Strategies For Recycling Krylov Subspace Methods And Bilinear Form Estimation

Katarzyna Świrydowicz

Dissertation submitted to the Faculty of the
Virginia Polytechnic Institute and State University
in partial fulfillment of the requirements for the degree of

Doctor of Philosophy
in
Mathematics

Eric de Sturler, Chair
Mark Embree
Christopher J Roy
Lizette Zietsman

July 18th, 2017
Blacksburg, Virginia

Keywords: Bilinear form estimation, quadratic form estimation, sparse approximate inverse preconditioning, high performance computing, Krylov subspace recycling, diffuse optical tomography, topology optimization, computational fluid dynamics.

Strategies For Recycling Krylov Subspace Methods And Bilinear Form Estimation.

Katarzyna (Kasia) Świrydowicz

Abstract

The main theme of this work is effectiveness and efficiency of Krylov subspace methods and Krylov subspace recycling. While solving long, slowly changing sequences of large linear systems, such as the ones that arise in engineering, there are many issues we need to consider if we want to make the process reliable (converging to a correct solution) and as fast as possible. This thesis is built on three main components. At first, we target bilinear and quadratic form estimation. Bilinear form $c^T A^{-1} b$ is often associated with long sequences of linear systems, especially in optimization problems. Thus, we devise algorithms that adapt cheap bilinear and quadratic form estimates for Krylov subspace recycling. In the second part, we develop a hybrid recycling method that is inspired by a complex CFD application. We aim to make the method robust and cheap at the same time. In the third part of the thesis, we optimize the implementation of Krylov subspace methods on Graphic Processing Units (GPUs). Since preconditioners based on incomplete matrix factorization (ILU, Cholesky) are very slow on the GPUs, we develop a preconditioner that is effective but well suited for GPU implementation.

Strategies For Recycling Krylov Subspace Methods And Bilinear Form Estimation.

Katarzyna (Kasia) Świrydowicz

General Audience Abstract

In many applications we encounter the repeated solution of a large number of slowly changing large linear systems. The cost of solving these systems typically dominates the computation. This is often the case in medical imaging, or more generally inverse problems, and optimization of designs. Because of the size of the matrices, Gaussian elimination is infeasible. Instead, we find a sufficiently accurate solution using iterative methods, so-called Krylov subspace methods, that improve the solution with every iteration computing a sequence of approximations spanning a Krylov subspace. However, these methods often take many iterations to construct a good solution, and these iterations can be expensive. Hence, we consider methods to reduce the number of iterations while keeping the iterations cheap. One such approach is Krylov subspace recycling, in which we recycle judiciously selected subspaces from previous linear solves to improve the rate of convergence and get a good initial guess.

In this thesis, we focus on improving efficiency (runtimes) and effectiveness (number of iterations) of Krylov subspace methods. The thesis has three parts. In the first part, we focus on efficiently estimating sequences of bilinear forms, $\mathbf{c}^T\mathbf{A}^{-1}\mathbf{b}$. We approximate the bilinear forms using the properties of Krylov subspaces and Krylov subspace solvers. We devise an algorithm that allows us to use Krylov subspace recycling methods to efficiently estimate bilinear forms, and we test our approach on three applications: topology optimization for the optimal design of structures, diffuse optical tomography, and error estimation and grid adaptation in computational fluid dynamics. In the second part, we focus on finding the best strategy for Krylov subspace recycling for two large computational fluid dynamics problems. We also present a new approach, which lets us reduce the computational cost of Krylov subspace recycling. In the third part, we investigate Krylov subspace methods on Graphics Processing Units. We use a lid driven cavity problem from computational fluid dynamics to perform a thorough analysis of how the choice of the Krylov subspace solver and preconditioner influences runtimes. We propose a new preconditioner, which is designed to work well on Graphics Processing Units.

Acknowledgments

To be capable of everything and do justice to everything, one certainly does not need less spiritual force and élan and warmth, but more. What you call passion is not spiritual force, but friction between the soul and the outside world. Where passion dominates, that does not signify the presence of greater desire and ambition, but rather the misdirection of these qualities toward an isolated and false goal, with a consequent tension and sultriness in the atmosphere. Those who direct the maximum force of their desires toward the center, toward true being, toward perfection, seem quieter than the passionate souls because the flame of their fervor cannot always be seen.

- Herman Hesse, *Glass Bead Game*

"Is my passion perfect? "
"No, do it once again."

- Leonard Cohen, *Teachers*

First and foremost I would like to express my gratitude towards my advisor, Dr. Eric de Sturler, for challenging me to be a better mathematician and a better programmer, for his support, his patience, and the guidance he provided during my PhD. I would like to thank Dr. Mark Embree for sharing his (infectious) passion for numerical analysis and for his deep theoretical insights; Dr. Christopher Roy for our bi-weekly meetings and for challenging me to understand the ideas behind CFD simulations; Dr. Lizette Zietsman for her support, kindness and interest in my work.

A substantial part of the work presented in this thesis was either done as a collaboration or inspired by my collaborators. Thus, I want to thank the collaborators: Dr. Amit Amritkar, William Tyson, Dr. Xiao Xu, Dr. Danesh Tafti, Shelly Zhang, and Dr. Glaucio Paulino.

During my PhD studies I received a lot of support on many different levels. I would like to thank professor Wojciech Gajda from Adam Mickiewicz University in Poznań, Poland, for his academic support. He encouraged me to apply to graduate school in the US and helped me

with the application process. Dr. Kazimierz Sowiński, MD and his wife, Maryla, integrated the small Polish community in Blacksburg; they were very kind, and helped me make the US my second home. I am thankful to my friends and fellow graduate students for sharing the graduate school journey, for inspiring me, and for their support and advice. I am thankful to Clara, Rachel, Reina, Simoni, Vitor, Michael, and Chris.

This work would had never been possible if I have not received proper medical care and treatment during the times when it was very much needed. I would like to thank professor Wojciech Kozubski, MD, PhD, from Poznań University of Medical Sciences in Poznań, Poland, Dr. Jill Cramer, MD, from Blacksburg Associates in Neurology, and Dr. Jim Reinhard from Cook Concealing Center at Virginia Tech.

My parents, Teresa and Kazimierz, supported me morally and sometimes financially during my PhD studies. I am grateful to my Dad, who never stopped believing I would become a mathematician. My Mom told me once that *everyone has their own way*. I always recalled her wise words while going through challenging times, especially when I was in the process of completely changing the subject of my research.

Finally, I would like to thank my fiancé, Craig W Powers, for his ultimate love and support, for always being on my side, and for the endless discussions we have had about research and life.

This material is based upon work supported by the Air Force Office of Scientific Research under award numbers FA9550-12-1-0442 and FA9550-17-1-0205.

Remarks on notation

For the sake of clarity, in this thesis, the vectors and matrices are denoted by bold-face font. For example, $\mathbf{x}$ and $\mathbf{p}$ are vectors whereas x and p are scalars, $\mathbf{M}$ is a matrix, whereas M is a scalar.

We use i to denote $\sqrt{-1}$. The indexing variables are j, k, l, etc.

Unless explicitly stated otherwise, $\|\cdot\|$ denotes Euclidean norm.

Contents

1 Introduction **1**

2 Applications: description, importance, relevance **8**

2.1 Lid Driven Cavity 8

2.2 Topology Optimization 13

2.3 Diffuse Optical Tomography 14

2.3.1 Problem description 14

2.3.2 Randomized approach 16

2.4 Error estimation and grid adaptation in CFD 17

2.5 GenIDLEST 19

3 Krylov Methods and Preconditioners **21**

3.1 Krylov spaces 21

3.2 Krylov subspace methods 22

3.2.1 GMRES 22

3.2.2 Conjugate Gradient 23

3.2.3 BiCG and BiCGstab 23

3.3 Preconditioners 26

3.4 Recycling 27

3.4.1 Recycling GMRES: rGCROT/rGCRO-DR 28

3.4.2 Recycling BiCG and BiCGStab 28

3.4.3 Recycling CG 30

3.5 Algorithm listings. 30

4 Estimating Bilinear and Quadratic Forms 38

4.1 Motivation 38

4.2 Background 39

4.3 Analysis 41

4.3.1 Biconjugate gradient and bilinear form approximation 41

4.3.2 Extension of the method to recycled BiCG 42

4.3.3 Quadratic form estimates using CG and recycling CG. 46

4.3.4 Remarks on error estimates 49

4.4 Results 52

4.4.1 Topology optimization 52

4.4.2 Diffuse Optical Tomography 54

4.4.3 Error estimation and grid adaptation 67

5 Hybrid method using Krylov subspace recycling 74

5.1 Introduction 74

5.1.1 Motivation 74

5.1.2 Background 75

5.2 Method 75

5.2.1 Hybrid method 75

5.2.2 rBiCGStab with one recycle space 76

5.3 Results 78

5.3.1 Details of the testing setup 78

5.3.2 Turbulent Channel Flow 79

5.3.3 Flow Through Porous Media 80

6 Using High Performance Computing with Krylov subspace methods 83

6.1 Motivation 83

6.2 GPU architecture and fine grain parallelism 84

6.3 Overview of the published literature . . . 85
6.4 Solvers and preconditioners on the GPU . . . 86
6.4.1 Sparse Approximate Inverse (SAI) Preconditioner . . . 87
6.4.2 Multi-step Sparse Approximate Inverse . . . 87
6.5 Matrix format . . . 89
6.6 Algorithms . . . 92
6.6.1 Sparse Approximate Inverse: preconditioner computation . . . 92
6.6.2 Computing pattern of $\mathbf{A}^2$ . . . 92
6.6.3 Computing $\mathbf{R}$ for two-step SAI . . . 96
6.7 Implementation . . . 96
6.7.1 ILUT and Block ILUT preconditioners . . . 96
6.7.2 Two-step SAI preconditioner . . . 98
6.7.3 Jacobi preconditioners . . . 98
6.8 Results . . . 98
6.8.1 GMRES(m) . . . 99
6.8.2 BiCGStab . . . 101
6.8.3 Preconditioner Computation . . . 104
6.8.4 Summary . . . 106

