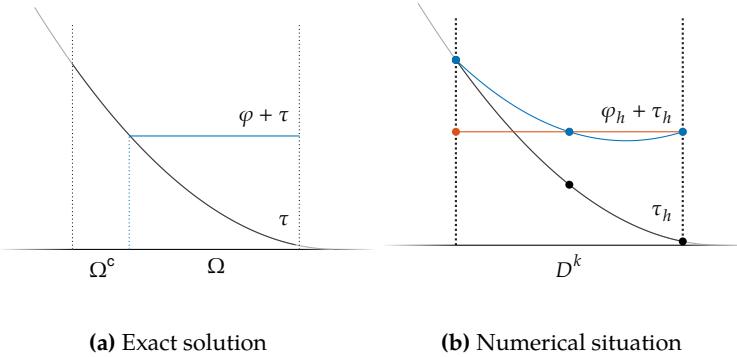
## 12.3 Well-balanced wet-dry transitions

Although the combination of timestep restriction and positivity limiter can handle dry areas, the outcome is not well-balanced. To illustrate this, let us analyze the situation in one dimension. Figure 12.1 depicts the lake at rest solution and its numerical approximation. As we can see, we cannot accurately represent  $\varphi_h + \tau_h = \text{const.}$  numerically in partly dry cells, due to the requirement of positive water heights. Consequently, the numerical representation of the lake at rest solution (11.3) has a non-zero slope. We can expect this to introduce artificial pressure gradients  $\varphi_h \nabla(\varphi_h + \tau_h)$ , in which the solution in the dry areas induces a non-zero gradient in the wet domain. Clearly, this interaction is not physically accurate as there should be no interaction between these two domains. As a consequence, spurious waves are created at the shores and pollute the domain of interest.

We propose a simple, yet effective approach to mitigate this problem. To do so, we switch to a local evaluation of the gradient using finite differences. This allows us to ignore the irrelevant dry areas and eliminate their influence on the solution. We introduce the finite difference operator

$$\mathcal{D}_\xi f(\xi_i) = \begin{cases} \frac{f(\xi_{i+1}) - f(\xi_{i-1})}{\xi_{i+1} - \xi_{i-1}} & \xi_{i+1}, \xi_i, \xi_{i-1} \text{ are wet nodes} \\ \frac{f(\xi_{i+1}) - f(\xi_i)}{\xi_{i+1} - \xi_i} & \xi_{i+1}, \xi_i \text{ are wet, } \xi_{i-1} \text{ is dry or } i-1 < 1 \\ \frac{f(\xi_i) - f(\xi_{i-1})}{\xi_i - \xi_{i-1}} & \xi_i, \xi_{i-1} \text{ are wet, } \xi_{i+1} \text{ is dry or } i+1 > p+1 \\ 0 & \text{otherwise,} \end{cases} \quad (12.6)$$

which takes the information on neighboring nodes into account only if they are wet. This finite difference operator (12.6) can be understood as a way of implicitly imposing boundary conditions within the semi-dry element. It is consistent with the physical situation as we do not have any information on what the water surface  $\varphi + \tau$  should be in the dry regions to recover the correct gradients in the wet part. We discard this



**Figure 12.1:** Comparison of the exact and numerical representations of the lake at rest solutions. The polynomial approximation cannot have a vanishing slope unless unphysical, negative water heights are allowed.

information by using the local finite difference discretization to compute the gradient of the water surface.

We can translate the finite difference operators to the physical domain as we would do for regular differentiation operators:

$$\mathcal{D}_x = \partial_x \xi \mathcal{D}_\xi + \partial_x \eta \mathcal{D}_\eta. \quad (12.7)$$

We define  $\mathcal{D}_y$  and  $\mathcal{D}_z$  in a similar fashion. Using these operators we construct the gradient-like operator

$$\nabla_h = \begin{bmatrix} \mathcal{D}_x \\ \mathcal{D}_y \\ \mathcal{D}_z \end{bmatrix}, \quad (12.8)$$

which replaces the conventional gradient operator for the computation of  $\varphi_h \nabla_h (\varphi_h + \tau_h)$ . We can now show the well-balancedness of our modified discretization:

**Proposition 12.3.1** Let  $\bar{\mathbf{q}}_h$  again be the numerical lake at rest solution (11.3) including dry domains. Then, let  $\mathbf{R}_h(\mathbf{q}_h)$ , denote the right-hand side (11.16), where we have replaced  $\nabla(\varphi_h + \tau_h)$  with  $\nabla_h(\varphi_h + \tau_h)$ . The modified scheme is well-balanced for  $\bar{\mathbf{q}}_h$ , i.e.  $\mathbf{R}_h(\bar{\mathbf{q}}_h) = \mathbf{0}$ .

*Proof.* Following the proof of Proposition 11.3.2, it is sufficient to show the final step

$$\varphi_h \nabla_h \varphi_h = -\varphi_h \nabla_h \tau_h,$$

for  $\bar{\mathbf{q}}_h$ . Due to the linearity of  $\nabla_h$ , this is equivalent to  $\varphi_h \nabla_h (\varphi_h + \tau_h) = 0$ . Inserting  $\bar{\varphi}_h$  yields

$$\begin{aligned} & \mathcal{D}_\xi \left( \max \left\{ \varphi_0 - \tau_h(x(\xi_i, \eta_j)), 0 \right\} + \tau_h(x(\xi_i, \eta_j)) \right) \\ &= \mathcal{D}_\xi \left( \varphi_0 - \tau_h(x(\xi_i, \eta_j)) + \tau_h(x(\xi_i, \eta_j)) \right) = 0. \end{aligned}$$

The first step is permissible as any water height  $\varphi \leq \varphi_{\text{tol}}$  will be ignored by  $\mathcal{D}_\xi$  anyway. The latter step holds as any finite difference of the constant function  $\varphi_0$  will yield 0. This is also true for  $\mathcal{D}_\eta$ , and consequently for  $\mathcal{D}_x$ ,  $\mathcal{D}_y$  and  $\mathcal{D}_z$ . This means that

$$\nabla_h(\bar{\varphi}_h + \tau_h) = \mathbf{0},$$

which concludes the proof.  $\square$

To maintain high-order accuracy wherever possible, we switch to this discretization only if an element contains dry areas. As such, the lower accuracy introduced by this operator is restricted to the semi-dry cells, in which one cannot expect high-order accuracy. Furthermore, the number of such cells is expected to be small.

## 12.4 A few notes on stability

Before we conclude, we discuss the effect of wetting/drying on the stability of the scheme. Numerical experiments show that the computation

of fluxes in the semi-dry cells can be unstable if performed in the wrong manner. While it might seem attractive to construct the flux term  $\partial_x \varphi u^2$  exactly using the derivatives  $\partial_x u$  and  $\partial_x \varphi$ , this leads to an unstable scheme. In general, the velocity  $u$  is not continuous at the wet-dry interface. Consequently, the evaluation of  $\partial_x u$  introduces Gibbs oscillations which render the scheme unstable. For this reason, we evaluate the derivatives of the flux variables  $\partial_x \varphi u^2, \partial_x \varphi u v, \dots$  directly. This is not exact as flux variables are first approximated by polynomials, however it avoids the problem of Gibbs oscillations associated with  $u$ . This step introduces an error, as we first approximate the flux using a polynomial of order  $p$  before we evaluate its derivatives.

Finally, high-order discontinuous Galerkin schemes oftentimes require some sort of artificial dissipation to stabilize the scheme in the presence of strong gradients and shocks. In the context of the shallow water equations, this happens in the vicinity of the wet-dry interface where the water depth is low. The lower water height leads to lower wave speeds and consequent build-up of water waves. To stabilize the scheme, we apply a filter, which modifies the solution according to

$$\mathbf{q}_h^F = \sum_{i,j=0}^p \sigma_i \sigma_j \hat{\mathbf{q}}_{ij} P_i(\xi) P_j(\eta), \quad (12.9)$$

where  $P_i, P_j$  denote the basis functions of a modal basis in one dimension and  $\hat{\mathbf{q}}_{ij}$  denotes the respective coefficient of the solution in the modal basis. We use Legendre polynomials and filter the solution using the filter weights  $\sigma_i$ , which dampen high-order modes. This acts as additional, artificial viscosity and has a stabilizing effect on the numerical scheme [78]. We choose an exponential filter with weights

$$\sigma_i = \exp(-a(i/p)^s) \quad (12.10)$$

and set the filter parameters to  $a = 30$  and  $s = 10$ . Finally, to ensure that filtering does not impact the well-balanced property of the scheme, we perform it on the reconstructed water surface (11.17). Moreover, we restrict the use of the filter to semi-dry cells. This is done to maintain the high-order accuracy of the method in wet areas, where we can expect the solutions to be sufficiently smooth. The entire post-processing procedure is described in Algorithm 12.1.

Compute the immediate solution  $\mathbf{q}_h(x, t_{n+1})$  at time  $t_{n+1}$ .

Reconstruct water surface  $\sigma_h$  (11.17).

In partly dry elements apply the filter (12.9)

Apply the positivity-preserving limiter (12.3a)

Set  $\mathbf{u} = \mathbf{0}$  on all dry nodes.

**Algorithm 12.1:** Post-processing of partly dry cells to ensure positivity and stability of the scheme.

# 13

## Numerical Results

In this chapter, we present results in one dimension, as well as in two dimensions on the rotating sphere. To analyze these results, it is useful to introduce some error norms. A classical one is the relative  $L^2$  error

$$\mathcal{E}_{L^2,\Omega} = \frac{\|\mathbf{q}_h - \mathbf{q}\|_{L^2(\Omega)}}{\|\mathbf{q}\|_{L^2(\Omega)}}, \quad (13.1)$$

where  $\Omega$  could also be replaced by a subset of the domain. The solution  $\mathbf{q}$  denotes the exact solution approximated by the numerical solution  $\mathbf{q}_h$ . As there is not always an analytical solution available, we introduce the relative mass error

$$\mathcal{E}_{\varphi,\Omega} = \frac{\int_{\Omega} \varphi_h \, dx - \int_{\Omega} \varphi \, dx}{\int_{\Omega} \varphi \, dx} \quad (13.2)$$

and energy error

$$\mathcal{E}_{E,\Omega} = \frac{\int_{\Omega} E(\mathbf{q}_h) \, dx - \int_{\Omega} E(\mathbf{q}) \, dx}{\int_{\Omega} E(\mathbf{q}) \, dx} \quad (13.3)$$

where  $E(\mathbf{q})$  denotes the total energy

$$E(\mathbf{q}) := \frac{1}{2g} \left( \varphi \|\mathbf{u}\|^2 + \varphi^2 + \varphi \tau \right). \quad (13.4)$$

For the conservation errors, we can replace the reference solution  $\mathbf{q}$  with the initial condition  $\mathbf{q}_0$ , as both mass and energy are conserved by the shallow water equations.

### 13.1 Results in one dimension

#### Standing wave

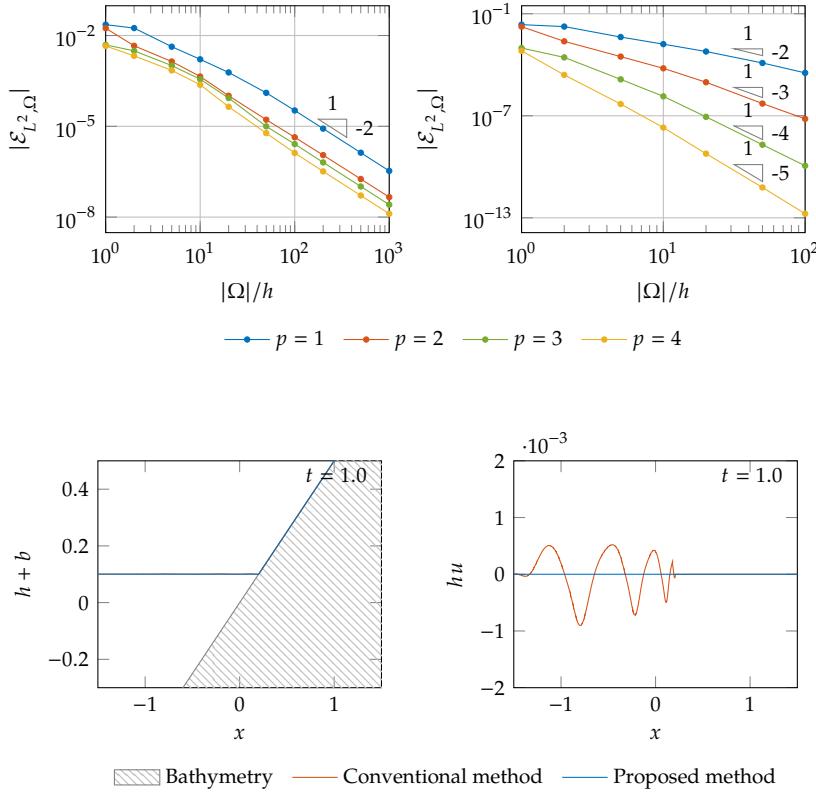
Before discussing some common test cases for the shallow water equations in one dimensions, we investigate the accuracy of the numerical scheme that we have derived for partly dry cells. To do so, we enforce a smooth solution  $\varphi, \varphi u \in C^\infty$  of the form

$$\begin{aligned} \varphi &= \varphi_0 + \varphi_A \cos \omega t \sin \kappa x, \\ \varphi u &= -\varphi_A \frac{\omega}{\kappa} \sin \omega t \cos \kappa x, \end{aligned}$$

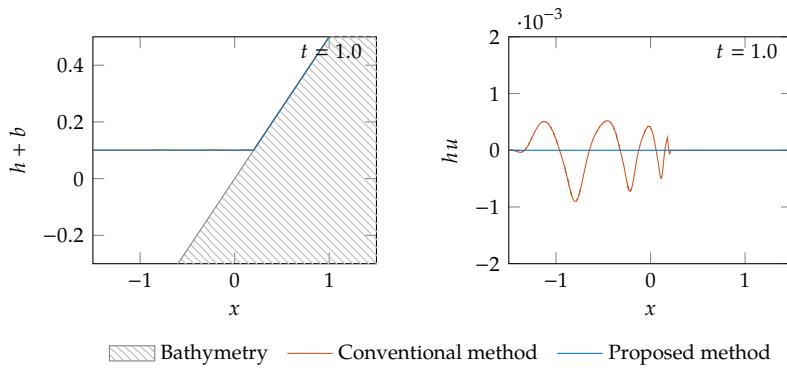
with  $\varphi_0 = 0.3$ ,  $\varphi_A = 0.1$ ,  $\omega = \pi$ ,  $\kappa = \pi$ . The domain is set to  $\Omega = [-0.5, 0.5]$  and exact Dirichlet boundary conditions are used. We enforce the solution by choosing a suitable right-hand side which yields this solution.<sup>1</sup> We compare the convergence behavior of the regular flux discretization to the wetting/drying flux discretization (12.8). The

13.1 Results in one dimension . . . . .	132
Standing wave . . . . .	132
Lake at rest . . . . .	133
Dam break on a dry domain	135
Oscillating lake . . . . .	137
13.2 Results on the sphere . . . . .	138
Lake at rest solution . . . . .	139
Adaptive mesh refinement .	139
Tsunami simulations . . . . .	140
13.3 Dynamic source models . . . . .	143
Tohoku tsunami . . . . .	145
Sumatra-Andaman tsunami	145
13.4 Concluding remarks . . . . .	148

1: A suitable forcing term can be found by inserting in the solution into the shallow water equations.



**Figure 13.1:** Convergence behavior for the standing wave solution. We compare the accuracy of the wetting/drying flux discretization on the left to the accuracy of the regular discontinuous Galerkin discretization on the right.