List of Figures

1.1 Left: Lid Driven Cavity problem, velocity magnitude and streamlines at the steady state, Reynolds # =100, grid size 101×101. Right: the structure of the matrix formed in LDC problem (11×11 grid) 2

1.2 Diagram of the DOT. 4

2.1 A fragment of an LDC matrix with $n_x = 11$. The dark dots represent non-zeros. 12

3.1 Convergence of BiCG used to solve main and dual systems simultaneously for a DOT problem with tolerance 10^{-12} (convergence criteria (3.6)). Matrix size is 40401×40401. 25

3.2 Convergence of BiCG used to solve a main and dual systems simultaneously for a DOT problem with tolerance 10^{-12} (convergence criteria (3.6)). Matrix size is 40401×40401. 26

4.1 Large bridge (844325×844325 stiffness matrix) generated using the stochastic approach combined with rCG-qf with incomplete Cholesky preconditioner. The bridge was generated using a MATLAB implementation. The optimization process took 941 optimization steps and approximately 30h on a server with 96 GB RAM. 53

4.2 Left: number of iterations per linear system (function evaluation) in each optimization step; black dots on the bottom point out the optimization step where the Jacobian is computed. Right: reconstructed shape (amoeba). The first number in the titles is the (relative) tolerance used for the linear solver for cases, when an approximation to the Jacobian is needed and the second number is the (relative) tolerance for cases, when the approximation to the Jacobian is not needed. 59

4.3 Left: number of iterations per linear system (function evaluation) in each optimization step; black dots on the bottom point out the optimization step where the Jacobian is computed. Right: reconstructed shape (amoeba). The first number in the titles is the (relative) tolerance used for the linear solver for cases, when an approximation to the Jacobian is needed and the second number is the (relative) tolerance for cases, when the approximation to the Jacobian is not needed. 60

4.4 Left: number of iterations per linear system (function evaluation) in each optimization step; black dots on the bottom point out the optimization step where the Jacobian is computed. Right: reconstructed shape (amoeba). The first number in the titles is the (relative) tolerance used for the linear solver for cases, when an approximation to the Jacobian is needed and the second number is the (relative) tolerance for cases, when the approximation to the Jacobian is not needed. 61

4.5 Left: number of iterations per linear system (function evaluation) in each optimization step; black dots on the bottom point out the optimization step where Jacobian is computed. Right: reconstructed shape (amoeba). The first number in the titles is the (relative) tolerance used for the linear solver for cases, when an approximation to the Jacobian is needed and the second number is the (relative) tolerance for cases, when the approximation to the Jacobian is not needed. 62

4.6 Left: number of iterations per linear system (function evaluation) in each optimization step; black dots on the bottom point out the optimization step where the Jacobian is computed. Right: reconstructed shape (amoeba). The first number in the titles is the (relative) tolerance used for the linear solver for cases, when an approximation to the Jacobian is needed and the second number is the (relative) tolerance for cases, when the approximation to the Jacobian is not needed. 63

4.7 Graphical representation of the bilinear forms $\mathbf{c}_j\mathbf{A}^{-1}\mathbf{b}_k$, $j,k = 1,\ldots,10$ 64

4.8 Graphical representation of an order of computation. The numbers in the red boxes indicate the order in which the bilinear forms are evaluated. We start with $\mathbf{b}_1\mathbf{A}^{-1}\mathbf{c}_1$, we evaluate $\mathbf{b}_2\mathbf{A}^{-1}\mathbf{c}_1$ next, then $\mathbf{b}_3\mathbf{A}^{-1}\mathbf{c}_1$, etc. 65

4.9 Four patterns (orders) for computing bilinear forms $\mathbf{c}_k\mathbf{A}^{-1}\mathbf{b}_j$ for $j = 1,\ldots,10$ and $k = 1,\ldots 10$ 66

4.10 rBiCG-bf and the influence of order of recycling on errors and the number of iterations. 67

4.11 Grid adaptation for Quasi-1D nozzle problem using node shifting. Figure comes from [93] 70

6.1 Runtime of the matrix-vector multiplication for LDC matrices using different storage formats. The bars show time in milliseconds per 100 SpMVs with the same random vector. The runtimes were computed on Tesla K20c GPU. 91

6.2 Illustration of neighbor-of-neighbor principle for 5 point stencil. 95

List of Tables

4.1 Results for a 3D bridge problem. The stiffness matrix has size 844325×844325, and there are 36 load cases in the original problem. The results were obtained using matlab, on a server with 96 GB RAM. In the table, **qf** denotes 'quadratic form'. 53

4.2 Results of the comparison of different tolerance settings for evaluating the objective function $\|\mathbf{R}\|^2$ in DOT. The errors in the three bottom rows of each table are relative and averaged out over multiple optimization steps. The presented values are relative errors computed using the solution obtained through direct solve. In the third row, the first number is the number of objective functions evaluations, and the number in the brackets indicates how many times we evaluated the Jacobian. BF = bilinear form. 57

4.3 Relative residual norms in the ETE and the adjoint problem (AP), and the relative error in the functional correction (BF) computed with rBiCG-bf solved with tolerance parameter shown on the very to of each individual table. 71

4.4 Time comparison (in seconds) between traditional (GMRES(45)) and BiCG-bf approach for Quasi-1D nozzle problem with different size grids. 72

5.1 Results for first 30 time steps of Turbulent Channel Flow simulation. Each time step corresponds to $5 \cdot 10^{-5}$ seconds. The relative tolerance for the solvers is 10^{-6}. The results were computed on a single machine with Dual Intel Xeon CPU X5650, 2.67GHz, 48GB RAM. 80

5.2 Results for the last 30 time steps of the turbulent channel flow simulation. Each time step corresponds to $5 \cdot 10^{-5}$ seconds. The initial guess for the first system is $\mathbf{x}_0 = \mathbf{0}$, and the relative tolerance for the solvers is 10^{-6}. The results were computed on a single machine with Dual Intel Xeon CPU X5650, 2.67GHz, 48GB RAM. 80

5.3 Total time for 10 time steps of flow through porous media for different preconditioners and solvers. The hyphens indicate that the method with the other preconditioner was worse. The convergence tolerance for all solvers is 10^{-6}. The results were computed on 16 CPU cores (Dual Intel Xeon CPU E5-2670, 2.60GHz, 64GB RAM) with MPI parallelism on BlueRidge HPC cluster. 81

5.4 Flow through porous media. The table shows the comparison between different solver used in the 10th time step. The numbers in brackets denote that the method performed maximum number of matrix vector products and did not converge. The convergence tolerance for all solvers is 10^{-6}. The results were computed on 16 CPU cores (Dual Intel Xeon CPU E5-2670, 2.60GHz, 64GB RAM) with MPI parallelism on BlueRidge HPC cluster. 81

5.5 Flow through porous media. The leftmost column shows the number of time steps, at which we use rGCROT(m,k) before we switch the solver to rBiCGStab. The middle column shows the solution time in the 10th time step, which is a good indicator of quality of the recycle space. The convergence tolerance for both solvers is 10^{-6}; the results were computed on 16 CPU cores (Dual Intel Xeon CPU E5-2670, 2.60GHz, 64GB RAM) with MPI parallelism on BlueRidge HPC cluster. 82

6.1 Basic characteristics of the matrices used in testing the influence of the matrix storage format on the performance of SpMV. 91

6.2 Average runtimes (per linear system) for GMRES(40) with ILUT as a preconditioner. We give time in milli1seconds on the left and the percentage of the total solver time on the right. We also provide runtimes for main parts of the solver: SpMV time (MV), precvec time (PrecV), modified Gram-Schmidt time (GS). The last column shows the runtimes for BiCGStab with SAI applied to the same problems. This shows how much the runtimes improve by change of preconditioner and solver. 99

6.3 Average runtimes (per linear system) for GMRES(40) with BILUT as a preconditioner. We give time in milli1seconds on the left and the percentage of the total solver time on the right. We also provide runtimes for main parts of the solver: SpMV time (MV), precvec time (PrecV), modified Gram-Schmidt time (GS). 100

6.4 Average runtimes (per linear system) for GMRES(40) with SAI as a preconditioner. We give time in milli1seconds on the left and the percentage of the total solver time on the right. We also provide runtimes for main parts of the solver: SpMV time (MV), precvec time (PrecV), modified Gram-Schmidt time (GS). 100

6.5 Average runtimes (per linear system) for GMRES(40) with Jacobi as a preconditioner. We give time in milli1seconds on the left and the percentage of the total solver time on the right. We also provide runtimes for main parts of the solver: SpMV time (MV), precvec time (PrecV), modified Gram-Schmidt time (GS). 101

6.6 Average runtimes (per linear system) for GMRES(40) with two-step SAI as a preconditioner. We give time in milli1seconds on the left and the percentage of the total solver time on the right. We also provide runtimes for main parts of the solver: combined SpMV-precvec time (MV-PrecV) and modified Gram-Schmidt time (GS). 101

6.7 Average runtimes (per linear system) for BiCGStab with ILUT as a preconditioner. We give time in milli1seconds on the left and the percentage of the total solver time on the right. We also provide runtimes for main parts of the solver: SpMV time (MV), Preconditioner-vector multiplication time (PrecV). . 102

6.8 Average runtimes (per liner system) for BiCGStab with SAI as a preconditioner. We give time in milli1seconds on the left and the percentage of the total solver time on the right. We also provide runtimes for main parts of the solver: SpMV time (MV), Preconditioner-vector multiplication time (PrecV). 103

6.9 Average runtimes (per linear system) for BiCGStab with Jacobi as a preconditioner. We give time in milli1seconds on the left and the percentage of the total solver time on the right. We also provide runtimes for main parts of the solver: SpMV time (MV), Preconditioner-vector multiplication time (PrecV). . 103

6.10 Average runtimes (per linear system) for BiCGStab with two-step SAI as a preconditioner. We give times as milliseconds on the left and percentage of the total solver time on the right. SpMV performed together with precvec is labeled as MV-Prec-V. 104

6.11 Average runtimes (per linear system) for preconditioner computation (setup) and preconditioner computation plus solver execution (total). 105

6.12 Average runtimes (per linear system) for preconditioner computation (setup) and preconditioner computation plus solver execution (total). SAI2 = two-step SAI. 105

Chapter 1

Introduction

Let $\mathbf{A}$ be an $n \times n$ sparse matrix and b be an $n \times 1$ vector. In this thesis, we aim to find an approximate solution to the system of equations

$$\mathbf{A}\mathbf{x} = \mathbf{b}, \tag{1.1}$$

and we focus on doing it in an efficient (computationally fast) way. We denote an approximate solution of (1.1) by $\tilde{\mathbf{x}}$. The efficiency of solving (1.1) depends on three factors: (1) the applied mathematical algorithm, (2) the implementation, and (3) the characteristics of the problem with which the system (1.1) is associated. In various parts of this thesis we consider one or more of the above-mentioned factors.

We sometimes solve (1.1) together with an accompanying system

$$\mathbf{A}^T\mathbf{y} = \mathbf{c}. \tag{1.2}$$

We refer to (1.2) as an *adjoint* or *dual* system. We denote its approximate solution by $\tilde{\mathbf{y}}$.

Moreover, many applications require solving long, slowly changing sequences of systems

$$\mathbf{A}^{(k)}\mathbf{x}^{(k)} = \mathbf{b}^{(k)}, \tag{1.3}$$

for $k = 1, 2, \ldots$, where $\mathbf{A}^{(k)}$ or $\mathbf{b}^{(k)}$ changes with k. We might be required to solve an analogous sequence of dual systems, too.

As a matter of fact, systems of type (1.1)–(1.3) are common in engineering applications. They arise in inverse and optimization problems, and in flow simulations based on the Navier-Stokes equations. In most applications, the matrix $\mathbf{A}$ is very large, and a direct solve of the linear system (1.1) is impossible or too costly. Often a large number of linear systems must be solved, and the solution of system k determines system $k + 1$. Thus, the entire process is computationally challenging. Krylov subspace methods [95] are a good choice for solving large systems (1.1)-(1.3). These methods find an approximate solution in the Krylov subspace

through an iterative process instead of solving (1.1) directly with Gaussian elimination or LU decomposition. However, Krylov subspace methods may require many iterations to produce a solution with desired accuracy. These methods usually require a preconditioner to keep the number of iterations reasonable, but a preconditioner might be expensive to compute or to apply.

A large part of this work is motivated by applications. In the next section, we briefly characterize the systems we are solving, and we discuss their main features to explain the challenges associated with the applications.

Applications

Lid Driven Cavity

The *Lid Driven Cavity (LDC)* problem is a standard computational fluid dynamics (CFD) problem. It is often used as a benchmark for testing new methods and codes. A viscous Newtonian (incompressible) fluid is placed in a two-dimensional rectangular cavity and put in motion by the movement of the upper edge (the lid), which moves with a constant velocity; see Table 1.1.

The governing equations are the Navier-Stokes equations (conservation of mass and momentum). We solve for a steady state using pseudo time-stepping. In every pseudo time-step, we form a linear system to solve for the two velocity components and the pressure. The resulting matrices are sparse, with maximum of 9 non-zeros per row.

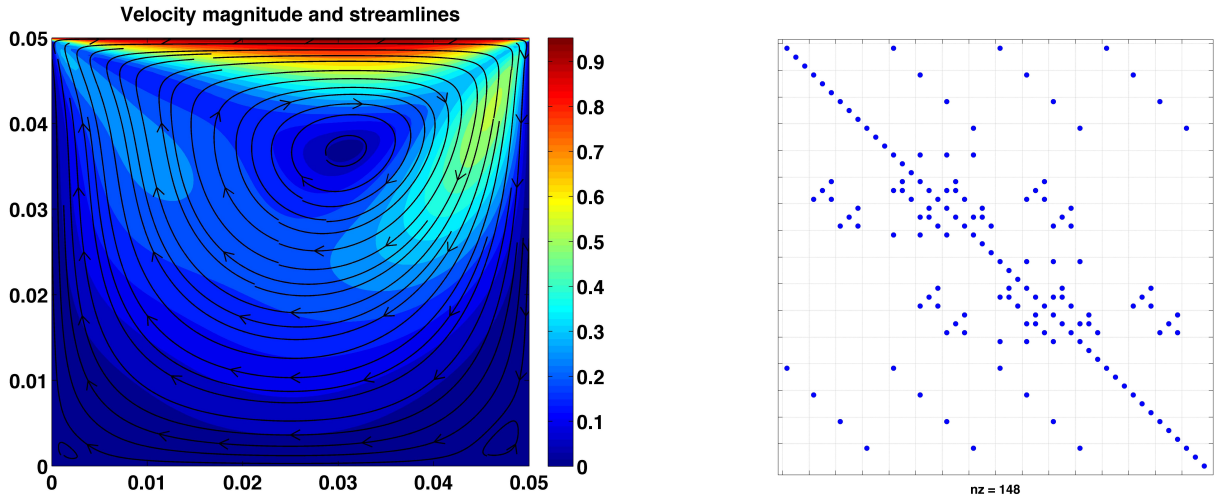


Figure 1.1: Left: Lid Driven Cavity problem, velocity magnitude and streamlines at the steady state, Reynolds # =100, grid size 101×101. Right: the structure of the matrix formed in LDC problem (11×11 grid)

To compute a steady state solution, we need to solve a long sequence of linear systems. Solving these systems by an iterative Krylov subspace method is the most time-consuming part of the simulation. Although the right hand side and the matrix change from time step to time step, the non-zero pattern of the matrix remains the same, see Table 1.1.

Computational results for the LDC problem are representative for a large class of CFD simulations based on the discretized Navier-Stokes equations. In Table 6, we discuss efficiently solving the resulting linear systems by Krylov subspace methods on the GPUs. We consider the influence of matrix storage formats on the performance. In addition, we examine two different Krylov subspace solver choices and various preconditioning techniques for the LDC problem. We analyze how combinations of these choices – and their implementation – affect the solver runtimes.

Topology optimization

Structural topology optimization [15] is a method for optimizing structural designs. In the problems considered in this thesis, we want to find a structure that minimizes compliance for a set of load cases and for a fixed volume. Using the equations from linear elasticity, we state the minimization problem as

$$\min_{\boldsymbol{\rho}} C(\boldsymbol{\rho}) = -\sum_{j=1}^{m} \alpha_j \mathbf{f}_j^T \mathbf{u}_j(\boldsymbol{\rho}), \tag{1.4}$$

where C is the compliance, ρ denotes the vector of design variables (density field) of size $M \times 1$ (M - number of elements in finite volume discretization), m is the number of load cases, $\mathbf{f}_j$ and $\mathbf{u}_j$, $j = 1, 2, \ldots, m$ are the load case and the displacement vectors, respectively, and the α_js are weights.

The displacement is defined as $\mathbf{u}_j = \mathbf{K}^{-1}(\boldsymbol{\rho})\mathbf{f}_j$, where $\mathbf{K}(\boldsymbol{\rho})$ is the stiffness matrix as a function of density per element, and can be rewritten as

$$\min_{\boldsymbol{\rho}} C(\boldsymbol{\rho}) = -\sum_{j=1}^{m} \alpha_j \mathbf{f}_j^T \mathbf{K}^{-1}(\boldsymbol{\rho})\mathbf{f}_j. \tag{1.5}$$

The matrix $\mathbf{K}$ is symmetric positive definite. We note that the equation (1.5) is actually a sum of m matrix quadratic forms, hence we are minimizing a sum of quadratic forms. The most obvious way to evaluate a sum of quadratic forms is to solve a linear system $\mathbf{K}(\boldsymbol{\rho})\mathbf{x}_j = \mathbf{f}_j$ and compute an inner product $\mathbf{f}_j^T\mathbf{x}_j$ for each quadratic form. Such strategy requires solving m, possibly very large, linear systems in each optimization step. The number of linear systems can be reduced by applying randomization [100], yet even with randomization, we still need to solve several linear systems per optimization step. Instead of evaluating each quadratic form through inner product, we can use a quadratic form estimate based on properties of Krylov subspaces (see Table 4 for details) to further reduce the computational cost. Moreover, the matrix $\mathbf{K}(\boldsymbol{\rho})$, we can use Krylov subspace recycling to preserve a part of Krylov subspace

from one linear system to another, and get faster convergence as a result (Krylov subspace recycling is explained in Table 3).

We use randomization, quadratic form estimates, and Krylov subspace recycling together. Improving a single part of the optimization process does not reduce the cost as much as thoughtfully integrating various strategies.

Diffuse Optical Tomography

In diffuse optical tomography (DOT), [8] we solve for absorption and diffusion fields in a domain, given measurements on the surface. The *sources*, placed on the top of the domain, send infrared or near infrared light into the medium; the light scatters and gets absorbed. The detectors, placed on the bottom, measure the propagated light. The diagram of the DOT is shown on Table 1.2.