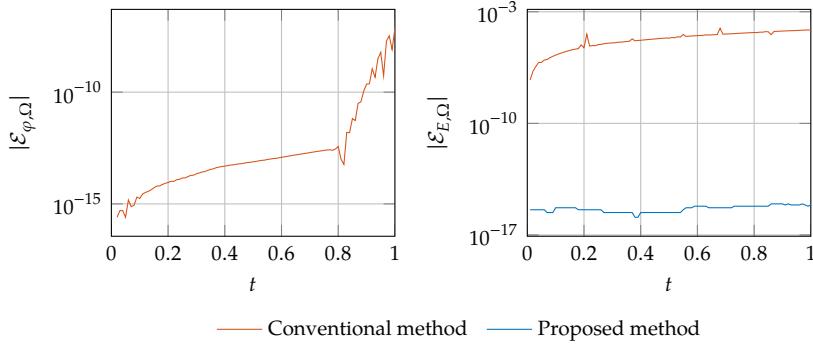


**Figure 13.2:** Lake at rest solution on a sloping bed at  $t = 1$ . We compare the conventional DG method to our well-balanced, which takes wetting/drying into consideration.

results are shown in Figure 13.1. Instead of using the wetting/drying discretization only in partly dry cells, we use it on the entire domain to generate the results on the left. The results on the right depict the regular discretization as a baseline for comparison. We observe that the unmodified discontinuous Galerkin scheme achieves high-order accuracy as we would expect. In contrast, we recover only second order accuracy with the wetting/drying discretization, even with higher-order polynomial approximants. We note that the discretization of the flux term remains unchanged if we choose the finite difference approximation for a first order discontinuous Galerkin method on a fully wet cell. For higher order approximations, we do not use the full gradient information available to us and we only see an improvement in the constant but not in the order of convergence. Thus, we can be confident that this change of flux discretization will indeed properly converge to the exact solution, provided it is sufficiently smooth. In practice, the transition from wet to dry areas is continuous but not differentiable and we therefore cannot expect more than first order accuracy using polynomial approximations. For two-dimensional problems, this loss of accuracy is acceptable however, as we can expect the number of partly dry elements (located at the shores) to scale as  $\mathcal{O}(h^{-1})$  in contrast to the overall number of elements, which scales as  $\mathcal{O}(h^{-2})$ .

### Lake at rest on a sloping beach

Let us verify the well-balanced property for one-dimensional problems. To do so, we will use the *lake at rest* solution on a linearly sloping bed with  $b(x) = 0.5x$ ,  $g = 9.81$ ,  $\Omega = [-1.5, 1.5]$  and  $\varphi_0 = 0.1005g$  as depicted



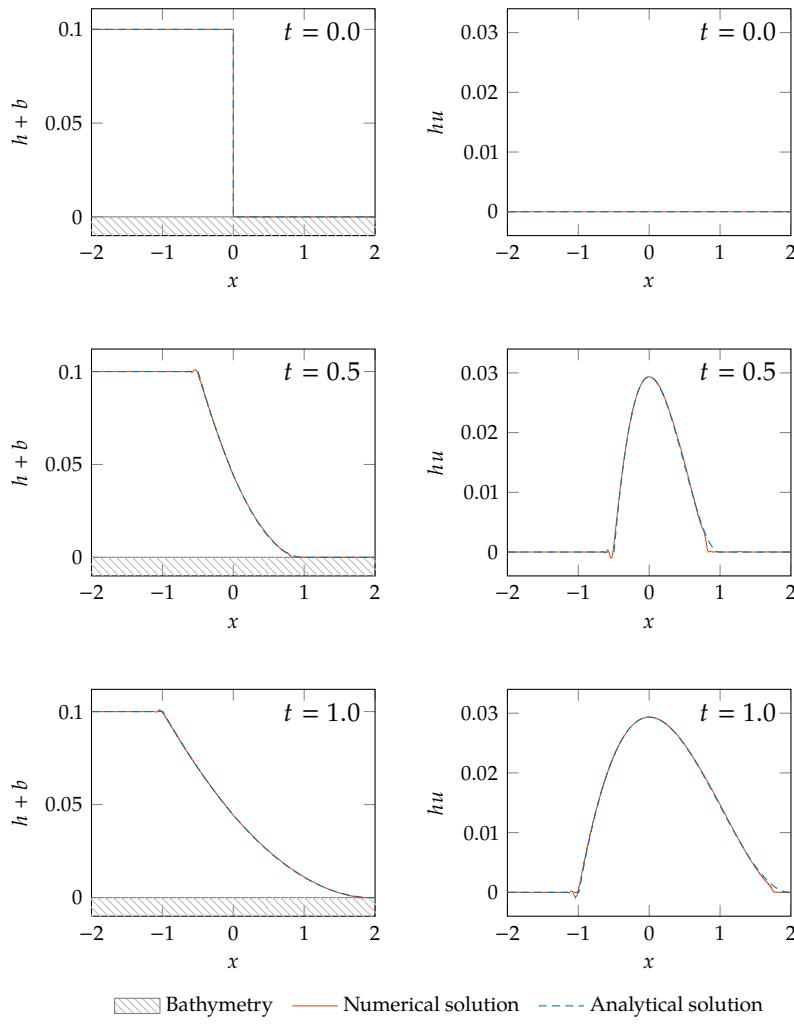
**Figure 13.3:** Comparison of the conservation errors of the proposed method to a conventional DG discretization. Mass conservation errors are shown on the left in logarithmic scale. The error of our method cannot be displayed as it is exactly 0. Energy conservation errors are shown on the right.

in Figure 13.2. We do not apply filtering and use the parameters  $p = 3$ ,  $K = 100$ ,  $\Delta t = 5 \cdot 10^{-5}$ ,  $\varphi_{\text{tol}} = 10^{-4}$ . As boundary conditions we use the exact solution on the opposing side of the interface.

Figure 13.2 compares the numerical results obtained with our method using finite difference approximations in the volume integral, to results obtained with the conventional DG discretization using the polynomial derivatives. The latter method corresponds to the positivity-preserving discontinuous Galerkin discretization presented in [116]. In contrast to [116], we ensure that the wet-dry interface does not coincide with the cell interface. This is important as otherwise, we would only be testing the well-balanced property for completely flooded cells, which is not sufficient for practical applications. Even with the existence of such a challenging semi-dry cell, we observe that our method preserves the lake at rest solution well. This is in contrast to the conventional method, where we observe artificial waves being created which propagate into the domain and therefore pollute the solution with spurious waves. These are especially noticeable in the discharge plot in Figure 13.2, which shows non-zero discharges from the semi-dry cell to the left. As we have discussed previously in Section 12.3, this is caused by artificial pressure gradients. The magnitude of the waves that are created might seem negligible. However, in our experience, this is not the case when we move to two dimensional problems. It is therefore clear that the semi-dry cells require a careful treatment to ensure the well-balanced property of the scheme.

The wetting/drying discretization (12.6) performs well as seen in Figure 13.2. In particular, we do not observe any spurious changes in the water surface. For the lake at rest solution, exact integrations implies both conservation of mass and energy as well as the well-balanced property. We can therefore use the mass and energy conservation errors to verify the well-balanced property for our scheme as it is based on integrating the lake at rest solution exactly. Figure 13.3 displays the relative mass and energy errors over time for both our scheme and a conventional DG discretization. While the latter does not conserve either energy or mass, we observe that our method does so to within machine precision.<sup>2</sup> In fact the mass error is exactly 0. For the conventional method we observe that the mass error is initially small while energy errors are accumulated right away. This is caused by the artificial pressure gradients at the wet-dry interface as discussed previously. Once these waves begin interacting with the boundary at  $t \approx 0.8$ , the rate at which mass conservation is

2: The well-balanced property implies exact conservation of mass and energy in the case of the associated stationary solution.



**Figure 13.4:** Dam break on a dry bed at different times. Comparison of the numerical solution with  $p = 4$  and  $K = 100$  to the analytical solution.

violated begins to increase.

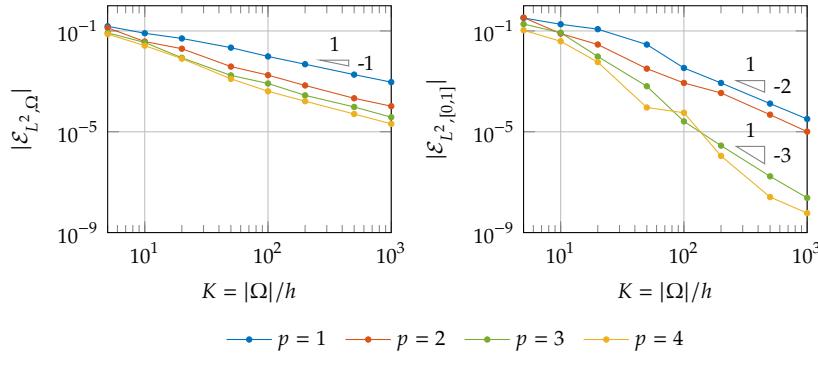
### Dam break on a dry domain

We move on to some dynamical test cases. In the next test case we model a dam break over a dry bed. Initially, we have a water reservoir of height  $\varphi_l = 0.1g$  on the left half-plane and a dry domain on the right one. We assume the dam break to be instantaneous and the bottom to be flat, i.e.  $\tau = 0$ . The analytical solution (and initial condition) of this problem is given by

$$\varphi(x, t) = \begin{cases} \varphi_l & x \leq x_A(t) \\ \frac{4}{9} \left( \sqrt{\varphi_l} - \frac{x-x_0}{2t} \right)^2 & x_A(t) < x < x_B(t) \\ 0 & x \geq x_B(t) \end{cases}$$

and

$$u(x, t) = \begin{cases} 0 & x \leq x_A(t) \\ \frac{2}{3} \left( \sqrt{\varphi_l} + \frac{x-x_0}{t} \right) & x_A(t) < x < x_B(t) \\ 0 & x \geq x_B(t) \end{cases}$$



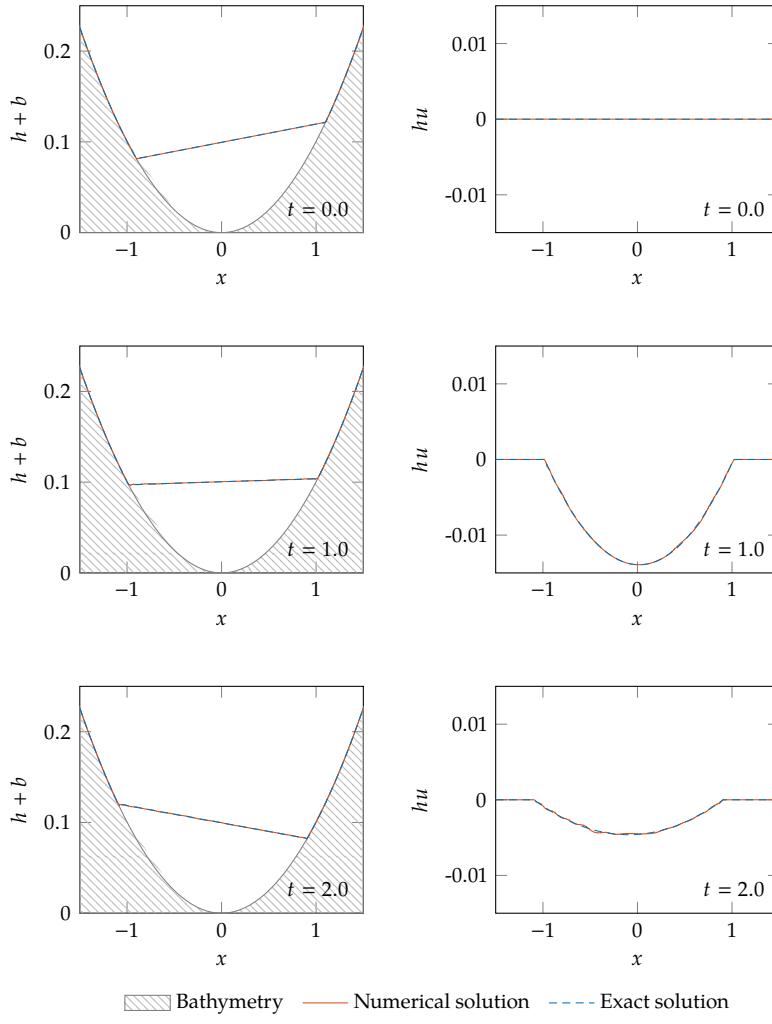
**Figure 13.5:** Convergence for the dam break solution on a dry bed. The global  $L^2$  error on  $\Omega$  on the left is compared to the local  $L^2$  error on the right, which is computed in region  $[0, 1]$  on the right. The solution is fully smooth in this region.

where  $x_A(t) = x_0 - t\sqrt{\varphi_l}$  and  $x_B(t) = x_0 + 2t\sqrt{\varphi_l}$  are the positions of the kinks in the solution [128]. This problem corresponds to the Riemann problem with a right state of  $\varphi = 0$ ,  $\varphi u = 0$ . This test case is particularly challenging due to the presence of the rarefaction wave at the wet-dry interface [98]. Thus, the scheme has to accurately resolve the shock while maintaining positivity, which can be expected to be challenging. Many schemes fail at this test case due to stability issues at the wet-dry interface. For this test case, we use both filtering and the positivity-preserving limiter and set the dry tolerance to  $\varphi_{tol} = 10^{-6}$ .

Figure 13.4 depicts the numerical solution in comparison to the analytical solution. The former is computed using  $p = 4$ ,  $K = 100$ , exact boundary conditions and a timestep of  $\Delta t = 5 \cdot 10^{-5}$ . We observe an excellent match between the solutions, as well as an accurate prediction of the location of the shore.

We investigate the accuracy of the method and consider the convergence of the  $L^2$  error. The solution is initialized at  $t = 0.1$  and filtering is only used in the partly dry cells. This is done to eliminate the effect of filtering as much as possible from the convergence results. Filtering is necessary on the other hand to yield a stable scheme. The simulation is run until  $t = 1$  with  $\Delta t = 5 \cdot 10^{-5}$  for varying polynomial orders  $p$  and mesh widths  $h$ . The wet-dry tolerance  $\varphi_{tol}$  requires special attention, as it can severely affect the accuracy if it is set too high or make the scheme unstable if it is set too low. In our experience, a reciprocal linear relation  $\varphi_{tol} = 10^{-4}/p$  yields satisfying results. This relation can be unnecessarily small as we have observed much higher tolerances to be possible for small mesh widths. Figure 13.5 shows the results of the convergence analysis. We compare the convergence of the relative  $L^2$  error on the entire domain  $\Omega = [-2, 2]$  to the convergence of the error in the smooth subdomain  $[0, 1]$ .

This can be an interesting test as the latter is initially dry but fully wet at  $t = 1$ . Moreover, the solution is a third order polynomial, which means that we cannot expect more than third order accuracy. While global convergence rates are limited due to the low regularity of the solution, we can indeed observe up to third order convergence rates in the wet areas where the solution is smooth. The convergence rates are not optimal however, as cubic ansatz functions should theoretically achieve machine precision accuracy in the smooth part if the scheme is exact. In practice, this is not possible due to the presence of the wetting/drying process and the errors associated with the flux discretization in the



**Figure 13.6:** Oscillating lake in a parabolic bed. Comparison of the exact solution to the numerical solution computed with  $p = 2$  and  $K = 100$ .

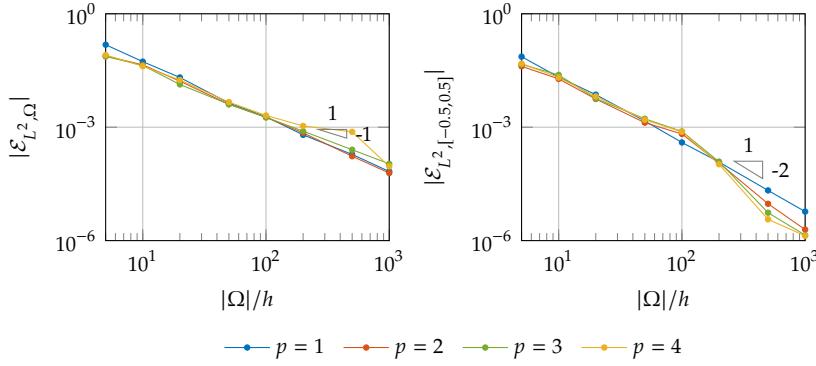
volume terms (10.18). We can expect errors to propagate into the rest of the solution, which causes the suboptimal convergence rates. The results are nonetheless encouraging and imply that we can expect higher order accuracy in wet areas, even if low order errors propagate into these areas. This is consistent with the behavior of DG methods in the presence of shocks, where Gibbs oscillations can pollute other areas far way from the shock [78]. We conclude that high-order accuracy can indeed be achieved in fully wet areas of the domain even in the presence of wet-dry transitions.

## Oscillating lake

Our final, one-dimensional test case is the oscillating lake in a parabolic channel

$$\begin{aligned}\varphi(x, t) &= \max \left\{ \varphi_0 + 2\varphi_0 \alpha \cos(\omega t) \left( x - \frac{\alpha}{2} \cos(\omega t) \right) - \varphi_0 x^2, 0 \right\}, \\ (\varphi u)(x, t) &= -\varphi(x, t) \alpha \omega \sin(\omega t).\end{aligned}$$

This is the analytical solution to the one-dimensional shallow water equations in a parabolic bed given by  $\tau(x) = \varphi_0 x^2$ , where  $\omega = \sqrt{2\varphi_0}$  is the frequency of the oscillation [129]. We choose the parameters



**Figure 13.7:** Convergence for the oscillating lake in a parabolic channel. Global  $L^2$  errors on the left are compared to the local error in a subdomain which remains wet at all time, on the right.

$\varphi_0 = 0.1005g$ ,  $\alpha = 0.1$  and set the domain to  $[-1.5, 1.5]$ . Boundary conditions do not play a role, as the two shore points should never reach the boundaries of the domain. For this example, filtering is performed directly on the conserved variables  $q_h$  as there is no meaningful water surface to reconstruct. This is because this solution is far from the lake at rest solution. Figure 13.6 compares the numerical solution, obtained with  $p = 2$ ,  $K = 100$ ,  $\Delta t = 1 \cdot 10^{-4}$  and  $\varphi_{\text{tol}} = 10^{-4}$  to the analytical solution. Although this is a challenging test case, due to the constant wetting and drying, we observe that the scheme remains stable and properly reconstructs the shores.

Figure 13.7 depicts the results of the convergence analysis for the oscillating lake problem. We compare global errors to local errors for a subdomain of the solution which remains wet at all times. To this end, we evaluate the relative  $L^2$  error both on the entire domain, as well as on the subdomain  $[-0.5, 0.5]$ . As in the dam break test, we observe that the convergence of the  $L^2$  error on the entire domain is only slightly higher than 1. Again, this is caused by spatial approximation errors at the wet-dry interface dominating the global approximation error. We have to accept this error as we cannot expect to do better than first order accuracy at the shore. In the smooth part however, the solution lies in the ansatz space  $V_h$  and high-order convergence is possible. The analytical solution is a quadratic polynomial in space and we observe convergence up to second order. This is similar to the behavior that we observed with the dam break case, where we could not achieve errors in the order of machine precision. Once again, we conclude that this is caused by errors introduced by the wetting/drying process.

## 13.2 Results on the sphere

We are finally ready to apply our methods to discretizations of the spherical shallow water equations, formulated on the rotating sphere. For all remaining simulations, we set physical constants to the values of Earth:  $R = 6.37122 \cdot 10^6$  m,  $g = 9.80616$  m/s $^2$  and  $\omega = 7.29 \cdot 10^{-5}$  rad/s. The bottom topography  $\tau = gb(x)$  is then generated by piecewise linear interpolation of the ETOPO1 Earth Relief dataset [130]. Moreover, we set the water surface globally to  $\varphi_0 = 0$  m relative to sea level, thus ignoring tidal effects. This approximation is acceptable as tidal effects are largely irrelevant for the propagation of tsunamis.

We use the icosahedral meshes as described in [91] and refine the mesh uniformly until a prescribed refinement level of  $L_{\text{uni}}$  is reached. Then, we define a circular region of interest on the sphere defined by

$$\{\mathbf{x} \in S^2(R) \mid d_{S^2}(\mathbf{x}, \mathbf{x}_0) \leq \rho\}.$$

Here,  $d_{S^2}(\cdot, \mathbf{x}_0)$  is the great-circle distance with respect to a point  $\mathbf{x}_0$  on the sphere. The angle  $\rho$  specifies the radius of the area of interest. In this region, the grid is further refined locally until a desired refinement level of  $L_{\text{loc}}$  is reached. After static mesh refinement, the bottom topography data is interpolated onto the locally refined grid in order to obtain maximally accurate topography data. Throughout our experiments, we set the filtering and wetting/drying parameters to  $s = 10$ ,  $\alpha = 30$  and  $\varphi_{\text{tol}} = g \cdot 10\text{m}$ .

## Lake at rest solution

We seek to verify the theory developed in Chapter 11 and Chapter 12 by verifying that the method is well-balanced on non-conforming, curved meshes. To do so, we construct a non-conforming mesh by refining the icosahedral mesh in a circular region with radius  $\rho = 40^\circ$  and center  $\mathbf{x}_0$  at  $-10^\circ$  longitude and  $-10^\circ$  latitude. On this mesh, we initialize the *lake at rest* solution and run the simulation until  $t = 10\text{d}$  is reached, which exceeds the timescale of tsunami events. The results obtained with various polynomial degrees at  $t = 5\text{d}$  and  $t = 10\text{d}$  are listed in Table 13.1. We see that in all four cases, errors are close to accumulated roundoff errors, which confirms that the method is well-balanced on curved, non-conforming grids with bottom topography and dry cells in the domain.

## Adaptive mesh refinement

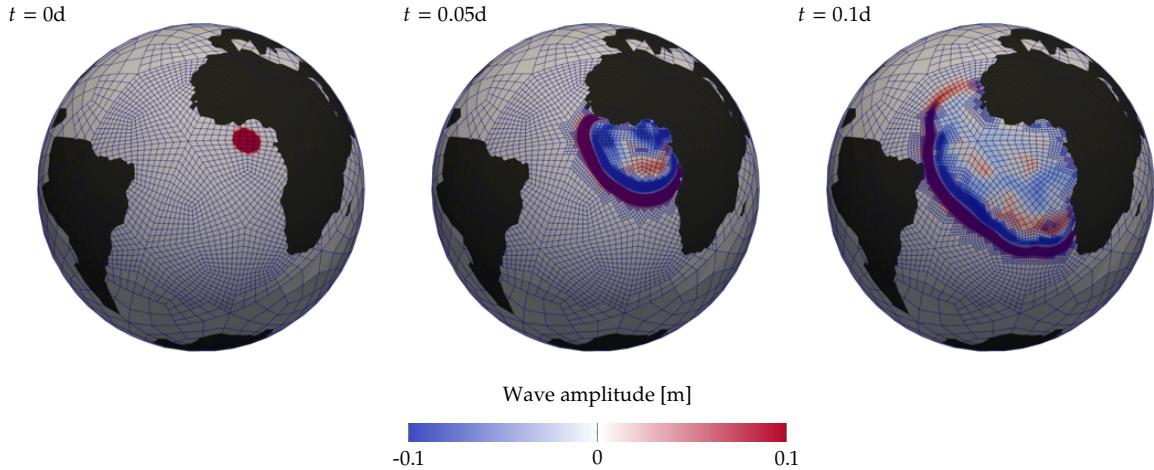
Before we move to simulations of actual tsunami events, we demonstrate that the methods for non-conforming discretizations allow for adaptive mesh refinement in a well-balanced manner. We re-use the same mesh and generate a tsunami at  $0^\circ$  longitude and  $0^\circ$  latitude. For the initial shape we choose a Gaussian of the form

$$\varphi + \tau = \varphi_0 + \varphi_a \exp(-(d_{S^2}(\mathbf{x}, \mathbf{x}_0) / \rho_d)^2),$$

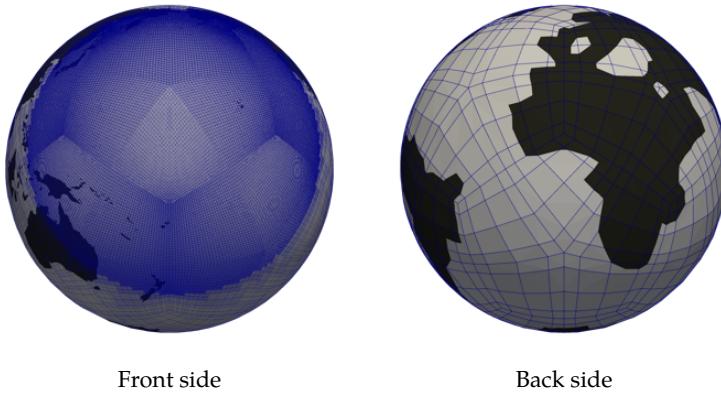
with a wave height of  $\varphi_a = g \cdot 10\text{m}$  and a width of  $\rho_d = 0.1\text{rad}$ . We utilize adaptive mesh refinement every 50 time steps, and adapt the mesh based on a velocity criterion. According to this criterion, refinement or coarsening is performed whenever the absolute velocity  $\|\mathbf{u}\|_2$  exceeds or

**Table 13.1:** Relative errors for the lake at rest solution on the sphere at  $t = 5\text{d}$  and  $t = 10\text{d}$  with various polynomial orders.

$p$	$L^2$ error $\mathcal{E}_{L^2, \Omega}$		Mass error $\mathcal{E}_{\varphi, \Omega}$		Energy error $\mathcal{E}_{E, \Omega}$	
	$t = 5\text{d}$	$t = 10\text{d}$	$t = 5\text{d}$	$t = 10\text{d}$	$t = 5\text{d}$	$t = 10\text{d}$
1	$2.428 \cdot 10^{-15}$	$5.015 \cdot 10^{-15}$	0.0	0.0	0.0	0.0
2	$7.592 \cdot 10^{-13}$	$1.641 \cdot 10^{-12}$	$-3.324 \cdot 10^{-15}$	$6.980 \cdot 10^{-14}$	$-1.000 \cdot 10^{-13}$	$-1.239 \cdot 10^{-13}$
3	$8.388 \cdot 10^{-13}$	$1.828 \cdot 10^{-12}$	$2.350 \cdot 10^{-14}$	$2.085 \cdot 10^{-15}$	$-6.134 \cdot 10^{-14}$	$-1.023 \cdot 10^{-13}$
4	$6.957 \cdot 10^{-13}$	$2.858 \cdot 10^{-13}$	$-1.360 \cdot 10^{-14}$	$-5.247 \cdot 10^{-14}$	$-5.674 \cdot 10^{-14}$	$-7.318 \cdot 10^{-14}$



**Figure 13.8:** Adaptive simulation of a solitary wave at  $0^\circ$  longitude and  $0^\circ$  latitude. The mesh is refined adaptively using our well-balanced mesh refinement. The corresponding video clip can be found [bonevbs.github.io/files/amr\\_showcase.mp4](http://bonevbs.github.io/files/amr_showcase.mp4).

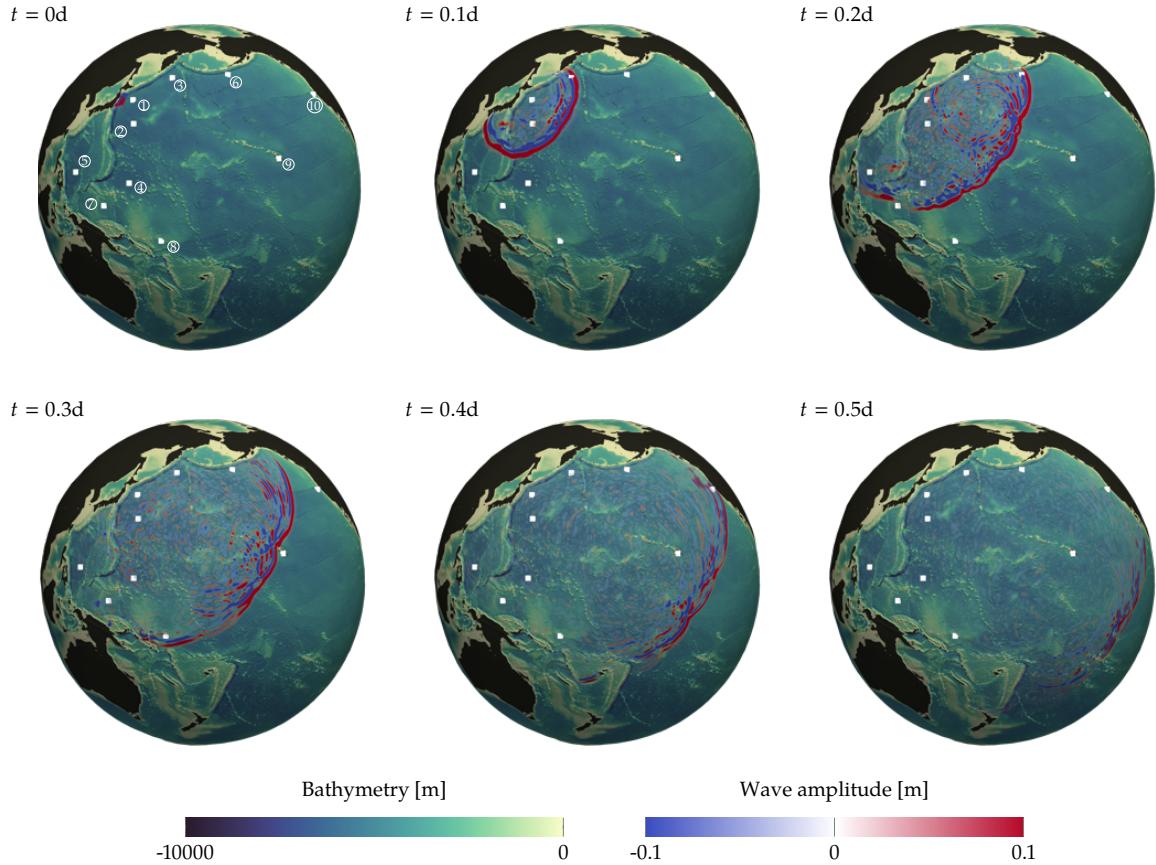


**Figure 13.9:** Mesh for the Tohoku tsunami simulation using biquartic polynomials i.e.  $p = 4$ . The mesh contains 14238 elements with 25 collocation points in each element.

falls below the threshold of  $0.01\text{m/s}$ . Figure 13.8 depicts the numerical solution at three different times. We observe that the mesh refinement is handled in a robust manner without introducing any spurious waves at non-conforming interfaces. Hence, this technique allows for dynamically refined meshes, while preserving the well-balanced property of the scheme. This potentially increases the efficiency of the method as computational effort can be directed towards regions of interests which require more accuracy.