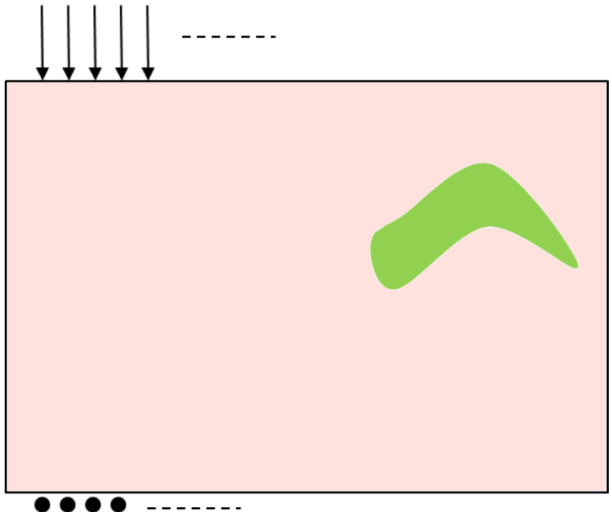


Figure 1.2: Diagram of the DOT.

This leads to a nonlinear least-squares problem

$$\min_{\mathbf{p}} \mathbf{f}(\mathbf{p}) = \min_{\mathbf{p}} \|\mathbf{M}(\mathbf{p}) - \mathbf{d}\|_2, \tag{1.6}$$

where $\mathbf{p}$ is the vector of parameters, $\mathbf{M}$ is a vector of computed measurements generated with the forward model, and $\mathbf{d}$ the vector of measured data at the detectors. In Table 2, we show that (1.6) can be written as

$$\min_{\mathbf{p}} \mathbf{f}(\mathbf{p}) = \min_{\mathbf{p}} \|\mathbf{C}^{\mathbf{T}}\mathbf{A}(\mathbf{p})^{-\mathbf{1}}\mathbf{B} - \mathbf{D}\|_F, \tag{1.7}$$

where $\mathbf{C}$ represents the detectors, $\mathbf{B}$ corresponds to the sources, and $\mathbf{A}(\mathbf{p})$ comes from the discretization of a partial differential equation that depends on the (unknown) absorption and

diffusion fields. The number of columns in $\mathbf{C}$ equals the number of detectors, and the number of columns in $\mathbf{B}$ equals the number of sources.

Problem (1.7) is computationally expensive, because we usually have many sources and many detectors, and a finely discretized domain. Each evaluation of $\mathbf{f}$ requires solving a linear system for each source-deterctor-frequency combination. If the Jacobian is required, in addition, for each source-deterctor-frequency combination, we need to solve a corresponding adjoint systems. Let us say that we have only one frequency and 32 sources and 32 detectors (in real applications we use more sources, detectors, and frequencies). Thus, we solve 32 linear systems at every optimization step, and 64 total linear systems if the Jacobian is evaluated. To find a minimum, we need to evaluate $\mathbf{f}$ (often with the Jacobian) many times.

The straightforward way to evaluate $\mathbf{f}$ is through solving $\mathbf{A}(\mathbf{p})\mathbf{X} = \mathbf{B}$ (note that the number of columns in $\mathbf{B}$ equals the number of sources) and using an inner product, i.e., $\mathbf{f}(\mathbf{p}) = \|\mathbf{C}^\mathbf{T}\mathbf{X} - \mathbf{D}\|_F$, or through solving $\mathbf{A}^\mathbf{T}(\mathbf{p})\mathbf{Y} = \mathbf{C}$ and using $\mathbf{f}(\mathbf{p}) = \|\mathbf{Y}^\mathbf{T}\mathbf{B} - \mathbf{D}\|_F$. If the Jacobian is needed, we need both $\mathbf{X}$ and $\mathbf{Y}$. The number of systems to be solved can be reduced by using a randomized approach [9], however, we still need to solve multiple linear systems in each optimization step.

We observe that for each column $\mathbf{b}_k$ of $\mathbf{B}$ and each column $\mathbf{c}_j$ of $\mathbf{C}$, the expression $\mathbf{c}_j\mathbf{A}^{-1}(\mathbf{p})\mathbf{b}_k$ is a matrix bilinear form. To further reduce the cost, we use bilinear form estimates based on BiCG [90] (the BiCG algorithm is explained in Table 3 and the estimates are discussed in Table 4). Krylov subspace recycling reduces the cost even more, because (1) the matrix $\mathbf{A}(\mathbf{p})$ is the same for all the systems in the same optimization step and the right hand sides repeat, too, and (2) the matrix changes slowly from one optimization step to another, [55].

The three strategies, randomization, bilinear form estimates, and Krylov subspace recycling applied together, significantly decrease runtime without affecting the quality of the solution to the optimization problem, see Table 4.

Functional-based error estimation and grid adaptation

Following [75], the discretization error in the CFD simulation is defined as *the difference between the exact solution to the discrete equations and the exact solution to the mathematical model (PDE)*. The discretization error is usually the largest source of numerical error in CFD [74]. The discretization error can be reduced by using mesh refinement. Applying uniform refinement to the entire mesh (to each cell or node) is often inefficient. Instead, we perform so-called *targeted* or *local mesh refinement*. We evaluate multiple functional errors ε_J, used as adaptation indicators (the functionals such as lift, drag, etc), which are approximated as

$$\varepsilon_\mathbf{J} = -\mathbf{c}_j^T\mathbf{A}_j^{-1}\mathbf{b}, \tag{1.8}$$

where $\mathbf{A}_j$ is a non-symmetric Jacobian matrix, $\mathbf{c}_j$ is the gradient of the discrete functional $\mathbf{J}_j$, $j = 1, 2, \ldots$, and $\mathbf{b}$ is the truncation error (truncation error does not change unless the grid is adapted, hence no subscript). In addition, we need approximate solutions to the equations $\mathbf{A}_j\mathbf{x} = \mathbf{b}$ (error transport equation) and $\mathbf{A}^T\mathbf{y} = \mathbf{c}_j$ (adjoint problem). We have multiple vectors

$\mathbf{c}_j$s and multiple matrices $\mathbf{A}_j$s, therefore we can apply Krylov subspace recycling. Also, $\varepsilon_{\mathbf{J}}$ is a matrix bilinear form. It turns out that all three quantities: ε_J, $\mathbf{y}_j$ and $\mathbf{x}_j$ can be computed at the same time using BiCG or recycled BiCG equipped with bilinear form estimation (we explain this method in detail in Table 4). Moreover, the (r)BiCG-based estimate for $\varepsilon_{\mathbf{J}}$ computed this way is more accurate than the inner product of truncation error and $\mathbf{c}_j$, [90].

This application is quite untypical, because we consider a method, not a particular engineering problem. We extend a method used in CFD by replacing expensive solves with bilinear form estimates. In Table 4 we demonstrate the effectiveness of this approach on a Quasi1D Euler (Quasi-1D Nozzle) problem; however, the approach is more general.

GENIDLEST (turbulent channel flow and flow through porous media)

GenIDLEST stands for *Generalized Incompressible Direct and Large-Eddy Simulations of Turbulence*, which is a large collection of CFD codes written in Fortran 77 and developed in the Department of Mechanical Engineering at Virginia Tech [91]. The code is used to solve the incompressible Navier-Stokes and energy equations in a generalized structured body-fitted multi-block framework. GenIDLEST is designed to work with large 2D and 3D problems.

In the process of solving the Navier-Stokes equations, we use pseudo-time steps. The goal is to either capture the transient flow characteristics of unsteady flows or to converge to a steady state solution. Similarly as in the LDC problem, we obtain a sequence of linear systems of equations. In each time step, we need to solve either three linear systems (for 2D problems) or four systems (for 3D problems). However, the only system that is expensive to solve is the system arising from the pressure Poisson equation. For non-Cartesian grids and certain boundary conditions, this system is non-symmetric. Applying the preconditioner is the most expensive part of the linear solver, thus a solver with small number of iterations is required. In this thesis, we consider two particular problems in GenIDLEST (turbulent channel flow and flow through porous media). In both cases, the matrix in the pressure Poisson equation remains constant and the right hand side changes from one time step the next.

GenIDLEST uses one of the two linear solvers: GMRES(m) and BiCGStab (Table 3). GMRES(m) is more robust, but also more expensive in terms of storage and computations compared to BiCGStab. In the beginning of the simulation, when the initial guess is far from the solution, the pressure Poisson system is particularly hard to solve. BiCGstab might fail in this phase, so we use GMRES(m). Once the systems get easier to solve, GenIDELEST switches to BiCGstab, which is much cheaper in terms of storage and computational cost.

The problems above are good candidates for using Krylov subspace recycling (Table 3). However, while recycling methods, such as rGCROT(m,k) [7], decrease the total number of iterations they may be expensive in terms of runtime. To reduce the runtime, we use rGCROT(m,k) only in the first few time steps, and then we switch to rBiCGstab, which is a cheaper solver. Moreover, after a few time steps with rGCROT(m,k), we obtain a recycle space, which we can use in rBiCGstab. In addition, we use a simplified version of rBiCGStab, which requires only one recycle space (the original method requires two recycle spaces). This version is

computationally very efficient. The details are given in Table 5. The hybrid approach (rC-CROT/rBiCGStab) provides a significant gain in the number of preconditioned matrix-vector products and the total time for all the time steps.

In Table 5, we describe the details of the approach, present numerical results, and analyze the influence of the various algorithmic choices on the total runtime.

Summary

Our discussion of several applications clearly demonstrates that efficiently solving (1.1), (1.2), and in particular, (1.3), is difficult but important for science and engineering. We focus on finding overall strategies for solving such system as fast as possible. We focus on different parts of the solution process in each application. In the beginning of this chapter, we emphasized that there are three factors which influence the efficiency of solving (1.1) - (1.3): (1) the applied mathematical algorithm, (2) the implementation, and (3) the characteristics of the problem with which the system (1.1) is associated.