## Tsunami simulations

We seek to validate the capability of the algorithm through numerical simulations of the 2011 Tohoku tsunami event, which occurred off the Japanese coast in 2011. As an initial mesh, we use the unrefined icosahedral mesh and locally refine the area of interest in the Pacific Ocean. We select the point  $x_0$  at  $-177^\circ$  longitude,  $12^\circ$  latitude and refine the initial grid within a radius of  $\rho = 55^\circ$  until a refinement level of  $L_{\text{loc}} = 5$  is reached. This amounts to a grid with a total of 14238 elements, most of which are located within the region of interest in the Pacific Ocean., see Figure 13.9. Using this approach, we can automatically generate meshes that are adapted to any region of interest and avoid boundary conditions at the cost of a few extra elements on the back side of the sphere. Moreover, we



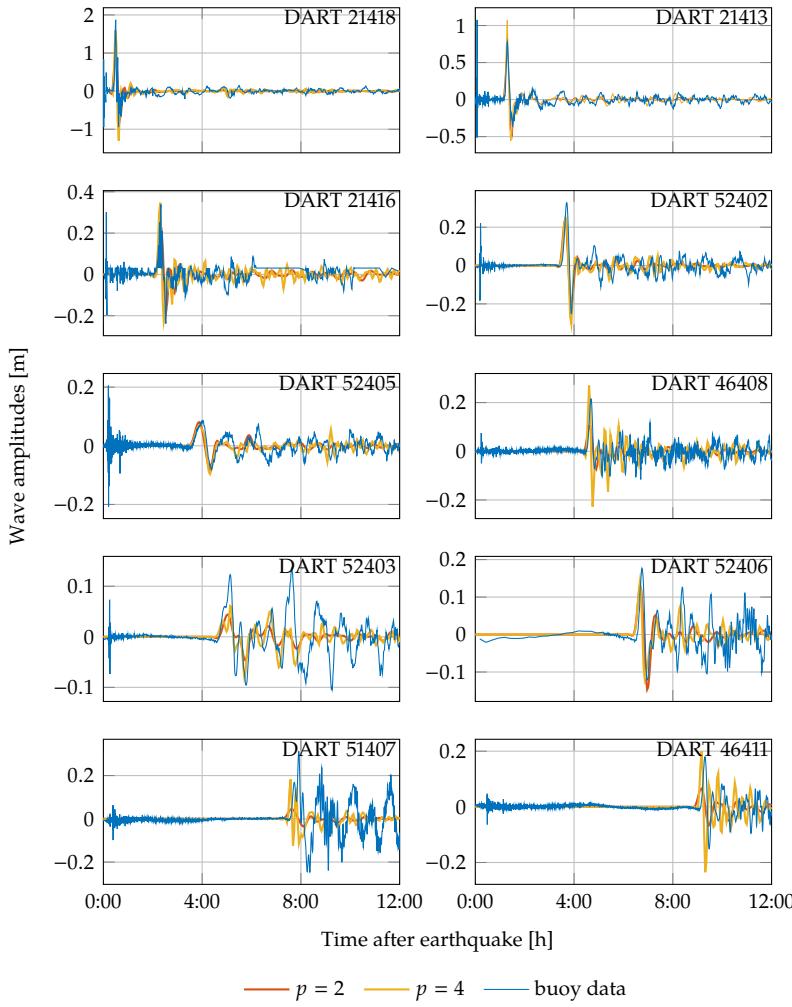
**Figure 13.10:** High-fidelity simulation of the Tohoku tsunami event with  $K = 14238$  and  $p = 4$ . The initial condition is taken to be a static initial water surface displacement from the lake at rest steady state. The buoys are numbered in ascending order with respect to the tsunami arrival times: 1 - DART 21418, 2 - DART 21413, 3 - DART 21416, 4 - DART 52402, 5 - DART 52405, 6 - DART 46408, 7 - DART 52403, 8 - DART 52406, 9 - DART 51407, 10 - DART 46411. The corresponding video clip can be found at [bonevbs.github.io/files/tohoku.mp4](http://bonevbs.github.io/files/tohoku.mp4).

would like to note that this type of mesh avoids the need for boundary conditions, as needed if an artificial domain truncation was considered.

The initial condition is generated using the Okada model [131] and the fault parameters of model III presented in [132]. For the moment, we are not concerned with the tsunami source model and delay the discussion of Okada solutions and tsunami sources to Section 13.3. For the moment, we assume an instantaneous slip and calculate the final displacement of the bottom topography, which is reached 200s after the earthquake has occurred. The resulting bathymetry displacement is directly translated to an initial displacement of the water surface.

Starting 200s after the initial earthquake at 2011-3-11 05:49:04 UTC, we simulate 12h of tsunami propagation. We repeat the simulation twice; once with biquadratic polynomials  $p = 2$  and a second time with biquartic polynomials  $p = 4$ . The simulations were run on a single core of an Intel Xeon E5-2643 v3 processor, clocked at 3.40GHz. The simulation took 1h01m for  $p = 2$  and 12h36m for  $p = 4$ . Figure 13.10 depicts our results using biquartic polynomials. We observe that we are able to simulate 12h of tsunami propagation in a stable manner. Moreover, visual comparison of both solutions show no significant differences and we can expect our method to be consistent under  $p$ -refinement.

This is further reinforced by our comparison with buoy data. In Figure



**Figure 13.11:** Comparison of wave amplitudes extracted from DART buoy data to amplitudes extracted from the simulations.

13.11 we show a comparison of real-world buoy data that was recorded during the Tsunami event to wave amplitudes extracted from the numerical simulations. We choose 10 buoys, which are part of the DART tsunami monitoring network [133] as reference. These buoys use pressure data from the sea bottom to infer the wave amplitudes at the surface. Their positions are marked as white dots in Figure 13.10. The oscillations recorded early on by the sensors can therefore be ignored as they are caused by seismic waves traveling through the Earth's crust. Moreover, our simulation does not take tidal effects into account, which we compensate for by subtracting the time-averaged water height from the time series of the buoy data.

As we can see from Figure 13.11, both simulations are able to accurately predict arrival times even over long distances. At longer propagation times, we observe a slight shift of both signals compared to the buoy data. This phase shift is caused by unresolved bathymetry and becomes noticeably smaller with the higher resolution simulation. Likewise, the time series of the buoy data is matched more accurately by the higher-order simulation. The incremental improvement in the time series from  $p = 2$  to  $p = 4$  suggests that the method is indeed consistent.

Table 13.2 summarizes the arrival times and states the absolute and relative errors in the amplitudes of the initial waves for the high-resolution

buoy	arrival time	forecast	amplitude error	relative error
DART 21418	06:13:04	06:13:04	-0.279m	-0.149
DART 21413	06:59:16	06:59:16	0.297m	0.382
DART 21416	07:59:16	07:58:05	0.008m	0.024
DART 52402	09:17:52	09:15:28	-0.079m	-0.242
DART 52405	09:26:52	09:20:52	-0.016m	-0.195
DART 46408	10:20:16	10:18:28	0.054m	0.251
DART 52403	10:31:40	10:25:40	-0.062m	-0.505
DART 52406	12:24:48	12:15:28	-0.033m	-0.186
DART 51407	13:25:40	13:17:52	-0.129m	-0.415
DART 46411	14:57:28	14:50:16	0.020m	0.110

**Table 13.2:** Comparison of forecasted and recorded tsunami arrival times. Absolute and relative errors in the height of the initial tsunami wave are also stated. The results are extracted from the refined simulation using biquartic polynomials i.e.  $p = 4$ .

simulation. As we have already observed in Figure 13.11, arrival times are quite accurate in the near field but begin to diverge in the far field. The largest error is smaller than 8 minutes, which is still quite accurate considering the propagation time of more than 7 hours. Moreover, errors in the predicted wave amplitudes do not exceed 0.3m even though the waves propagate over thousands of kilometers. We remark that for both arrival times and wave amplitudes, the largest errors occur in the signals of DART 52403 and DART 51407. As previously mentioned, this is most likely caused by underresolved bathymetry and uncertainties in the initial condition. DART 51407 is positioned close to the coast of Hawaii and the tsunami has to travel along the Hawaiian-Emperor seamount chain to reach it. Because the bathymetry is underresolved, small islands in the path of the tsunami are missing. This can be expected to have a noticeable effect on the computed solution as these islands would normally cause reflections that are now missing. This is an interesting effect as underresolved bathymetry would normally only cause refraction as long as it is submerged. This only results in a phase error which we can observe in the buoy data. Once the bathymetry reaches the water surface, the wavespeed  $c = \sqrt{\varphi}$  becomes 0 and waves begin to be reflected. The resulting error is more dramatic than a mere phase shift and we would expect larger errors. Nevertheless, the time series data in Figure 13.11 indicates that the method is able to capture the physical effects accurately enough to predict the long-term evolution of the tsunami waves.

Finally, we remark that the proposed method can be easily adapted to allow well-balanced  $p$ -refinement. In this way, we could fully leverage the flexibility of  $hp$ -adaptivity that discontinuous Galerkin methods have to offer. This opens up the possibility of using elements with low polynomial order in coastal areas, where  $h$ -refinement is preferable due to the restrictions in accuracy caused by the wetting/drying process. High-order approximations, on the other hand, retain their advantages [78, 134] in the wet parts of the domain, as we have seen with the one-dimensional examples. An  $hp$ -adaptive scheme would then allow to leverage high-order elements in these areas where the solutions can be expected to be smooth.

### 13.3 Dynamic source models

In the previous section we have demonstrated the ability to accurately model tsunami propagation on a large scale. The accuracy of the outcome is fundamentally related to the quality of the initial conditions that are used. This raises two questions; namely what kind of source model should be considered and how the initial condition should be obtained. The latter

question has been covered extensively in the literature and there exist a variety of methods to do so. The main approach used in the literature is seismic inversion, where the traces of the seismic waves are used to reconstruct the original displacement of the sea bed displacement [132]. A notable approach in the context of tsunami simulation is described in [135], which proposes to use incoming data from buoys in conjunction with an adjoint model of the discontinuous Galerkin shallow water model, to directly reconstruct the initial conditions.

For the source model, there are essentially two approaches. The first approach is to assume that the timescales of the slip process<sup>3</sup> are much faster than the timescales of the wave propagation. Under this assumption, we only need to know how the bottom topography is deformed due to the slip to impose this deformation on the water surface height. We therefore replace  $\tau$  with  $\tau'$ , which is the updated bottom topography due to the slip. The new water surface height is then  $\varphi + \tau'$ . In other words, the water waves had no time to react to the instantaneous change in the bottom topography and the change in the sea surface height is instantaneous. To model the deformation of the seabed, one often assumes that it can be written as a sum of so-called Okada solutions, which model the static deformation on the surface due to dislocations in the halfspace beneath it [131, 136]. Consequently, we write the displacement of the seabed as

$$\tau'(\mathbf{x}) - \tau(\mathbf{x}) = \sum_{i=1}^{n_{\text{faults}}} O_i(\mathbf{x}), \quad (13.5)$$

where  $O_i(\mathbf{x})$  denote the Okada solutions. Each of them is associated to one single dislocation out of  $n_{\text{faults}}$  dislocations beneath the seabed surface. These dislocations are therefore called *subfaults*. We avoid going into further details on the geophysical models and instead refer the reader to the original paper [131].

One alternative approach to this static model is to consider a dynamic rupture model, which allows the bottom topography to vary in time. A possible approach to model the dynamic fault process is to activate each Okada solution separately using an individual activation function  $\sigma_i(t)$  for each subfault [136, 137]. The resulting time-dependent deformation of the seabed can therefore be written as

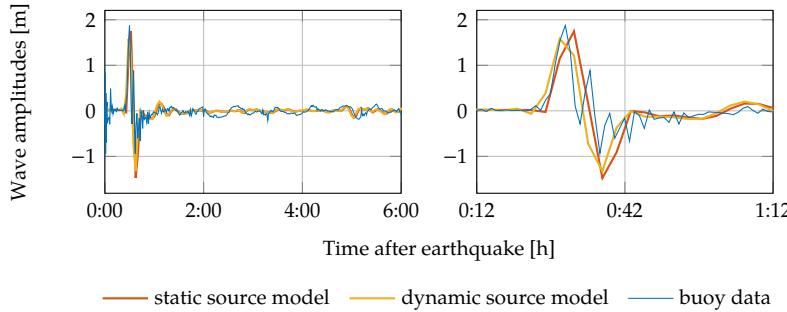
$$\tau'(\mathbf{x}, t) - \tau(\mathbf{x}) = \delta(\mathbf{x}, t) = \sum_{i=1}^{n_{\text{faults}}} \sigma_i(t) O_i(\mathbf{x}). \quad (13.6)$$

A possible choice for the activation function is the piecewise linear activation function

$$\sigma_i(t) = \begin{cases} 0 & t \leq t_{0,i} \\ \frac{t-t_{0,i}}{t_{1,i}-t_{0,i}} & t_{0,i} \leq t \leq t_{1,i} \\ 1 & t_{1,i} \leq t \end{cases}$$

where  $t_{0,i}$  and  $t_{1,i}$  denote the start and end times of the slip process occurring at the  $i$ -th subfault [137]. In the following, we use this simple dynamic fault model to evaluate the influence of the seabed dynamics on the tsunami simulation and resulting predictions. To do so, we compare results obtained with both dynamic and static source models for the 2011 Tohoku tsunami event, as well as the 2004 Sumatra-Andaman tsunami.

<sup>3</sup>: Faults refer to the interface between two blocks of rock. Slips refer to the displacement of opposite sides of the fault. Typically this happens in an abrupt manner, which causes the earthquake.



**Figure 13.12:** Comparison of wave amplitudes extracted from DART buoy 21418 during the Tohoku tsunami event to wave amplitudes computed with the static and dynamic source models. On the right the signal is zoomed in to better depict the initial wave.

## Comparison of source models for the Tohoku tsunami

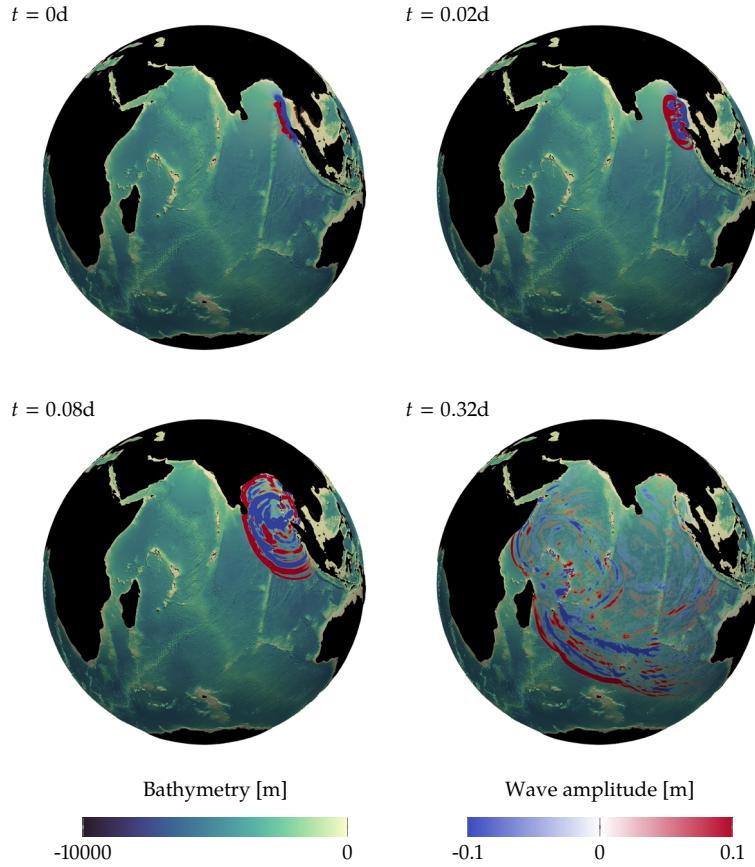
We repeat the previous experiment of Section 13.2 and simulate the 2011 Tohoku tsunami event. As source model, we re-use the model from [132] as before. This time, however, we also use the dynamic parameters listed in [132] to be able to compare both source models. These models contain a total number of  $n_{\text{faults}} = 190$  subfaults and the rupture process is basically static after 105 seconds. To facilitate this in our model, we run the simulation at smaller timesteps until the rupture process is completed.

The resulting simulation is visually indistinguishable to the results shown in Figure 13.10 and the resulting differences in wave signals is extremely small. To compare both models, we use the extracted wave signals at DART buoy positions. Figure 13.12 depicts the signal of DART 21418, as well as the wave signals extracted from simulations using both source models. The differences are small and both source models produce almost identical solutions with only a small difference in the phase.

However, the dynamic source model is able to better approximate the arrival time with a smaller phase error, while the static model seems to better predict the amplitude of the initial wave. Similar comparisons for other buoys in the wave field show similar, but small differences between both models. From this test case alone we conclude that the static fault model is good enough and that the dynamic model does not offer significant improvements. In fact, it is likely that most rupture processes and tsunami formations can be well-approximated with an instantaneous slip as the assumption that the change in bottom topography results in an equally instantaneous change at the water surface, is reasonable for most scenarios.

## Comparison of source models for the Sumatra-Andaman tsunami

A notable exception to this is the Sumatra-Andaman tsunami, which is known to have had a notably slow rupture that lasted approximately 10 minutes. Moreover the spatial distribution of the slips spans approximately 1000 km, resulting in a potentially complex dynamical formation of the tsunami. While this alone should make it interesting for our investigation, there is also a larger uncertainty in the initial parameters computed by seismic inversion and various source models have been proposed in the literature [138–140]. This illustrates the difficulty



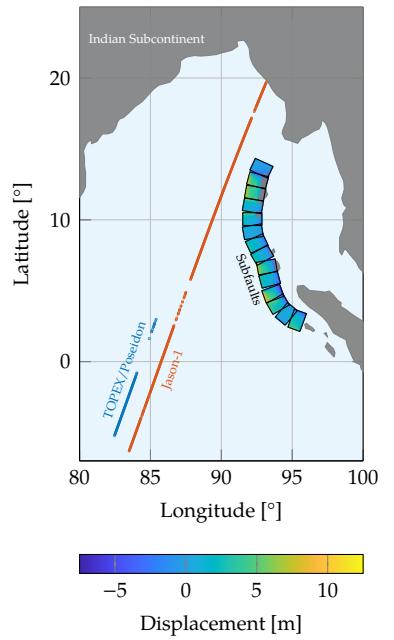
**Figure 13.13:** High-fidelity discontinuous Galerkin simulation of the Sumatra-Andaman tsunami with a locally refined mesh using  $K = 30378$  elements and a polynomial degree of  $p = 4$ .

of determining source parameters using seismic inversion. Using one dataset over another will likely imply that the resulting sources will yield diverging results when compared to one another [140].

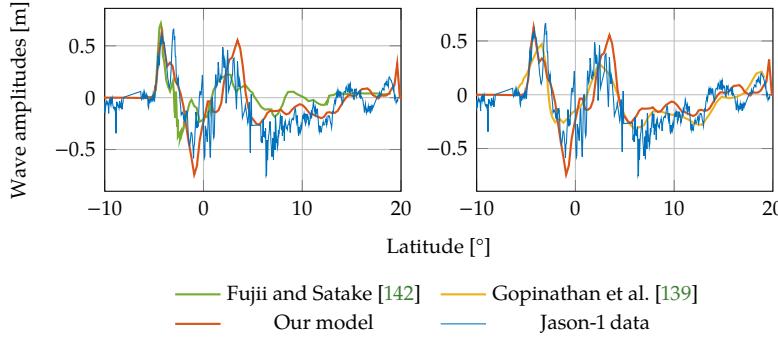
For our purpose, we use fault parameters from [138]. For the dynamic fault model we use parameters as listed in [139]. Figure 13.14 shows the subfaults which are located between the Andaman Islands and Sumatra in the Bay of Bengal. The subfaults are numbered in increasing order from south to west, which coincides with the order in which they are activated. Table 13.3 lists the subfault parameters, complete with their activation times  $t_{0,i}$  and rise times  $t_{1,i} - t_{0,i}$ . We observe the slow nature of the rupture, with more than 580 seconds between the activation of the first subfault and the last one.

To simulate the Sumatra-Andaman tsunami, we use an icosahedral mesh and locally refine a circular area of radius  $\rho = 40^\circ$  around the point at  $84^\circ$  longitude and  $-10^\circ$  latitude. This results in a mesh with a total number of  $K = 30378$  elements, where most of them are located in the Indian Ocean. Figure 13.13 shows the results of the simulation using a polynomial degree of  $p = 4$  and the dynamic source model. The simulation is repeated with a static seabed deformation, in which we impose the final seabed deformation on the sea surface.

As before, we use real-world data as a reference for our results. Because there is little buoy data available, we use sea surface heights reconstructed from satellite data. Here, we use radio altimetry data obtained from the Jason-1 and TOPEX/Poseidon earth observation satellites [141], which passed over the tsunami from south west to north east, as illustrated in



**Figure 13.14:** Location of the subfaults for the Sumatra-Andaman tsunami. The red and blue points illustrate the position of the satellite radio altimetry measurements.



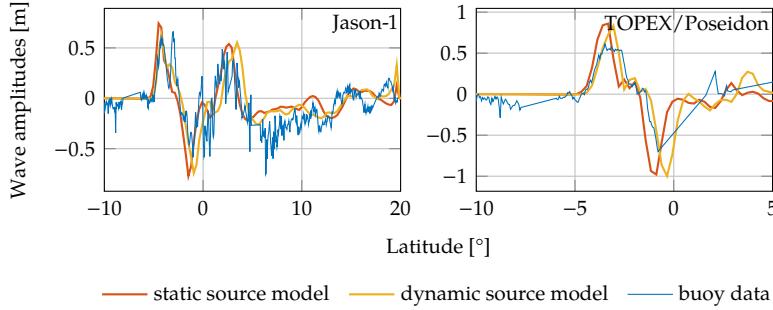
**Figure 13.15:** Comparison of the wave field cross section to satellite measurements and results obtained by others.

Figure 13.14. As such, it represents a cross section of the wave field, rather than a wave signal arriving over time at one point. The results are shown in Figure 13.15 alongside predictions from Fujii and Satake [142], as well as Gopinathan et al. [139]. Fujii and Satake [142] use a finite difference method for the linearized shallow water equations, whereas Gopinathan et al. [139] use a finite volume solver for the non-linear shallow water equations. We observe good agreement of all methods, especially for arrival times and amplitudes of the first wave. In the far-field we observe that the proposed method captures wave amplitudes remarkably well, with a slight error in phase which can likely be attributed to underresolved bathymetry.

To quantify the impact of the dynamic source model, we compare the results to simulation results obtained with a static source. Figure 13.16 depicts a comparison of the aforementioned satellite data to the corresponding cross sections in the wavefield, extracted from our two simulations. Again, for both models, we observe a good match of phase and amplitude of the leading wave. This time however, the dynamic source model is able to better predict phase and amplitude, especially for the leading wave. This is also reflected in the  $L^2$  errors of the aforementioned signals. For the TOPEX/Poseidon data the normalized errors are 0.2775 and 0.1527 for the static and dynamic models respectively. For the Jason-1 data however, the errors for both models are 0.2047 and 0.2048 and therefore practically identical. While the improvement using the dynamic method seems small, we have to keep in mind that the satellite data only provides a one-dimensional extract of the wave-field to assess the accuracy of the results. As such, we conclude that a dynamic source

**Table 13.3:** Subfault parameters and activation times for the simulation of the SumatraAndaman tsunami. Subfaults are ordered according to latitude from south to north.

subfault	longitude	latitude	depth	length	width	strike	dip	rake	slip	$t_{0,i}$	$t_{1,i} - t_{0,i}$
1	95.54	2.13	10	100	150	290	10	71	16.5	0.00	48.5
2	94.5	2.57	10	100	150	310	10	91	0	31.96	48.5
3	93.64	3.33	10	100	150	330	10	104	14.9	136.85	37.5
4	94.5	4.15	10	100	150	340	10	105	29.1	167.62	117.3
5	94.5	5.18	10	100	150	345	10	102	10.4	227.09	138.7
6	94.5	6.12	10	100	150	350	10	100	23.4	254.98	62.1
7	94.5	6.78	10	100	150	330	10	90	9.4	282.41	38.5
8	94.5	7.64	10	100	150	335	10	86	11.5	308.57	111.2
9	94.5	8.60	10	100	150	350	10	99	1.5	334.06	63.9
10	94.5	9.60	10	100	150	0	10	106	12	396.16	51.3
11	94.5	10.66	10	100	150	10	10	115	6.1	433.33	79.0
12	94.5	11.56	10	100	150	10	10	115	25.7	467.73	54.6
13	94.5	12.51	10	100	150	15	10	120	27.2	502.11	76.8
14	94.5	13.51	10	100	150	25	10	130	0	583.82	36.6



**Figure 13.16:** Comparison of wave amplitudes measured by the Jason-1 and TOPEX/Poseidon satellites compared to signals extracted from our simulation using. On the right the signal is zoomed in to show the initial wave.

model can indeed improve the predictions of the model in situations where the rupture process is small.

## 13.4 Concluding remarks

We have presented an efficient discontinuous Galerkin model for modeling geophysical flows on the rotating sphere using the shallow water equations. To do so, we have developed well-balanced methods for wetting/drying as well as grid adaptation. Moreover, we have made a case for using the strong form of the discontinuous Galerkin formulation, due to its favorable properties. Using one-dimensional examples and examples on the sphere, we have verified these methods, proving the properties that we have previously claimed. In addition to this, we have validated the approach through real-world examples in the form of simulations of the 2011 Tohoku tsunami and the 2004 Indian Ocean tsunami. Finally, we performed a comparison of source models to see whether they could offer improvements by incorporating dynamic effects of the rupture process into the existing model.

The developed methods have proven to work well and produce physically accurate results of predictive value. While the dynamic effects of tsunami sources may play an important role for slow ruptures, they can be neglected in most scenarios. It is therefore possible, that other improvements such as non-hydrostatic corrections are of greater benefits for these models.

Our contributions relating well-balancing and wetting/drying offer an interesting choice for discontinuous Galerkin models based on the shallow water equations and potentially other physical systems. This includes the Euler equations where near-vacuum solutions pose similar challenges as drying processes in shallow water systems. To the best of our knowledge, most alternatives for wetting/drying often incorporate unphysical assumptions or are simply not well-balanced. Interesting research avenues are therefore the applicability of these methods to other systems of conservation laws with similar problems.

The tsunami model that we have developed using these ideas demonstrates a range of desirable properties for an early warning system. Not only are the predictions physically accurate but the overall method is also robust, highly adaptable and efficient due to the well-balanced mesh refinement. This allows for automatic generation of grids adapted to the problem, an essential requirement for any early warning system

that needs to function without human intervention. Moreover, we have proven that our mildly optimized and non-parallel implementation can yield physically accurate results faster than real-time, which is another necessary property for any early warning system. Our implementation could easily be accelerated further due to the intrinsic flexibility and parallelity of discontinuous Galerkin methods.

# Bibliography

- [1] Lloyd N. Trefethen and David Bau. *Numerical Linear Algebra*. 1997 (cited on pages ix, 9, 12, 29).
- [2] Boris Bonev et al. 'Discontinuous Galerkin scheme for the spherical shallow water equations with applications to tsunami modeling and prediction'. In: *Journal of Computational Physics* (2018). doi: [10.1016/j.jcp.2018.02.008](https://doi.org/10.1016/j.jcp.2018.02.008) (cited on pages ix, 109, 126).
- [3] Mahya Hajihassanpour, Boris Bonev, and Jan S. Hesthaven. 'A comparative study of earthquake source models in high-order accurate tsunami simulations'. In: *Ocean Modelling* 141. August (Sept. 2019), p. 101429. doi: [10.1016/j.ocemod.2019.101429](https://doi.org/10.1016/j.ocemod.2019.101429) (cited on page ix).
- [4] Boris Bonev and Jan S. Hesthaven. 'A hierarchical preconditioner for wave problems in quasilinear complexity'. In: 513966 (May 2021), pp. 1-33 (cited on page ix).
- [5] Sangmin Kwak, Youngseo Kim, and Changsoo Shin. 'Frequency-domain direct waveform inversion based on perturbation theory'. In: *Geophysical Journal International* 197.2 (May 2014), pp. 987–1001. doi: [10.1093/gji/ggu026](https://doi.org/10.1093/gji/ggu026) (cited on pages 4, 90).
- [6] Claes Johnson. *Numerical Solution of Partial Differential Equations by the Finite Element Method*. New York, NY: Cambridge University Press, 1987 (cited on page 4).
- [7] L C Evans. *Partial Differential Equations*. American Mathematical Society, 1998 (cited on pages 4, 7, 98, 100, 101).
- [8] J T Oden and L Demkowicz. *Applied Functional Analysis, Third Edition*. Textbooks in Mathematics. CRC Press, 2017 (cited on pages 4, 7).
- [9] Alexandre Ern and Jean-Luc Guermond. *Theory and Practice of Finite Elements*. Vol. 159. Applied Mathematical Sciences. New York, NY: Springer New York, 2004, pp. xiv+524 (cited on page 4).
- [10] George B Arfken, Hans J Weber, and Frank E Harris. *Mathematical Methods for Physicists*. Seventh Ed. Elsevier, 2013, pp. 871–933 (cited on page 7).
- [11] Daniel Kressner and Robert Luce. *Computational Linear Algebra*. 2017 (cited on pages 9, 29, 31, 36).
- [12] G H Golub and C F Van Loan. *Matrix Computations*. Johns Hopkins Studies in the Mathematical Sciences. Johns Hopkins University Press, 2013 (cited on pages 9, 11, 12, 16, 70).
- [13] Roger A. Horn and Charles R. Johnson. *Matrix Analysis*. Cambridge University Press, Dec. 1985 (cited on pages 9, 14).
- [14] Steven S. Skiena. *The algorithm design manual: Second edition*. 2008, pp. 1–730 (cited on pages 13, 23).
- [15] Ming Gu and Stanley C. Eisenstat. 'Efficient algorithms for computing a strong rank-revealing QR factorization'. In: *SIAM Journal of Scientific Computing* 17.4 (1996), pp. 848–869. doi: [10.1137/0917055](https://doi.org/10.1137/0917055) (cited on pages 15, 16).
- [16] Tony F. Chan. 'Rank revealing QR factorizations'. In: *Linear Algebra and its Applications* 88-89.C (Apr. 1987), pp. 67–82. doi: [10.1016/0024-3795\(87\)90103-0](https://doi.org/10.1016/0024-3795(87)90103-0) (cited on page 16).
- [17] N. Halko, P.-G. Martinsson, and J. A. Tropp. 'Finding Structure with Randomness: Probabilistic Algorithms for Constructing Approximate Matrix Decompositions'. In: *SIAM Review* 53.2 (Jan. 2011), pp. 217–288. doi: [10.1137/090771806](https://doi.org/10.1137/090771806) (cited on pages 16, 17, 19, 77).
- [18] M.F. Hutchinson. 'A stochastic estimator of the trace of the influence matrix for laplacian smoothing splines'. In: *Communications in Statistics - Simulation and Computation* 19.2 (Jan. 1990), pp. 433–450. doi: [10.1080/03610919008812866](https://doi.org/10.1080/03610919008812866) (cited on page 18).
- [19] I. S. Duff, A. M. Erisman, and J. K. Reid. *Direct Methods for Sparse Matrices*. Oxford University Press, Jan. 2017 (cited on page 23).
- [20] E. Cuthill and J. McKee. 'Reducing the bandwidth of sparse symmetric matrices'. In: *Proceedings of the 1969 24th National Conference, ACM* 1969 (1969), pp. 157–172. doi: [10.1145/800195.805928](https://doi.org/10.1145/800195.805928) (cited on pages 23, 24).