We target the algorithm while devising the hybrid method for GenIDLEST, and while using the bilinear and quadratic form estimates. We target the implementation while optimizing the performance of the linear solvers on the GPU for the Lid Driven Cavity problem, and while finding an efficient preconditioner for use on the GPUs. We reformulate the problem, while using stochastic approach combined with Krylov subspace recycling in topology optimization and diffuse optical tomography, and in the functional-based error estimation in CFD. We often need to combine all three optimization strategies to see large runtime reduction. We note that the computational acceleration discussed in detail in Table 6 and implemented only for the LDC problem, can also be used for the remaining applications.

Chapter 2

Applications: description, importance, relevance

2.1 Lid Driven Cavity

Navier-Stokes Equations

The governing equations for the LDC problem are the incompressible Navier-Stokes equations, where p is pressure, u and v are velocities in the x and y directions, respectively, ν is the kinematic viscosity, and ρ is the density. In addition, we assume that the bottom, left and right walls are fixed with no-slip boundary conditions. The boundary condition for the lid is of Dirichlet type, setting a fixed velocity.

$$\frac{\partial u}{\partial x} + \frac{\partial v}{\partial y} = 0, \tag{2.1}$$

$$\frac{\partial u}{\partial t} + u\frac{\partial u}{\partial x} + v\frac{\partial u}{\partial y} + \frac{1}{\rho}\frac{\partial p}{\partial x} = \nu\left(\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2}\right), \tag{2.2}$$

$$\frac{\partial v}{\partial t} + u\frac{\partial v}{\partial x} + v\frac{\partial v}{\partial y} + \frac{1}{\rho}\frac{\partial p}{\partial y} = \nu\left(\frac{\partial^2 v}{\partial x^2} + \frac{\partial^2 v}{\partial y^2}\right). \tag{2.3}$$

In our implementation, the mass-conservation equation (2.1) is replaced by a different equation. We use artificial compressibility to get a time dependence for pressure [48]. This guarantees that the divergence-free condition is satisfied once we reach a steady state solution. Equation (2.1) becomes

$$\frac{1}{\rho\beta^2}\frac{\partial p}{\partial t} + \frac{\partial u}{\partial x} + \frac{\partial v}{\partial y} = -\lambda_x C_x \frac{\Delta x^3}{\rho\beta^2}\frac{\partial^4 p}{\partial x^4} - \lambda_y C_y \frac{\Delta y^3}{\rho\beta^2}\frac{\partial^4 p}{\partial y^4}, \tag{2.4}$$

where the fourth derivatives of pressure serve as an artificial viscosity [96]. Artificial viscosity prevents odd-even decoupling of the pressure. Equations (2.1) -(2.3) are the governing equations for the LDC problem. The coefficients C_x, C_y in (2.4) are both equal 0.01 and λ_x, λ_y are related to the velocities, following $\lambda_x = \frac{1}{2}\left[u + \sqrt{u^2 + 4\beta^2}\right]$ and $\lambda_y = \frac{1}{2}\left[v + \sqrt{v^2 + 4\beta^2}\right]$, where β is the artificial compressibility parameter.

As we solve for a steady state, we define the steady state residuals, which measure how close we are to a steady state.

$$\begin{aligned} R_1 &= \frac{\partial u}{\partial x} + \frac{\partial v}{\partial y} + \lambda_x C_x \frac{\Delta x^3}{\rho\beta^2}\frac{\partial^4 p}{\partial x^4} + \lambda_y C_y \frac{\Delta y^3}{\rho\beta^2}\frac{\partial^4 p}{\partial y^4}, \\ R_2 &= u\frac{\partial u}{\partial x} + v\frac{\partial u}{\partial y} + \frac{1}{\rho}\frac{\partial p}{\partial x} - \nu\left(\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2}\right), \\ R_3 &= u\frac{\partial v}{\partial x} + v\frac{\partial v}{\partial y} + \frac{1}{\rho}\frac{\partial p}{\partial y} - \nu\left(\frac{\partial^2 v}{\partial x^2} + \frac{\partial^2 v}{\partial y^2}\right). \end{aligned}$$

We rewrite (2.1)–(2.3) using R_1, R_2 and R_3, and we get

$$\begin{aligned} \frac{1}{\rho\beta^2}\frac{\partial p}{\partial t} + R_1 &= 0, \\ \frac{\partial u}{\partial t} + R_2 &= 0, \\ \frac{\partial v}{\partial t} + R_3 &= 0. \end{aligned} \tag{2.5}$$

We define $\mathbf{V} = [p, u, v]^T$ and $\mathbf{R} = [R_1, R_2, R_3]^T$, which gives

$$\mathbf{B}B\frac{\partial \mathbf{V}}{\partial t} + \mathbf{R} = 0, \tag{2.6}$$

where $\mathbf{B}$ is the 3×3 diagonal matrix $\begin{bmatrix} \frac{1}{\rho\beta^2} & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$.

Discretization

We use a five point, second-order, central finite difference stencil for spatial discretization. The artificial viscosity term in the mass conservation equation is treated explicitly for the purpose of preserving the block pentadiagonal structure.

We define the steady state iterative residuals as

$$\begin{aligned} R^{n+1}_{1,(j,k)} = &\frac{u^{n+1}_{j+1,k} - u^{n+1}_{j-1,k}}{2\Delta x} + \frac{v^{n+1}_{j,k+1} - v^{n+1}_{j,k-1}}{2\Delta y} \\ &+ \lambda_x C_x \frac{p^n_{i+2,j} - 4p^n_{j+1,k} + 6p^n_{j,k} - 4p^n_{j-1,k} + p^n_{i-2,j}}{\Delta x^4} \\ &+ \lambda_y C_y \frac{p^n_{j,k+2} - 4p^n_{j,k+1} + 6p^n_{j,k} - 4p^n_{j,k-1} + p^n_{j,k-2}}{\Delta y^4}, \end{aligned} \tag{2.7}$$

$$\begin{aligned} R^{n+1}_{2,(j,k)} = u^n_{j,k} &\frac{v^{n+1}_{j+1,k} - v^{n+1}_{j-1,k}}{2\Delta x} + v^n_{j,k} \frac{v^{n+1}_{j,k+1} - v^{n+1}_{j,k-1}}{2\Delta y} + \frac{1}{\rho} \frac{p^{n+1}_{j,k+1} - p^{n+1}_{j,k-1}}{2\Delta y} \\ &- \nu \left(\frac{v^{n+1}_{j+1,k} - 2v^{n+1}_{j,k} - v^{n+1}_{j-1,k}}{\Delta x^2} + \frac{v^{n+1}_{j,k+1} - 2v^{n+1}_{j,k} - v^{n+1}_{j,k-1}}{\Delta y^2} \right), \end{aligned} \tag{2.8}$$

$$\begin{aligned} R^{n+1}_{3,(j,k)} = u^n_{j,k} &\frac{v^{n+1}_{j+1,k} - v^{n+1}_{j-1,k}}{2\Delta x} + v^n_{j,k} \frac{v^{n+1}_{j,k+1} - v^{n+1}_{j,k-1}}{2\Delta y} + \frac{1}{\rho} \frac{p^{n+1}_{j,k+1} - p^{n+1}_{j,k-1}}{2\Delta y} \\ &- \nu \left(\frac{v^{n+1}_{j+1,k} - 2v^{n+1}_{j,k} - v^{n+1}_{j-1,k}}{\Delta x^2} + \frac{v^{n+1}_{j,k+1} - 2v^{n+1}_{j,k} - v^{n+1}_{j,k-1}}{\Delta y^2} \right). \end{aligned} \tag{2.9}$$

Since we apply a five point stencil, $\mathbf{R}^{n+1}$ at grid point (j,k) relates only to its immediate neighbors on the grid: (j,k), $(j+1,k)$, $(j-1,k)$, $(j,k+1)$, and $(j,k-1)$. Hence, $\mathbf{R}^{n+1}_{j,k}$ is a function of $\mathbf{V}^{n+1}_{j,k}, \mathbf{V}^{n+1}_{j-1,k}, \mathbf{V}^{n+1}_{j+1,k}, \mathbf{V}^{n+1}_{j,k+1}$, and $\mathbf{V}^{n+1}_{j,k-1}$.

Neglecting the higher order terms, in Taylor expansion we have for $\mathbf{R}^{n+1}_{j,k}$ and $\mathbf{R}^n_{j,k}$

$$\begin{aligned} \mathbf{R}^{n+1}_{j,k} = \mathbf{R}^n_{j,k} &+ \left.\frac{\partial \mathbf{R}}{\partial \mathbf{V}}\right|^n_{j,k} \Delta \mathbf{V}^{n+1}_{j,k} + \left.\frac{\partial \mathbf{R}}{\partial \mathbf{V}}\right|^n_{j+1,k} \Delta \mathbf{V}^{n+1}_{j+1,k} + \left.\frac{\partial \mathbf{R}}{\partial \mathbf{V}}\right|^n_{j-1,k} \Delta \mathbf{V}^{n+1}_{j-1,k} \\ &+ \left.\frac{\partial \mathbf{R}}{\partial \mathbf{V}}\right|^n_{j,k+1} \Delta \mathbf{V}^{n+1}_{j,k+1} + \left.\frac{\partial \mathbf{R}}{\partial \mathbf{V}}\right|^n_{j,k-1} \Delta \mathbf{V}^{n+1}_{j,k-1}, \end{aligned} \tag{2.10}$$

where $\Delta \mathbf{V}^{n+1}_{j,k} = \mathbf{V}^{n+1}_{j,k} - \mathbf{V}^n_{j,k}$, etc.