- [21] Alan George. 'Nested Dissection of a Regular Finite Element Mesh'. In: *SIAM Journal on Numerical Analysis* 10.2 (Apr. 1973), pp. 345–363. doi: [10.1137/0710032](https://doi.org/10.1137/0710032) (cited on pages 23, 26).
- [22] Alan J. Hoffman, Michael S. Martin, and Donald J. Rose. 'Complexity Bounds for Regular Finite Difference and Finite Element Grids'. In: *SIAM Journal on Numerical Analysis* 10.2 (Apr. 1973), pp. 364–369. doi: [10.1137/0710033](https://doi.org/10.1137/0710033) (cited on page 23).
- [23] Alan George and Joseph W.H. Liu. 'The Evolution of the Minimum Degree Ordering Algorithm'. In: *SIAM Review* 31.1 (Mar. 1989), pp. 1–19. doi: [10.1137/1031001](https://doi.org/10.1137/1031001) (cited on page 24).
- [24] Joseph W.H. Liu. 'The Multifrontal Method for Sparse Matrix Solution: Theory and Practice'. In: *SIAM Review* 34.1 (May 1992), pp. 82–109 (cited on page 28).
- [25] Per-Gunnar Martinsson. 'Fast Direct Solvers for Elliptic PDEs'. In: Philadelphia, PA: Society for Industrial and Applied Mathematics, Jan. 2019, pp. i–xv. doi: [10.1137/1.9781611976045](https://doi.org/10.1137/1.9781611976045) (cited on pages 28, 37, 38, 40, 41, 44, 52, 54, 68, 86, 93–95).
- [26] Yousef Saad. 'Iterative Methods for Sparse Linear Systems, Second Edition'. In: *Methods* (2003), p. 528 (cited on pages 31, 36).
- [27] Y. A. Erlangga, C. W. Oosterlee, and C. Vuik. 'A novel multigrid based preconditioner for heterogeneous Helmholtz problems'. In: *SIAM Journal on Scientific Computing* 27.4 (2006), pp. 1471–1492. doi: [10.1137/040615195](https://doi.org/10.1137/040615195) (cited on page 35).
- [28] C. W. Oosterlee et al. 'Shifted-Laplacian Preconditioners for Heterogeneous Helmholtz Problems'. In: *Lecture Notes in Electrical Engineering*. Vol. 71 LNCSE. 2009, pp. 21–46. doi: [10.1007/978-3-642-03344-5\\_2](https://doi.org/10.1007/978-3-642-03344-5_2) (cited on page 35).
- [29] G.P. Astrakhantsev. 'An iterative method of solving elliptic net problems'. In: *USSR Computational Mathematics and Mathematical Physics* 11.2 (Jan. 1971), pp. 171–182. doi: [10.1016/0041-5553\(71\)90170-4](https://doi.org/10.1016/0041-5553(71)90170-4) (cited on page 36).
- [30] Wolfgang Hackbusch. *Hierarchical Matrices: Algorithms and Analysis*. Vol. 49. Springer Series in Computational Mathematics. Berlin, Heidelberg: Springer Berlin Heidelberg, 2015, pp. 53–82 (cited on pages 37–40, 44, 46).
- [31] V. Rokhlin. 'Rapid solution of integral equations of classical potential theory'. In: *Journal of Computational Physics* 60.2 (Sept. 1985), pp. 187–207. doi: [10.1016/0021-9991\(85\)90002-6](https://doi.org/10.1016/0021-9991(85)90002-6) (cited on pages 41, 42, 54).
- [32] L. Greengard and V. Rokhlin. 'A fast algorithm for particle simulations'. In: *Journal of Computational Physics* 135.2 (1997), pp. 280–292. doi: [10.1006/jcph.1997.5706](https://doi.org/10.1006/jcph.1997.5706) (cited on pages 41, 66).
- [33] David J. Griffiths. *Introduction to Electrodynamics*. Cambridge University Press, June 2017 (cited on page 41).
- [34] D.E. Winch and P.H. Roberts. 'Derivatives of addition theorems for Legendre functions'. In: *The Journal of the Australian Mathematical Society. Series B. Applied Mathematics* 37.2 (Oct. 1995), pp. 212–234. doi: [10.1017/S0334270000007670](https://doi.org/10.1017/S0334270000007670) (cited on page 42).
- [35] S. Chandrasekaran et al. 'A fast solver for HSS representations via sparse matrices'. In: *SIAM Journal on Matrix Analysis and Applications* 29.1 (2006), pp. 67–81. doi: [10.1137/050639028](https://doi.org/10.1137/050639028) (cited on page 43).
- [36] Sia Amini and Anthony Profit. 'Analysis of the truncation errors in the fast multipole method for scattering problems'. In: *Journal of Computational and Applied Mathematics* 115.1-2 (Mar. 2000), pp. 23–33. doi: [10.1016/S0377-0427\(99\)00175-2](https://doi.org/10.1016/S0377-0427(99)00175-2) (cited on page 43).
- [37] Steffen Börm, Lars Grasedyck, and Wolfgang Hackbusch. 'Introduction to hierarchical matrices with applications'. In: *Engineering Analysis with Boundary Elements* 27.5 (May 2003), pp. 405–422. doi: [10.1016/S0955-7997\(02\)00152-2](https://doi.org/10.1016/S0955-7997(02)00152-2) (cited on pages 44, 46).
- [38] Steffen Börm. *Hierarchical Matrices*. Vol. 63. Lecture Notes in Computational Science and Engineering. Berlin, Heidelberg: Springer Berlin Heidelberg, 2008 (cited on page 44).
- [39] Patrick R. Amestoy et al. 'Bridging the Gap Between Flat and Hierarchical Low-Rank Matrix Formats: The Multilevel Block Low-Rank Format'. In: *SIAM Journal on Scientific Computing* 41.3 (Jan. 2019), A1414–A1442. doi: [10.1137/18M1182760](https://doi.org/10.1137/18M1182760) (cited on page 46).

- [40] Steffen Börm. *Efficient Numerical Methods for Non-local Operators*. Zuerich, Switzerland: European Mathematical Society Publishing House, Dec. 2010 (cited on pages 46, 47, 57, 68, 70).
- [41] Raf Vandebril, Marc Van Barel and Nicola Mastronardi. *Matrix Computations and Semiseparable Matrices - Linear Systems*. 1st ed. Johns Hopkins University Press, 2008, p. 584 (cited on pages 48, 49).
- [42] James Vogel et al. ‘Superfast Divide-and-Conquer Method and Perturbation Analysis for Structured Eigenvalue Solutions’. In: *SIAM Journal on Scientific Computing* 38.3 (Jan. 2016), A1358–A1382. doi: [10.1137/15M1018812](https://doi.org/10.1137/15M1018812) (cited on pages 49, 62).
- [43] Yuanzhe Xi et al. ‘Superfast and Stable Structured Solvers for Toeplitz Least Squares via Randomized Sampling’. In: *SIAM Journal on Matrix Analysis and Applications* 35.1 (Jan. 2014), pp. 44–72. doi: [10.1137/120895755](https://doi.org/10.1137/120895755) (cited on page 49).
- [44] Daniel Kressner, Stefano Massei, and Leonardo Robol. ‘Low-Rank Updates and a Divide-And-Conquer Method for Linear Matrix Equations’. In: *SIAM Journal on Scientific Computing* 41.2 (Jan. 2019), A848–A876. doi: [10.1137/17M1161038](https://doi.org/10.1137/17M1161038) (cited on page 50).
- [45] Stefano Massei, Leonardo Robol, and Daniel Kressner. *hm-toolbox: MATLAB SOFTWARE FOR HODLR AND HSS MATRICES*. Tech. rep. (cited on pages 53, 55, 57, 60, 62, 75, 95).
- [46] Boris Bonev. ‘HssMatrices.jl: A Julia package for hierarchically semi-separable matrices’. In: (Apr. 2021). doi: [10.5281/ZENODO.4696465](https://doi.org/10.5281/ZENODO.4696465) (cited on pages 54, 57, 59, 60, 63, 65).
- [47] S Chandrasekaran, M Gu, and T Pals. ‘A Fast ULV Decomposition Solver for Hierarchically Semiseparable Representations’. In: *SIAM Journal on Matrix Analysis and Applications* 28.3 (Jan. 2006), pp. 603–622. doi: [10.1137/S0895479803436652](https://doi.org/10.1137/S0895479803436652) (cited on pages 54, 55, 75).
- [48] Difeng Cai et al. ‘SMASH: Structured matrix approximation by separation and hierarchy’. In: *arXiv* (2017) (cited on page 57).
- [49] Jianlin Xia et al. ‘Fast algorithms for hierarchically semiseparable matrices’. In: *Numerical Linear Algebra with Applications* 17.6 (Dec. 2010), pp. 953–976. doi: [10.1002/nla.691](https://doi.org/10.1002/nla.691) (cited on pages 57, 60).
- [50] Per-Gunnar Martinsson. ‘A Fast Randomized Algorithm for Computing a Hierarchically Semiseparable Representation of a Matrix’. In: *SIAM Journal on Matrix Analysis and Applications* 32.4 (Oct. 2011), pp. 1251–1274. doi: [10.1137/100786617](https://doi.org/10.1137/100786617) (cited on pages 60, 61).
- [51] Xiao Liu, Jianlin Xia, and Maarten V. De Hoop. ‘Parallel randomized and matrix-free direct solvers for large structured dense linear systems’. In: *SIAM Journal on Scientific Computing* 38.5 (2016), S508–S538. doi: [10.1137/15M1023774](https://doi.org/10.1137/15M1023774) (cited on pages 62, 66).
- [52] Christopher Gorman et al. ‘Robust and Accurate Stopping Criteria for Adaptive Randomized Sampling in Matrix-Free Hierarchically Semiseparable Construction’. In: *SIAM Journal on Scientific Computing* 41.5 (Jan. 2019), S61–S85. doi: [10.1137/18M1194961](https://doi.org/10.1137/18M1194961) (cited on page 62).
- [53] Lin Lin, Jianfeng Lu, and Lexing Ying. ‘Fast construction of hierarchical matrix representation from matrixvector multiplication’. In: *Journal of Computational Physics* 230.10 (May 2011), pp. 4071–4087. doi: [10.1016/j.jcp.2011.02.033](https://doi.org/10.1016/j.jcp.2011.02.033) (cited on page 62).
- [54] Mario Bebendorf and Wolfgang Hackbusch. ‘Existence of H-matrix approximants to the inverse FE-matrix of elliptic operators with L-coefficients’. In: *Numerische Mathematik* 95.1 (July 2003), pp. 1–28. doi: [10.1007/s00211-002-0445-6](https://doi.org/10.1007/s00211-002-0445-6) (cited on pages 66, 76).
- [55] Mario Bebendorf. ‘Efficient inversion of the Galerkin matrix of general second-order elliptic operators with nonsmooth coefficients’. In: *Mathematics of Computation* 74.251 (Sept. 2004), pp. 1179–1200. doi: [10.1090/s0025-5718-04-01716-8](https://doi.org/10.1090/s0025-5718-04-01716-8) (cited on pages 66, 76).
- [56] Mario Bebendorf. ‘Why finite element discretizations can be factored by triangular hierarchical matrices’. In: *SIAM Journal on Numerical Analysis* 45.4 (2007), pp. 1472–1494. doi: [10.1137/060669747](https://doi.org/10.1137/060669747) (cited on pages 66, 76).
- [57] Per-Gunnar Martinsson. ‘A Fast direct solver for a class of elliptic partial differential equations’. In: *Journal of Scientific Computing* 38.3 (2009), pp. 316–330. doi: [10.1007/s10915-008-9240-6](https://doi.org/10.1007/s10915-008-9240-6) (cited on pages 66, 70).

- [58] S Chandrasekaran et al. ‘On the Numerical Rank of the Off-Diagonal Blocks of Schur Complements of Discretized Elliptic PDEs’. In: *SIAM Journal on Matrix Analysis and Applications* 31.5 (Jan. 2010), pp. 2261–2290. doi: [10.1137/090775932](https://doi.org/10.1137/090775932) (cited on page 66).
- [59] Björn Engquist and Hongkai Zhao. ‘Approximate Separability of the Green’s Function of the Helmholtz Equation in the High Frequency Limit’. In: *Communications on Pure and Applied Mathematics* 71.11 (Nov. 2018), pp. 2220–2274. doi: [10.1002/cpa.21755](https://doi.org/10.1002/cpa.21755) (cited on pages 66, 89).
- [60] Jianlin Xia et al. ‘Superfast Multifrontal Method for Large Structured Linear Systems of Equations’. In: *SIAM Journal on Matrix Analysis and Applications* 31.3 (Jan. 2010), pp. 1382–1411. doi: [10.1137/09074543X](https://doi.org/10.1137/09074543X) (cited on pages 66, 69).
- [61] Phillip G. Schmitz and Lexing Ying. ‘A fast direct solver for elliptic problems on general meshes in 2D’. In: *Journal of Computational Physics* 231.4 (2012), pp. 1314–1338. doi: [10.1016/j.jcp.2011.10.013](https://doi.org/10.1016/j.jcp.2011.10.013) (cited on pages 66, 69).
- [62] Jianlin Xia. ‘Randomized Sparse Direct Solvers’. In: *SIAM Journal on Matrix Analysis and Applications* 34.1 (Jan. 2013), pp. 197–227. doi: [10.1137/12087116X](https://doi.org/10.1137/12087116X) (cited on pages 66, 93).
- [63] Jianlin Xia. ‘Efficient Structured Multifrontal Factorization for General Large Sparse Matrices’. In: *SIAM Journal on Scientific Computing* 35.2 (Jan. 2013), A832–A860. doi: [10.1137/120867032](https://doi.org/10.1137/120867032) (cited on pages 66, 86).
- [64] A. Gillman and P.-G. Martinsson. ‘A Direct Solver with  $O(N)$  Complexity for Variable Coefficient Elliptic PDEs Discretized via a High-Order Composite Spectral Collocation Method’. In: *SIAM Journal on Scientific Computing* 36.4 (Jan. 2014), A2023–A2046. doi: [10.1137/130918988](https://doi.org/10.1137/130918988) (cited on page 66).
- [65] Shen Wang et al. ‘A Parallel Geometric Multifrontal Solver Using Hierarchically Semiseparable Structure’. In: *ACM Transactions on Mathematical Software* 42.3 (June 2016), pp. 1–21. doi: [10.1145/2830569](https://doi.org/10.1145/2830569) (cited on pages 66, 95).
- [66] Pieter Ghysels et al. ‘An Efficient Multicore Implementation of a Novel HSS-Structured Multifrontal Solver Using Randomized Sampling’. In: *SIAM Journal on Scientific Computing* 38.5 (Jan. 2016), S358–S384. doi: [10.1137/15M1010117](https://doi.org/10.1137/15M1010117) (cited on pages 66, 70).
- [67] P. Gatto and J. S. Hesthaven. ‘Efficient Preconditioning of hp-FEM Matrices by Hierarchical Low-Rank Approximations’. In: *Journal of Scientific Computing* 72.1 (July 2017), pp. 49–80. doi: [10.1007/s10915-016-0347-x](https://doi.org/10.1007/s10915-016-0347-x) (cited on pages 66, 70).
- [68] Kai Yang, Hadi Pouransari, and Eric Darve. ‘Sparse hierarchical solvers with guaranteed convergence’. In: *International Journal for Numerical Methods in Engineering* 120.8 (2019), pp. 964–986. doi: [10.1002/nme.6166](https://doi.org/10.1002/nme.6166) (cited on pages 66, 76).
- [69] D. E. Keyes, H. Ltaief, and G. Turkiyyah. ‘Hierarchical algorithms on hierarchical architectures’. In: *Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences* 378.2166 (Mar. 2020), p. 20190055. doi: [10.1098/rsta.2019.0055](https://doi.org/10.1098/rsta.2019.0055) (cited on page 66).
- [70] Miroslav Fiedler. ‘Structure ranks of matrices’. In: *Linear Algebra and its Applications* 179.C (Jan. 1993), pp. 119–127. doi: [10.1016/0024-3795\(93\)90324-H](https://doi.org/10.1016/0024-3795(93)90324-H) (cited on page 66).
- [71] Fuzhen Zhang, ed. *The Schur Complement and Its Applications*. Vol. 4. Numerical Methods and Algorithms. New York: Springer-Verlag, 2005 (cited on page 67).
- [72] M. Bebendorf. ‘Hierarchical LU Decomposition-based Preconditioners for BEM’. In: *Computing* 74.3 (May 2005), pp. 225–247. doi: [10.1007/s00607-004-0099-6](https://doi.org/10.1007/s00607-004-0099-6) (cited on pages 68, 70).
- [73] Adrianna Gillman, Patrick M. Young, and Per-Gunnar Martinsson. ‘A direct solver with  $O(N)$  complexity for integral equations on one-dimensional domains’. In: *Frontiers of Mathematics in China* 7.2 (Apr. 2012), pp. 217–247. doi: [10.1007/s11464-012-0188-3](https://doi.org/10.1007/s11464-012-0188-3) (cited on pages 68, 70).
- [74] Adrianna Gillman and Per-Gunnar Martinsson. ‘An  $O(N)$  algorithm for constructing the solution operator to 2D elliptic boundary value problems in the absence of body loads’. In: *Advances in Computational Mathematics* 40.4 (Aug. 2014), pp. 773–796. doi: [10.1007/s10444-013-9326-z](https://doi.org/10.1007/s10444-013-9326-z) (cited on pages 68, 70).

- [75] Jianlin Xia. 'Efficient Structured Multifrontal Factorization for General Large Sparse Matrices'. In: *SIAM Journal on Scientific Computing* 35.2 (Jan. 2013), A832–A860. doi: [10.1137/120867032](https://doi.org/10.1137/120867032) (cited on pages 69, 70).
- [76] AmirHossein Aminfar, Sivaram Ambikasaran, and Eric Darve. 'A fast block low-rank dense solver with applications to finite-element matrices'. In: *Journal of Computational Physics* 304 (Jan. 2016), pp. 170–188. doi: [10.1016/j.jcp.2015.10.012](https://doi.org/10.1016/j.jcp.2015.10.012) (cited on page 70).
- [77] Hadi Pouransari, Pieter Coulier, and Eric Darve. 'Fast Hierarchical Solvers For Sparse Matrices Using Extended Sparsification and Low-Rank Approximation'. In: *SIAM Journal on Scientific Computing* 39.3 (Jan. 2017), A797–A830. doi: [10.1137/15M1046939](https://doi.org/10.1137/15M1046939) (cited on page 76).
- [78] Jan S. Hesthaven and Tim Warburton. *Nodal Discontinuous Galerkin Methods*. Vol. 54. Texts in Applied Mathematics 2. New York, NY: Springer New York, 2008, pp. 127–41 (cited on pages 82, 98, 107, 109, 131, 137, 143).
- [79] Frank Ihlenburg and I. Babuka. 'Finite element solution of the Helmholtz equation with high wave number Part I: The h-version of the FEM'. In: *Computers & Mathematics with Applications* 30.9 (Nov. 1995), pp. 9–37. doi: [10.1016/0898-1221\(95\)00144-N](https://doi.org/10.1016/0898-1221(95)00144-N) (cited on page 84).
- [80] Ivo M. Babuka and Stefan A Sauter. 'Is the Pollution Effect of the FEM Avoidable for the Helmholtz Equation Considering High Wave Numbers?' In: *SIAM Journal on Numerical Analysis* 34.6 (Dec. 1997), pp. 2392–2423. doi: [10.1137/S0036142994269186](https://doi.org/10.1137/S0036142994269186) (cited on page 84).
- [81] Timo. Lähivaara. 'Discontinuous Galerkin Method for Time-domain Wave Problems'. PhD thesis. 2010, pp. 1–79 (cited on page 84).
- [82] M. J. Gander, I. G. Graham, and E. A. Spence. 'Applying GMRES to the Helmholtz equation with shifted Laplacian preconditioning: what is the largest shift for which wavenumber-independent convergence is guaranteed?' In: *Numerische Mathematik* 131.3 (2015), pp. 567–614. doi: [10.1007/s00211-015-0700-2](https://doi.org/10.1007/s00211-015-0700-2) (cited on page 88).
- [83] Clint Dawson, Shuyu Sun, and Mary F. Wheeler. 'Compatible algorithms for coupled flow and transport'. In: *Computer Methods in Applied Mechanics and Engineering* 193.23–26 (2004), pp. 2565–2580. doi: [10.1016/j.cma.2003.12.059](https://doi.org/10.1016/j.cma.2003.12.059) (cited on page 89).
- [84] Yohann Dudouit. 'Spatio-temporal refinement using a discontinuous Galerkin approach for elasto-dynamic in a high performance computing framework'. In: *Université De Bordeaux* (2014) (cited on page 89).
- [85] Gary S. Martin, Robert Wiley, and Kurt J. Marfurt. 'Marmousi2: An elastic upgrade for Marmousi'. In: *The Leading Edge* 25.2 (Feb. 2006), pp. 156–166. doi: [10.1190/1.2172306](https://doi.org/10.1190/1.2172306) (cited on page 90).
- [86] Randall J. LeVeque. *Finite Difference Methods for Ordinary and Partial Differential Equations*. Society for Industrial and Applied Mathematics, Jan. 2007 (cited on page 98).
- [87] Randall J. LeVeque. *Numerical Methods for Conservation Laws*. Basel: Birkhäuser Basel, 1992 (cited on pages 98, 100, 101, 105, 106).
- [88] Siddhartha Mishra, Ulrie Fjordholm, and Remi Abgrall. *Numerical methods for conservation laws and related equations* (cited on pages 98, 105).
- [89] Jan S. Hesthaven. *Numerical Methods for Conservation Laws*. Philadelphia, PA: Society for Industrial and Applied Mathematics, Dec. 2018 (cited on pages 98, 106, 109).
- [90] Francis X Giraldo. *An Introduction to Galerkin Methods on Tensor-Product Bases*. Vol. 24. 2020 (cited on pages 98, 106, 109).
- [91] F.X. Giraldo, J.S. Hesthaven, and T. Warburton. 'Nodal High-Order Discontinuous Galerkin Methods for the Spherical Shallow Water Equations'. In: *Journal of Computational Physics* 181.2 (Sept. 2002), pp. 499–525. doi: [10.1006/jcph.2002.7139](https://doi.org/10.1006/jcph.2002.7139) (cited on pages 98, 101, 109, 113, 139).
- [92] Alfredo Bermudez and Ma Elena Vazquez. 'Upwind methods for hyperbolic conservation laws with source terms'. In: *Computers & Fluids* 23.8 (Nov. 1994), pp. 1049–1071. doi: [10.1016/0045-7930\(94\)90004-3](https://doi.org/10.1016/0045-7930(94)90004-3) (cited on pages 99, 118).

- [93] Emmanuel Audusse et al. 'A Fast and Stable Well-Balanced Scheme with Hydrostatic Reconstruction for Shallow Water Flows'. In: *SIAM Journal on Scientific Computing* 25.6 (Jan. 2004), pp. 2050–2065. doi: [10.1137/S1064827503431090](https://doi.org/10.1137/S1064827503431090) (cited on pages 99, 118, 120, 123).
- [94] Sebastian Noelle et al. 'Well-balanced finite volume schemes of arbitrary order of accuracy for shallow water flows'. In: *Journal of Computational Physics* 213.2 (Apr. 2006), pp. 474–499. doi: [10.1016/j.jcp.2005.08.019](https://doi.org/10.1016/j.jcp.2005.08.019) (cited on page 99).
- [95] Geoffrey K. Vallis. *Atmospheric and Oceanic Fluid Dynamics*. Cambridge: Cambridge University Press, 2017, pp. 1–946 (cited on page 99).
- [96] Michael Baldauf. 'Discontinuous Galerkin solver for the shallow-water equations in covariant form on the sphere and the ellipsoid'. In: *Journal of Computational Physics* 410 (June 2020), p. 109384. doi: [10.1016/j.jcp.2020.109384](https://doi.org/10.1016/j.jcp.2020.109384) (cited on page 101).
- [97] Sergei Godunov and I Bohachevsky. 'Finite difference method for numerical computation of discontinuous solutions of the equations of fluid dynamics'. In: *Matematicheskij sbornik* 47.3 (1959), pp. 271–306 (cited on page 104).
- [98] Philippe G. LeFloch and Mai Duc Thanh. 'The Riemann problem for the shallow water equations with discontinuous topography'. In: *Communications in Mathematical Sciences* 5.4 (2007), pp. 865–885. doi: [10.4310/CMS.2007.v5.n4.a7](https://doi.org/10.4310/CMS.2007.v5.n4.a7) (cited on pages 104, 136).
- [99] Bernardo Cockburn and Chi Wang Shu. 'Runge-Kutta Discontinuous Galerkin methods for convection-dominated problems'. In: *Journal of Scientific Computing* 16.3 (2001), pp. 173–261. doi: [10.1023/A:1012873910884](https://doi.org/10.1023/A:1012873910884) (cited on page 106).
- [100] Alfio Quarteroni, Riccardo Sacco, and Fausto Saleri. *Numerical Mathematics*. Vol. 37. Texts in Applied Mathematics. New York, NY: Springer New York, 2007, p. 655 (cited on pages 109, 111, 112).
- [101] Ramachandran D. Nair, Stephen J. Thomas, and Richard D. Loft. 'A Discontinuous Galerkin Global Shallow Water Model'. In: *Monthly Weather Review* 133.4 (Apr. 2005), pp. 876–888. doi: [10.1175/MWR2903.1](https://doi.org/10.1175/MWR2903.1) (cited on page 113).
- [102] Günther Zängl et al. 'The ICON (ICOahedral Non-hydrostatic) modelling framework of DWD and MPI-M: Description of the non-hydrostatic dynamical core'. In: *Quarterly Journal of the Royal Meteorological Society* 141.687 (Jan. 2015), pp. 563–579. doi: [10.1002/qj.2378](https://doi.org/10.1002/qj.2378) (cited on page 113).
- [103] John Thuburn. 'A PV-Based Shallow-Water Model on a HexagonalIcosahedral Grid'. In: *Monthly Weather Review* 125.9 (Sept. 1997), pp. 2328–2347. doi: [10.1175/1520-0493\(1997\)125<2328:APBSWM>2.0.CO;2](https://doi.org/10.1175/1520-0493(1997)125<2328:APBSWM>2.0.CO;2) (cited on page 113).
- [104] Michal A. Kopera and Francis X. Giraldo. 'Mass conservation of the unified continuous and discontinuous element-based Galerkin methods on dynamically adaptive grids with application to atmospheric simulations'. In: *Journal of Computational Physics* 297 (Sept. 2015), pp. 90–103. doi: [10.1016/j.jcp.2015.05.010](https://doi.org/10.1016/j.jcp.2015.05.010) (cited on pages 113, 115).
- [105] Michal A. Kopera and Francis X. Giraldo. 'Analysis of adaptive mesh refinement for IMEX discontinuous Galerkin solutions of the compressible Euler equations with application to atmospheric simulations'. In: *Journal of Computational Physics* 275 (Oct. 2014), pp. 92–117. doi: [10.1016/j.jcp.2014.06.026](https://doi.org/10.1016/j.jcp.2014.06.026) (cited on pages 113, 115).
- [106] David A. Kopriva, Stephen L. Woodruff, and M. Y. Hussaini. 'Computation of electromagnetic scattering with a non-conforming discontinuous spectral element method'. In: *International Journal for Numerical Methods in Engineering* 53.1 (Jan. 2002), pp. 105–122. doi: [10.1002/nme.394](https://doi.org/10.1002/nme.394) (cited on page 113).
- [107] Cuneyt Sert and Ali Beskok. 'Spectral element formulations on non-conforming grids: A comparative study of pointwise matching and integral projection methods'. In: *Journal of Computational Physics* 211.1 (2006), pp. 300–325. doi: [10.1016/j.jcp.2005.05.019](https://doi.org/10.1016/j.jcp.2005.05.019) (cited on page 115).
- [108] S. Marras, M. A. Kopera, and F. X. Giraldo. 'Simulation of shallowwater jets with a unified elementbased continuous/discontinuous Galerkin model with grid flexibility on the sphere'. In: *Quarterly Journal of the Royal Meteorological Society* 141.690 (July 2015), pp. 1727–1739. doi: [10.1002/qj.2474](https://doi.org/10.1002/qj.2474) (cited on page 115).

- [109] J. C. Butcher. *Numerical Methods for Ordinary Differential Equations*. Chichester, UK: John Wiley & Sons, Ltd, Mar. 2008 (cited on pages 115, 116).
- [110] John Strikwerda. *Finite Difference Schemes and Partial Differential Equations*. Society for Industrial and Applied Mathematics, Jan. 2004 (cited on page 116).
- [111] Sigal Gottlieb, David Ketcheson, and Chi-Wang Shu. *Strong Stability Preserving Runge-Kutta and Multistep Time Discretizations*. WORLD SCIENTIFIC, Jan. 2011 (cited on pages 116, 117, 128).
- [112] Chi-Wang Shu and Stanley Osher. ‘Efficient implementation of essentially non-oscillatory shock-capturing schemes’. In: *Journal of Computational Physics* 77.2 (Aug. 1988), pp. 439–471. doi: [10.1016/0021-9991\(88\)90177-5](https://doi.org/10.1016/0021-9991(88)90177-5) (cited on page 117).
- [113] Ethan J. Kubatko, Benjamin A. Yeager, and David I. Ketcheson. ‘Optimal Strong-Stability-Preserving RungeKutta Time Discretizations for Discontinuous Galerkin Methods’. In: *Journal of Scientific Computing* 60.2 (Aug. 2014), pp. 313–344. doi: [10.1007/s10915-013-9796-7](https://doi.org/10.1007/s10915-013-9796-7) (cited on page 117).
- [114] J. M. Greenberg and A. Y. Leroux. ‘A Well-Balanced Scheme for the Numerical Processing of Source Terms in Hyperbolic Equations’. In: *SIAM Journal on Numerical Analysis* 33.1 (Feb. 1996), pp. 1–16. doi: [10.1137/0733001](https://doi.org/10.1137/0733001) (cited on page 118).
- [115] David A. Kopriva. ‘Metric Identities and the Discontinuous Spectral Element Method on Curvilinear Meshes’. In: *Journal of Scientific Computing* 26.3 (Mar. 2006), pp. 301–327. doi: [10.1007/s10915-005-9070-8](https://doi.org/10.1007/s10915-005-9070-8) (cited on page 121).
- [116] Yulong Xing, Xiangxiong Zhang, and Chi-Wang Shu. ‘Positivity-preserving high order well-balanced discontinuous Galerkin methods for the shallow water equations’. In: *Advances in Water Resources* 33.12 (Dec. 2010), pp. 1476–1493. doi: [10.1016/j.advwatres.2010.08.005](https://doi.org/10.1016/j.advwatres.2010.08.005) (cited on pages 122, 123, 125, 126, 128, 134).
- [117] Yulong Xing and Chi-Wang Shu. ‘High order well-balanced finite volume WENO schemes and discontinuous Galerkin methods for a class of hyperbolic systems with source terms’. In: *Journal of Computational Physics* 214.2 (May 2006), pp. 567–598. doi: [10.1016/j.jcp.2005.10.005](https://doi.org/10.1016/j.jcp.2005.10.005) (cited on page 122).
- [118] Stefan Vater, Nicole Beisiegel, and Jörn Behrens. ‘A limiter-based well-balanced discontinuous Galerkin method for shallow-water flows with wetting and drying: One-dimensional case’. In: *Advances in Water Resources* 85 (Nov. 2015), pp. 1–13. doi: [10.1016/j.advwatres.2015.08.008](https://doi.org/10.1016/j.advwatres.2015.08.008) (cited on pages 122, 126).
- [119] A. Meister and S. Ortleb. ‘On unconditionally positive implicit time integration for the DG scheme applied to shallow water flows’. In: *International Journal for Numerical Methods in Fluids* 76.2 (Sept. 2014), pp. 69–94. doi: [10.1002/fld.3921](https://doi.org/10.1002/fld.3921) (cited on pages 122, 125, 126).
- [120] Georges Kesserwani et al. ‘Well-balancing issues related to the RKDG2 scheme for the shallow water equations’. In: *International Journal for Numerical Methods in Fluids* (2010). doi: [10.1002/fld.2027](https://doi.org/10.1002/fld.2027) (cited on pages 122, 126).
- [121] Yulong Xing and Xiangxiong Zhang. ‘Positivity-Preserving Well-Balanced Discontinuous Galerkin Methods for the Shallow Water Equations on Unstructured Triangular Meshes’. In: *Journal of Scientific Computing* 57.1 (Oct. 2013), pp. 19–41. doi: [10.1007/s10915-013-9695-y](https://doi.org/10.1007/s10915-013-9695-y) (cited on pages 123, 126).
- [122] Onno Bokhove. ‘Flooding and Drying in Discontinuous Galerkin Finite-Element Discretizations of Shallow-Water Equations. Part 1: One Dimension’. In: *Journal of Scientific Computing* 22-23.1-3 (June 2005), pp. 47–82. doi: [10.1007/s10915-004-4136-6](https://doi.org/10.1007/s10915-004-4136-6) (cited on page 125).
- [123] A. Ern, S. Piperno, and K. Djadeli. ‘A well-balanced Runge-Kutta discontinuous Galerkin method for the shallow-water equations with flooding and drying’. In: *International Journal for Numerical Methods in Fluids* 58.1 (Sept. 2008), pp. 1–25. doi: [10.1002/fld.1674](https://doi.org/10.1002/fld.1674) (cited on page 125).
- [124] Niklas Wintermeyer et al. ‘An entropy stable discontinuous Galerkin method for the shallow water equations on curvilinear meshes with wet/dry fronts accelerated by GPUs’. In: *Journal of Computational Physics* 375 (2018). doi: [10.1016/j.jcp.2018.08.038](https://doi.org/10.1016/j.jcp.2018.08.038) (cited on page 126).