We discretize the time-dependent term using the backward Euler method, and we treat the steady state residuals $\mathbf{R}$ implicitly. The vector form LDC equations (2.6) can be written in the following discretized form for each grid point (j,k):

$$B_{j,k} \frac{\mathbf{V}^{n+1}_{j,k} - \mathbf{V}^n_{j,k}}{\Delta t} = -\mathbf{R}^{n+1}_{j,k}. \tag{2.11}$$

Substituting (2.11) back into (2.10) gives us

$$\left(\frac{B_{j,k}}{\Delta t}+\frac{\partial \mathbf{R}}{\partial \mathbf{V}}\Big|_{j,k}^{n}\right)\Delta \mathbf{V}_{j,k}^{n+1}+\frac{\partial \mathbf{R}}{\partial \mathbf{V}}\Big|_{j+1,k}^{n}\Delta \mathbf{V}_{j+1,k}^{n+1}+\frac{\partial \mathbf{R}}{\partial \mathbf{V}}\Big|_{j-1,k}^{n}\Delta \mathbf{V}_{j-1,k}^{n+1} + \frac{\partial \mathbf{R}}{\partial \mathbf{V}}\Big|_{j,k+1}^{n}\Delta \mathbf{V}_{j,k+1}^{n+1}\frac{\partial \mathbf{R}}{\partial \mathbf{V}}\Big|_{j,k-1}^{n}\Delta \mathbf{V}_{j,k-1}^{n+1} = -\mathbf{R}_{j,k}^{n}. \tag{2.12}$$

Using (2.7)-(2.9), we write (2.12) as

$$LL_{j,k}\Delta \mathbf{V}_{j,k-1}^{n+1}+L_{j,k}\Delta \mathbf{V}_{j-1,k}^{n+1}+\left(\frac{\beta_{j,k}}{\Delta t}+\Delta_{j,k}\right)\Delta \mathbf{V}_{j,k}^{n+1}+U_{j,k}\Delta \mathbf{V}_{j+1,k}^{n+1}+UU_{j,k}\Delta \mathbf{V}_{j,k-1}^{n+1}=-\mathbf{R}_{j,k}^{n},$$

where

$$LL_{j,k} = \frac{\partial \mathbf{R}}{\partial \mathbf{V}}\Big|_{j,k-1}^{n} = \begin{bmatrix} 0 & 0 & \frac{-\rho}{2\Delta y} \\ 0 & \frac{-u_{j,k}}{2\Delta y}-\frac{\nu}{\Delta y^2} & 0 \\ \frac{-1}{2\rho\Delta y} & 0 & \frac{-u_{j,k}}{2\Delta y}-\frac{\nu}{\Delta y^2} \end{bmatrix},$$

$$L_{j,k} = \frac{\partial \mathbf{R}}{\partial \mathbf{V}}\Big|_{j-1,k}^{n} = \begin{bmatrix} 0 & 0 & \frac{-\rho}{2\Delta x} \\ 0 & \frac{-u_{j,k}}{2\Delta x}-\frac{\nu}{\Delta x^2} & 0 \\ \frac{-1}{2\rho\Delta x} & 0 & \frac{-u_{j,k}}{2\Delta x}-\frac{\nu}{\Delta x^2} \end{bmatrix},$$

$$\Delta_{j,k} = \frac{\partial \mathbf{R}}{\partial \mathbf{V}}\Big|_{j,k}^{n} = \begin{bmatrix} 0 & 0 & 0 \\ 0 & \frac{2\nu(\Delta x^2+\Delta y^2)}{\Delta x^2\Delta y^2} & 0 \\ 0 & 0 & \frac{2\nu(\Delta x^2+\Delta y^2)}{\Delta x^2\Delta y^2} \end{bmatrix},$$

$$U_{j,k} = \frac{\partial \mathbf{R}}{\partial \mathbf{V}}\Big|_{j+1,k}^{n} = \begin{bmatrix} 0 & \frac{\rho}{2\Delta x} & 0 \\ \frac{\rho}{2\rho\Delta x} & \frac{u_{j,k}}{2\Delta x}-\frac{\nu}{\Delta x^2} & 0 \\ 0 & 0 & \frac{u_{j,k}}{2\Delta x}-\frac{\nu}{\Delta x^2} \end{bmatrix},$$

$$UU_{j,k} = \frac{\partial \mathbf{R}}{\partial \mathbf{V}}\Big|_{j,k+1}^{n} = \begin{bmatrix} 0 & 0 & \frac{\rho}{2\Delta y} \\ 0 & \frac{u_{j,k}}{2\Delta y}-\frac{\nu}{\Delta y^2} & 0 \\ \frac{1}{2\nu\Delta y} & 0 & \frac{u_{j,k}}{2\Delta y}-\frac{\nu}{\Delta y^2} \end{bmatrix}.$$

We enumerate the points on the grid starting from lower left corner. We move horizontally to the right, and once we reach the grid boundary, we move up and start from the left again. The structure of the resulting matrix is shown in the Table 2.1.

The equations above give the linear system for each time step. An $n_x \times n_x$ grid and five point stencil results in a $3n_x^2 \times 3n_x^2$ matrix. We halt the computations once the norms of R_2 and R_3 are sufficiently small. The simulation goes through three phases. The linear systems require only

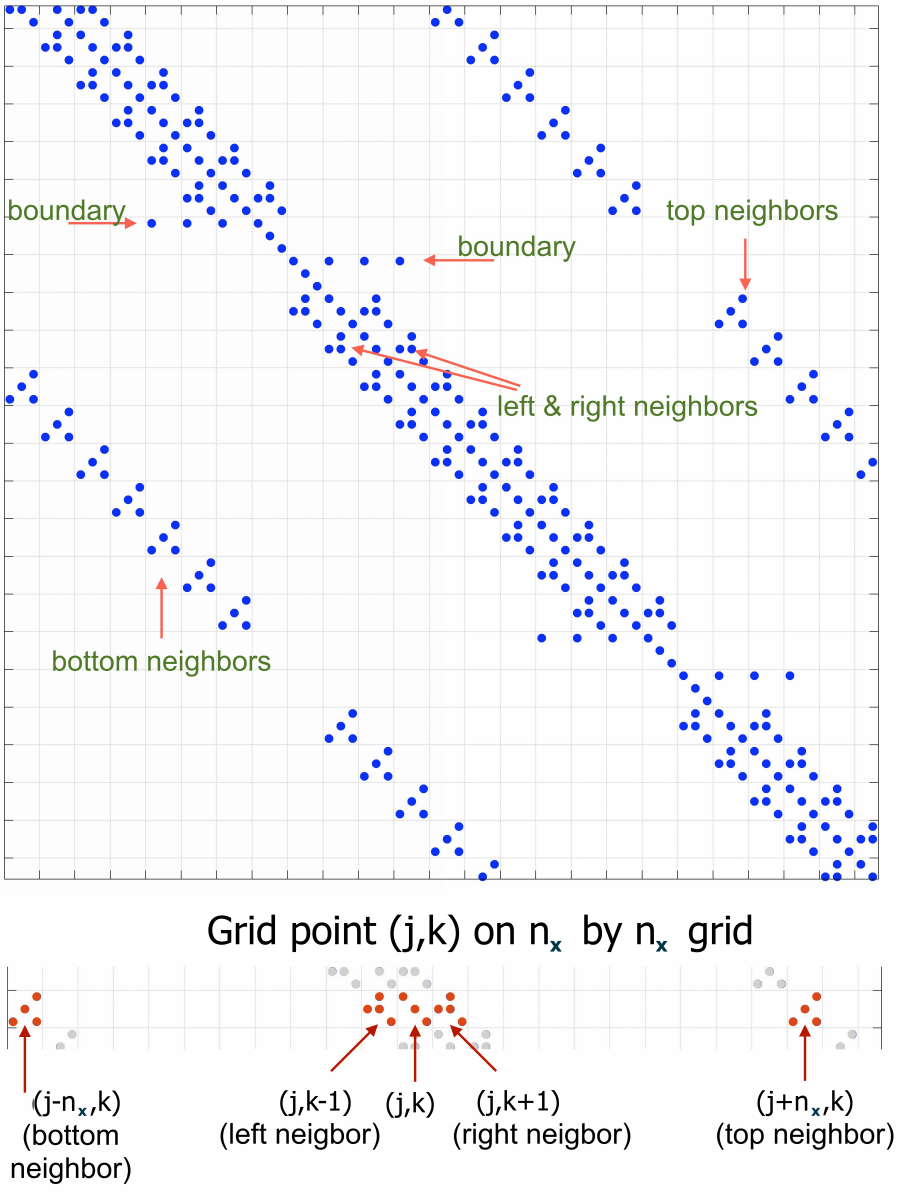


Figure 2.1: A fragment of an LDC matrix with $n_x = 11$. The dark dots represent non-zeros.

a small number of solver iterations in the initial phase. This phase is followed by a transition phase, in which the linear systems need many more iterations of the Krylov subspace solver to converge. Once the transition phase is completed, we enter the final phase, which continues until we reach a steady state. In the final phase, the number of Krylov subspace solver iterations per system is low and roughly constant. Once the residual norms for R_2 and R_3 are smaller than 10^{-1}, we are already in the final phase. The tolerance 10^{-1} is not sufficiently accurate for CFD simulation but for our purpose it is enough to provide understanding of the

behavior (runtimes, effectiveness of preconditioners) of the applied numerical algorithms for this problem. The LDC simulation is the subject of detailed discussion in Table 6, where we use it to test various GPU-based implementation strategies for iterative solvers.

2.2 Topology Optimization

In order to minimize the compliance $\mathbf{C}(\mathbf{p})$, we use the Optimality Criterion (OC) algorithm [45] as an update scheme. The algorithm requires gradient information of the objective function and volume constraint. The minimization algorithm stops when the maximum change in the design variables falls below a prescribed threshold. The problems considered in this thesis use the density-based approach [100]. In this approach, the density in each finite element is constant. Let $\mathbf{F} = \left[\sqrt{\alpha_1}\mathbf{f}_1\ \sqrt{\alpha_2}\mathbf{f}_2\ \ldots\ \sqrt{\alpha_m}\mathbf{f}_m\right]$, where $\sqrt{\alpha_j}\mathbf{f}_j$ is a weighted load case.

Let $\mathbf{U} = \left[\sqrt{\alpha_1}\mathbf{u}_1\ \sqrt{\alpha_2}\mathbf{u}_2\ \ldots\ \sqrt{\alpha_m}\mathbf{u}_m\right]$ be a matrix with the corresponding displacement vectors as its columns. Then the compliance $\mathbf{C}(\boldsymbol{\rho})$ is

$$\mathbf{C}(\boldsymbol{\rho}) = \sum_{l=1}^{m} \alpha_l \mathbf{f}_l^T \mathbf{u}_l = \operatorname{Tr}\left(\mathbf{F}^T\mathbf{U}\right) = \operatorname{Tr}\left(\mathbf{F}^T\mathbf{K}^{-1}(\boldsymbol{\rho})\,\mathbf{F}\right). \tag{2.13}$$

The gradient for an element e is

$$\nabla\mathbf{C}_{\boldsymbol{\rho}}^{(e)} = -\sum_{l=1}^{m} \alpha_l \mathbf{u}_l^T \frac{\partial\mathbf{K}}{\partial\boldsymbol{\rho}^{(e)}}\mathbf{u}_l = -\operatorname{Tr}\left(\mathbf{U}^T\frac{\partial\mathbf{K}}{\partial\boldsymbol{\rho}^{(e)}}\mathbf{U}\right) = -\operatorname{Tr}\left(\mathbf{F}^T\mathbf{K}^{-1}\frac{\partial\mathbf{K}}{\partial\boldsymbol{\rho}^{(e)}}\mathbf{K}^{-1}\mathbf{F}\right). \tag{2.14}$$

In every step of the optimization algorithm, we need to solve m linear systems $\mathbf{K}\mathbf{u}_l = \mathbf{f}_l$, $l = 1, 2, \ldots, m$, where m is the number of load cases. These systems are usually very large, and the number of load cases is often large as well. The number of linear solves can be reduced by applying a stochastic sampling strategy very similar to that used for the DOT problem [100]. We briefly summarize this approach.

Let $\mathbf{w} \in \mathbb{R}^n$ be an independent, identically distributed (i.i.d) vector with components taking on values ± 1, with equal probability for each value. Let $\mathbf{A} \in \mathbb{R}^{n\times n}$. Then

$$\mathbb{E}\left[v_j v_k\right] = \begin{cases} 0 & j \neq k, \\ 1 & j = k. \end{cases} \tag{2.15}$$

Using (2.15), we obtain

$$\mathbb{E}\left[\mathbf{w}^T\mathbf{A}\mathbf{w}\right] = \mathbb{E}\left[\sum_{k=1}^{n}\sum_{j=1}^{n} w_k \mathbf{A}_{kj} w_j\right] = \sum_{k=1}^{n}\sum_{j=1}^{n} \mathbf{A}_{kj}\mathbb{E}\left[w_k w_j\right] = \sum_{j=1}^{n} \mathbf{A}_{jj} = \operatorname{Tr}(\mathbf{A}).$$

If we take multiple vectors $\mathbf{w}_1, \mathbf{w}_2, \ldots, \mathbf{w}_q$, defined in the same way as $\mathbf{w}$, we get

$$\frac{1}{q}\sum_{k=1}^{q}\mathbf{w}_k^T\mathbf{A}\mathbf{w}_k \xrightarrow{q\to\infty} \mathbb{E}\left[\mathbf{w}^T\mathbf{A}\mathbf{w}\right] = \mathrm{Tr}\left(\mathbf{A}\right). \tag{2.16}$$

Next, we incorporate (2.16) into (2.13) and (2.14) to get

$$\mathbf{C}\left(\boldsymbol{\rho}\right) = \mathrm{Tr}\left(\mathbf{F}^T\mathbf{K}^{-1}\mathbf{F}\right) = \mathbb{E}\left[\mathbf{w}^T\mathbf{F}^T\mathbf{K}^{-1}\mathbf{F}\mathbf{w}\right] = \mathbb{E}\left[\left(\mathbf{F}\mathbf{w}\right)^T\mathbf{K}^{-1}\left(\mathbf{F}\mathbf{w}\right)\right],$$
$$\nabla\mathbf{C}_{\boldsymbol{\rho}}^{(e)} = -\mathrm{Tr}\left(\mathbf{F}^T\mathbf{K}^{-1}\frac{\partial\mathbf{K}}{\partial\boldsymbol{\rho}^{(e)}}\mathbf{K}^{-1}\mathbf{F}\right) = -\mathbb{E}\left[\left(\mathbf{F}\mathbf{w}\right)^T\mathbf{K}^{-1}\frac{\partial\mathbf{K}}{\partial\boldsymbol{\rho}^{(e)}}\mathbf{K}^{-1}\left(\mathbf{F}\mathbf{w}\right)\right].$$

In practice, we use multiple realizations of the vector $\mathbf{w}$ and we take the average (see (2.16)). More details on this method and estimates for the variance and standard deviation can be found in [100].

The expression $\left(\mathbf{F}\mathbf{w}\right)^T\mathbf{K}^{-1}\left(\mathbf{F}\mathbf{w}\right)$ is a matrix quadratic form ($\mathbf{K}$ is SPD). In Table 4 we estimate the quadratic form using the Conjugate Gradient (CG) algorithm in addition to the randomized approach. We also apply Krylov subspace recycling; the stiffness matrix changes only slightly for every optimization step. It turns out that recycling in this case speeds up the convergence of linear systems.

2.3 Diffuse Optical Tomography

2.3.1 Problem description

The mathematical model for diffuse optical tomography is based on energy conservation equations. While posed in the frequency domain, the model is given by the equation [9, 28]

$$-\nabla\cdot\left(D(\mathbf{x})\nabla\phi(\mathbf{x})\right) + \mu(\mathbf{x})\phi(\mathbf{x}) + \frac{i\omega}{\nu}\tilde{I} = 0, \tag{2.17}$$

where $\mathbf{x} = (x_1, x_2, x_3)^T$, $-a \le x_1 \le a$, $-b \le x_2 \le b$, $0 \le x_3 \le c$, ω is the frequency modulation of light, and ν is the speed of light in the medium of interest, $\phi(\mathbf{x})$ is the photon flux obtained from the source $g(\mathbf{x})$, $D(\mathbf{x})$ is the diffusion (scattering) coefficient and $\mu(\mathbf{x})$ is the absorption coefficient. $\phi(\mathbf{x}) = 0$ for $0 \le x_3 \le c$ and either $x_1 = \pm a$ or $x_2 = \pm b$, $0.25\phi(\mathbf{x}) + \frac{D(\mathbf{x})}{2}\frac{\partial\phi(\mathbf{x})}{\partial\xi} = g(x)$ for $x_3 = 0$ and $x_3 = c$ (here ξ denotes outward unit normal).

In our case, we assume that $D(\mathbf{x})$ is known (well specified), and we are solving only for $\mu(\mathbf{x})$. Jumps in the absorption and scattering indicate the existence of anomalies in the medium. We

parametrize $\mu(\mathbf{x})$ using a modest number of parameters, i.e., $\mu(\mathbf{x}) = \mu(\mathbf{x};\mathbf{p})$ to avoid solving for absorption at every grid point.

We use the parametric level set (PaLS) method [28] to set up the inverse problem. This approach leads to a significant reduction in the dimension of the parameter space, and also has the benefit of regularizing the problem, so no additional regularization is needed.

Let φ be a smooth, compactly supported radial basis function, with $\varphi : \mathbb{R}^+ \to \mathbb{R}$, and let γ be a small positive real number. By $\|x\|^\dagger$ we understand $\sqrt{\|x\|^2 + \gamma^2}$. Let $\mathbf{p}_-$ be a vector of unknown values, which consist of α_j (expansion coefficients), β_j (dilation coefficients), and χ_j (the center location coordinates). The PaLS function η is given by

$$\eta(\mathbf{x}, \mathbf{p}_-) = \sum_{j=1}^{m_0} \alpha_j \varphi\left(\|\beta_j\,(\mathbf{x} - \chi_j)\,\|^\dagger\right).$$

We define the level set parametrization of $\mu(\mathbf{x}, \mathbf{p})$ to be

$$\mu(\mathbf{x};\mathbf{p}) = \mu_{in}(\mathbf{x}) H_\varepsilon\left(\eta\left(\mathbf{x}, \mathbf{p}_-\right) - c\right) + \mu_{out}(\mathbf{x})\left[1 - H_\varepsilon\left(\eta\left(\mathbf{x}, \mathbf{p}_-\right) - c\right)\right],$$

where $H_\varepsilon(r)$ is a scalar-valued continuous approximation to Heaviside function, and c is cutoff value for the level set. The parameter vector $\mathbf{p}$ is a concatenation of $\mathbf{p}_-$ and the parameters that come with $\mu_{in}(\mathbf{x})$ and $\mu_{out}(\mathbf{x})$. We assume that the length of $\mathbf{p}$ is n_p.