- [125] Shintaro Bunya et al. ‘A wetting and drying treatment for the RungeKutta discontinuous Galerkin solution to the shallow water equations’. In: *Computer Methods in Applied Mechanics and Engineering* 198.17-20 (Apr. 2009), pp. 1548–1562. doi: [10.1016/j.cma.2009.01.008](https://doi.org/10.1016/j.cma.2009.01.008) (cited on page 126).
- [126] Bas van’t Hof and E. A. H. Vollebregt. ‘Modelling of wetting and drying of shallow water using artificial porosity’. In: *International Journal for Numerical Methods in Fluids* 48.11 (Aug. 2005), pp. 1199–1217. doi: [10.1002/fld.959](https://doi.org/10.1002/fld.959) (cited on page 126).
- [127] Tuomas Kärnä et al. ‘A fully implicit wettingdrying method for DG-FEM shallow water models, with an application to the Scheldt Estuary’. In: *Computer Methods in Applied Mechanics and Engineering* 200.5-8 (Jan. 2011), pp. 509–524. doi: [10.1016/j.cma.2010.07.001](https://doi.org/10.1016/j.cma.2010.07.001) (cited on page 126).
- [128] Olivier Delestre et al. ‘SWASHES: a compilation of shallow water analytic solutions for hydraulic and environmental studies’. In: *International Journal for Numerical Methods in Fluids* 72.3 (May 2013), pp. 269–300. doi: [10.1002/fld.3741](https://doi.org/10.1002/fld.3741) (cited on page 136).
- [129] William Carlisle Thacker. ‘Some exact solutions to the nonlinear shallow-water wave equations’. In: *Journal of Fluid Mechanics* 107 (June 1981), p. 499. doi: [10.1017/S0022112081001882](https://doi.org/10.1017/S0022112081001882) (cited on page 137).
- [130] NOAA National Geophysical Data Center. *ETOPO1 1 Arc-Minute Global Relief Model*. 2009. (Visited on 06/14/2021) (cited on page 138).
- [131] Y. Okada. ‘Surface deformation due to shear and tensile faults in a half space’. In: *Bulletin - Seismological Society of America* 75.4 (1985), pp. 1135–1154 (cited on pages 141, 144).
- [132] Guangfu Shao et al. ‘Focal mechanism and slip history of the 2011 M w 9.1 off the Pacific coast of Tohoku Earthquake, constrained with teleseismic body and surface waves’. In: *Earth, Planets and Space* 63.7 (July 2011), pp. 559–564. doi: [10.5047/eps.2011.06.028](https://doi.org/10.5047/eps.2011.06.028) (cited on pages 141, 144, 145).
- [133] National Oceanic and Atmospheric Administration (2005). *Deep-Ocean Assessment and Reporting of Tsunamis (DART(R))*. National Centers for Environmental Information, NOAA. Sept. 2017 (cited on page 142).
- [134] D. Wirasaet et al. ‘Discontinuous Galerkin methods with nodal and hybrid modal/nodal triangular, quadrilateral, and polygonal elements for nonlinear shallow water flow’. In: *Computer Methods in Applied Mechanics and Engineering* 270 (Mar. 2014), pp. 113–149. doi: [10.1016/j.cma.2013.11.006](https://doi.org/10.1016/j.cma.2013.11.006) (cited on page 143).
- [135] Sébastien Blaise et al. ‘Discontinuous Galerkin unsteady discrete adjoint method for real-time efficient tsunami simulations’. In: *Journal of Computational Physics* 232.1 (2013), pp. 416–430. doi: [10.1016/j.jcp.2012.08.022](https://doi.org/10.1016/j.jcp.2012.08.022) (cited on page 144).
- [136] Denys Dutykh and Frédéric Dias. ‘Water waves generated by a moving bottom’. In: *Tsunami and Nonlinear Waves*. Berlin, Heidelberg: Springer Berlin Heidelberg, pp. 65–95. doi: [10.1007/978-3-540-71256-5\\_4](https://doi.org/10.1007/978-3-540-71256-5_4) (cited on page 144).
- [137] Yoshiki Yamazaki, Kwok Fai Cheung, and Zygmunt Kowalik. ‘Depth-integrated, non-hydrostatic model with grid nesting for tsunami generation, propagation, and run-up’. In: *International Journal for Numerical Methods in Fluids* 67.12 (Dec. 2011), pp. 2081–2107. doi: [10.1002/fld.2485](https://doi.org/10.1002/fld.2485) (cited on page 144).
- [138] Kenji Hirata et al. ‘The 2004 Indian Ocean tsunami: Tsunami source model from satellite altimetry’. In: *Earth, Planets and Space* 58.2 (Feb. 2006), pp. 195–201. doi: [10.1186/BF03353378](https://doi.org/10.1186/BF03353378) (cited on pages 145, 146).
- [139] D. Gopinathan et al. ‘Uncertainties in the 2004 SumatraAndaman source through nonlinear stochastic inversion of tsunami waves’. In: *Proceedings of the Royal Society A: Mathematical, Physical and Engineering Sciences* 473.2205 (Sept. 2017), p. 20170353. doi: [10.1098/rspa.2017.0353](https://doi.org/10.1098/rspa.2017.0353) (cited on pages 145–147).
- [140] B. Poisson, C. Oliveros, and R. Pedreros. ‘Is there a best source model of the Sumatra 2004 earthquake for simulating the consecutive tsunami?’ In: *Geophysical Journal International* 185.3 (June 2011), pp. 1365–1378. doi: [10.1111/j.1365-246X.2011.05009.x](https://doi.org/10.1111/j.1365-246X.2011.05009.x) (cited on pages 145, 146).
- [141] Remko Scharroo et al. ‘RADS: Consistent multi-mission products’. In: *20 Years of Progress in Radar Altimetry*. Vol. 710. 2013 (cited on page 146).

- [142] Yushiro Fujii and Kenji Satake. 'Tsunami Source of the 2004 Sumatra-Andaman Earthquake Inferred from Tide Gauge and Satellite Data'. In: *Bulletin of the Seismological Society of America* 97.1A (Jan. 2007), S192–S207. doi: [10.1785/0120050613](https://doi.org/10.1785/0120050613) (cited on page 147).



# Boris BONEV

## Ph.D. student in Applied Mathematics

 Chemin de la Venoge 2 1028 Préverenges, Switzerland  
 29. Januar 1989  
 +41(0)787073340  
 bonevbs@gmail.com  
 linkedin.com/in/bonevbs

 German/Bulgarian  
 bonevbs.github.io  
 github.com/bonevbs  
 scholar.google.com

## EDUCATION

Expected 2021	Ph.D. in Applied Mathematics, EPFL, Switzerland
February 2017	Supervised by Prof. Jan S. Hesthaven <ul style="list-style-type: none"><li>➢ Research and development of numerical methods and computational linear algebra algorithms for solving linear systems efficiently</li><li>➢ Research on finite element methods for solving PDEs with applications to geophysical flows</li><li>➢ Independent research, conceptualization and development of novel methods and algorithms</li><li>➢ Supervised thesis projects related to numerical methods and machine learning</li><li>➢ Authored peer-reviewed articles in top-tier journals</li><li>➢ Principal teaching assistant for Analysis I, II, III and for Statistics &amp; Probability for Engineers</li></ul>
January 2017	Researcher in Machine Learning, TU MUNICH, Germany
January 2016	<ul style="list-style-type: none"><li>➢ Research and design of novel machine learning methods for physics-based simulations</li><li>➢ Designed novel Machine Learning algorithms for efficient simulations of fluids</li><li>➢ Authored peer-reviewed article in a top-tier journal</li><li>➢ Experience in working with popular ML frameworks</li></ul>
December 2015	M.Sc. in Aerospace Engineering, UNIVERSITY OF STUTTGART, Germany
May 2013	<ul style="list-style-type: none"><li>➢ Focus on numerical methods and mathematical modelling</li><li>➢ Thesis work on the discontinuous Galerkin method for the shallow water equations</li></ul>
February 2015	B.Sc. in Physics, UNIVERSITY OF STUTTGART, Germany
October 2011	<ul style="list-style-type: none"><li>➢ Study of Physics in parallel to my degree in Aerospace Engineering</li><li>➢ Focus on Theoretical Physics and in particular Statistical Physics</li><li>➢ Thesis work on Markov chain models and optimization of collective transport using molecular motors</li></ul>
April 2013	B.Sc. in Aerospace Engineering, UNIVERSITY OF STUTTGART, Germany
October 2009	<ul style="list-style-type: none"><li>➢ Bachelor thesis on heat transfer in porous media</li></ul>

## SKILLS

Programming Matlab, Julia, Python, Fortran, C, C++, Mathematica  
Libraries Numpy, PyTorch, Tensorflow, Pandas  
Other tools git, make, gmake, bash, Xcode, Paraview, Unix, L<sup>A</sup>T<sub>E</sub>X

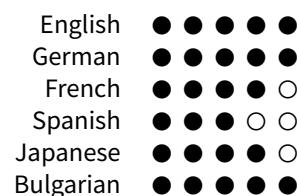
## INTERESTS

- Algorithm Design
- Scientific Computing
- Numerical Linear Algebra
- Machine Learning

## AWARDS

Scholarship of the German Academic Scholarship Foundation  
2009-2016

## LANGAUGES



## INTERNSHIPS & VOLUNTEER WORK

2014	Student Teaching Assistant in the Physics department.
2012	Research Intern at the German Aerospace Center (DLR) in Stuttgart, Germany.
2010-2012	Volunteer at the Aerospace Lab in Herrenberg, Germany. Helping high-school students build a micro-satellite.
2009	Intern at Mercedes-Benz in Untertürkheim, Germany
2008	Military service in the German Air Force

## SELECTED PROJECTS

---

### HssMATRICES.JL

2020 - 2021

 [github.com/bonevbs/HssMatrices.jl](https://github.com/bonevbs/HssMatrices.jl) To be presented at JuliaCon 2021

A Julia package for working with hierarchically semi-separable matrices. Provides efficient implementations of algorithms and arithmetic as well as tools for visualization.

 Julia

### HIERARCHICALSOLVERS.JL

2021

 [github.com/bonevbs/HierarchicalSolvers.jl](https://github.com/bonevbs/HierarchicalSolvers.jl)

A Julia implementation of efficient structured direct solvers for sparse matrices. Offers quasilinear complexity Gaussian elimination for sparse FEM matrices. Can be used as an approximate direct solver or as a preconditioner.

 Julia

### EXTENSIONS TO NODAL-DG

2018

 [github.com/bonevbs/nodal-dg-extension](https://github.com/bonevbs/nodal-dg-extension)

Extensions to the nodal-dg library, implementing continuous Galerkin and discontinuous Galerkin methods for a range of methods. Complete with some methods for generating a nested dissection as well as tools for visualization.

 Matlab

### CUSTOM TENSORFLOW EXTENSIONS

2016

*Not publicly available*

Custom TensorFlow modules implementing deconvolutions in 2d and 4d, complete with gradient information to facilitate backpropagation.

 C++  Python  Eigen

## REFEREED PUBLICATIONS & TALKS

---

Bonev, B., Hesthaven, J.S., **A HIERARCHICAL PRECONDITIONER FOR WAVE PROBLEMS IN QUASILINEAR COMPLEXITY.** *Under Review*, 2020.

 Preprint

Prantl, L., Bonev, B., Thuerey, N., **GENERATING LIQUID SIMULATIONS WITH DEFORMATION-AWARE NEURAL NETWORKS.** *ICLR Conference Proceedings*, 2019.

 openreview.net  Video

Hajihassanpour, M., Bonev, B., Hesthaven, J.S., **A COMPARATIVE STUDY OF EARTHQUAKE SOURCE MODELS IN HIGH-ORDER ACCURATE TSUNAMI SIMULATIONS.** *Ocean Modelling*, 141, 101429 2019.

 doi.org/10.1016/j.ocemod.2019.101429

Bonev, B., Hesthaven, J.S., Giraldo, F.X., Kopera, M.A., **DISCONTINUOUS GALERKIN SCHEME FOR THE SPHERICAL SHALLOW WATER EQUATIONS WITH APPLICATIONS TO TSUNAMI MODELING AND PREDICTION.** *Journal of Computational Physics*, 362, pp. 425-448 2018.

 doi.org/10.1016/j.jcp.2018.02.008  Preprint  Video 1  Video 2

Bonev, B., **HssMATRICES.JL - A JULIA PACKAGE FOR HIERARCHICALLY SEMI-SEPARABLE MATRICES.** *talk at JuliaCon*, 2021.

Upcoming

Bonev, B., **MODELING TSUNAMIS USING THE DISCONTINUOUS GALERKIN METHOD FOR THE SPHERICAL SHALLOW WATER EQUATIONS.** *2nd ASCETE workshop*, 2018.

Bonev, B., **LARGE-SCALE TSUNAMI SIMULATIONS USING THE DISCONTINUOUS GALERKIN METHOD.** *talk at the 27th Biennial Conference on Numerical Analysis*, 2017.