We observe that $\mu(\mathbf{x};\mathbf{p})$ equals $\mu_{in}(\mathbf{x})$ if $\mathbf{x}$ is inside the region defined by the c-level set and $\mu_{out}(\mathbf{x})$ otherwise. The function $\mu(\mathbf{x};\mathbf{p})$ is thus almost piecewise constant, with sharp yet smooth transitions from the background (domain) to the anomaly. Theoretical background material on DOT using PaLS can be found in [28].

Next, we discretize the PDE using finite difference scheme with the parameters of the absorption carrying over to the discretized equations. Let n_s, n_d and n_ω be the number of sources, number of detectors, and number of frequencies, respectively. The resulting computed measurement for every source term b_j, $j = 1, \ldots, n_s$ and every frequency ω_k, $k = 1, 2, \ldots, n_\omega$ is

$$\mathbf{m}_j\left(\omega_k, \mathbf{p}\right) = \mathbf{C}^T\left(i\frac{\omega_k}{\nu}\mathbf{E} - \mathbf{A}^{-1}\left(\mathbf{p}\right)\right)\mathbf{b}_j,$$

where $\mathbf{C}^T$ corresponds to the detectors, $\mathbf{A}\left(\mathbf{p}\right)$ comes from the discretization of the absorption and diffusion coefficients, and $\mathbf{E}$ corresponds to the frequency terms. $\mathbf{E}$ is a singular matrix (identity matrix with zero rows in the boundaries $x_3 = 0$ and $x_3 = c$). Note that $\mathbf{m}_j\left(\omega_k, \mathbf{p}\right) \in \mathbb{C}^{n_d}$. In our experiments, we use only the zero frequency, $\omega_k = 0$, so, we can simplify notation, i.e., $\mathbf{m}_j\left(\mathbf{p}\right) := \mathbf{m}_j\left(\omega_k, \mathbf{p}\right)$.

Let us define the residual function as the difference between the predictions generated by the model, $\mathbf{m}_1\left(\mathbf{p}\right)$, $\ldots$, $\mathbf{m}_{n_s}\left(\mathbf{p}\right)$, and the measurements observed at the detectors, $\mathbf{d}_1$, $\ldots$, $\mathbf{d}_{n_s}$. Our goal is to find the vector $\mathbf{p}$ that minimizes the norm of the residual function. Since the data contains noise, we stop the minimization process at the noise level.

The residual function is given as

$$\mathbf{r}(\mathbf{p}) = \begin{bmatrix} \mathbf{r}_1(\mathbf{p}) \\ \mathbf{r}_2(\mathbf{p}) \\ \vdots \\ \mathbf{r}_{n_s}(\mathbf{p}) \end{bmatrix} = \begin{bmatrix} \mathbf{m}_1(\mathbf{p}) - \mathbf{d}_1 \\ \mathbf{m}_2(\mathbf{p}) - \mathbf{d}_2 \\ \vdots \\ \mathbf{m}_{n_s}(\mathbf{p}) - \mathbf{d}_{n_s} \end{bmatrix} = \begin{bmatrix} \mathbf{C}^T\mathbf{A}^{-1}(\mathbf{p})\,\mathbf{b}_1 - \mathbf{d}_1 \\ \mathbf{C}^T\mathbf{A}^{-1}(\mathbf{p})\,\mathbf{b}_2 - \mathbf{d}_2 \\ \vdots \\ \mathbf{C}^T\mathbf{A}^{-1}(\mathbf{p})\,\mathbf{b}_{n_s} - \mathbf{d}_{n_s}, \end{bmatrix}, \tag{2.18}$$

and the resulting minimization problem is

$$\min_{\mathbf{p}} \frac{1}{2}\|\mathbf{r}(\mathbf{p})\|^2. \tag{2.19}$$

The Jacobian of $\mathbf{r}(\mathbf{p})$ is defined as

$$\mathbf{J} = \frac{\partial \mathbf{r}(\mathbf{p})}{\partial \mathbf{p}} = \left[\frac{\partial \mathbf{r}(\mathbf{p})}{\partial \mathbf{p}_1}, \frac{\partial \mathbf{r}(\mathbf{p})}{\partial \mathbf{p}_2}, \ldots, \frac{\partial \mathbf{r}(\mathbf{p})}{\partial \mathbf{p}_{n_p}} \right],$$

,

and its components are computed as

$$\mathbf{J}_{jk}(\mathbf{p}) = \frac{\partial}{\partial \mathbf{p}_k}\left(\mathbf{C}^T\mathbf{A}^{-1}(\mathbf{p})\,\mathbf{b}_j\right) = -\mathbf{C}^T\mathbf{A}^{-1}(\mathbf{p})\,\frac{\partial \mathbf{A}(\mathbf{p})}{\partial \mathbf{p}_k}\mathbf{A}^{-1}(\mathbf{p})\,\mathbf{b}_j \in \mathbb{R}^{n_d}.$$

To evaluate $\mathbf{r}(\mathbf{p})$, we either solve n_s linear systems $\mathbf{A}(\mathbf{p})\,\mathbf{x}_j = \mathbf{b}_j$ for $j = 1, \ldots, n_s$ or n_d linear systems $\mathbf{A}^T(\mathbf{p})\,\mathbf{y}_k = \mathbf{c}_k$ with $k = 1, \ldots, n_d$.

We search for a minimum in (2.19) using TREGS method [29]. TREGS is a trust region method combined with regularized Gauss-Newton steps for the trust region model problem. The method aims to minimize the total number of function and Jacobian evaluations.

2.3.2 Randomized approach

Finding the approximate minimal solution requires solving a large number of linear systems, since we must solve $n_s \cdot n_\omega$ or $n_d \cdot n_\omega$ linear systems for every function evaluation. If the Jacobian is needed, we need to solve $n_s \cdot n_d \cdot n_\omega$ linear systems. A randomized approach allows us to decrease the total number of systems to be solved through simultaneous random sources and detectors [53, 47, 9]. We follow the approach given in [9].

We reformulate (2.18) to write it in a matrix form. Next, we introduce random vectors and we derive the formula to estimate the Frobenius norm of this matrix.

We define

$$\mathbf{R}(\mathbf{p}) = [\mathbf{r}_1(\mathbf{p})\ \ \mathbf{r}_2(\mathbf{p})\ \ \ldots\ \ \mathbf{r}_{n_d}(\mathbf{p})] = \mathbf{C}^T\mathbf{A}^{-1}(\mathbf{p})\,\mathbf{B} - \mathbf{D}. \tag{2.20}$$

Let $\mathbf{v} = [v_1, \ldots, v_{n_d}]^T \in \{-1, 1\}^{n_d}$ and $\mathbf{w} = [w_1, \ldots, w_{n_s}]^T \in \{-1, 1\}^{n_s}$ be vectors with all components being independent identically distributed (i.i.d) uniformly from $\{+1, -1\}$.

Using (2.15) we obtain

$$\begin{aligned}
\mathbb{E}\left[\left(\mathbf{v}^T\mathbf{R}\mathbf{w}\right)^2\right] &= \mathbb{E}\left[\left(\sum_{j=1}^{n_s}\sum_{k=1}^{n_d} R_{kj}v_k w_j\right)\left(\sum_{l=1}^{n_s}\sum_{s=1}^{n_d} R_{sl}w_l v_s\right)\right] \\
&= \sum_{j=1}^{n_s}\sum_{k=1}^{n_d}\sum_{l=1}^{n_s}\sum_{s=1}^{n_d} R_{kj}R_{sl}\mathbb{E}\left[v_k v_s w_j w_l\right] \\
&= \sum_{j=1}^{n_s}\sum_{k=1}^{n_d}\sum_{l=1}^{n_s}\sum_{s=1}^{n_d} R_{kj}R_{sl}\mathbb{E}\left[v_k v_s\right]\mathbb{E}\left[w_j w_l\right] \\
&= \sum_{j=1}^{n_s}\sum_{k=1}^{n_d} R_{kj}^2 = \|\mathbf{R}\|_F^2,
\end{aligned} \tag{2.21}$$

where $\|\cdot\|_F$ denotes the Frobenius norm.

Let $\mathbf{V} \in \{-1, 1\}^{n_d \cdot s_s}$ and $\mathbf{W} \in \{-1, 1\}^{n_s \cdot s_d}$ be matrices with i.i.d columns uniformly from $\{-1, 1\}$. Here, s_s and s_d denote the number of random vectors for simultaneous random sources and simultaneous random detectors, respectively. Then, following [9]

$$\frac{1}{s_d s_s}\mathbb{E}\left[\|\mathbf{V}^T\mathbf{R}\mathbf{W}\|_F^2\right] = \|\mathbf{R}\|_F^2. \tag{2.22}$$

The estimate (2.22) allows us to drastically reduce the number of linear solves. **WB** and **CV** can be understood as simultaneous detectors and simultaneous sources. Similar estimates are used for the Jacobian [9].

2.4 Error estimation and grid adaptation in CFD

Truncation error and error transport equation

In CFD, the truncation error can be defined using the Generalized Truncation Error Expression [74] as

$$\mathbf{L}_h\left(\mathbf{I}^h\mathbf{u}\right) = \mathbf{I}^h\mathbf{L}\left(\mathbf{u}\right) + \tau_h\left(\mathbf{u}\right), \tag{2.23}$$

where $\mathbf{L}\left(\cdot\right)$ is a set of continuous governing equations, $\mathbf{L}_h\left(\cdot\right)$ is a set of consistently discretized equations, $\mathbf{u}$ is a continuous function, $\tau_h\left(\mathbf{u}\right)$ is the truncation error, and h is the characteristic size of the grid. Furthermore, $\mathbf{I}^h$ is a transfer operator from the space in which the continuous problem is posed to the space of discretized solutions.

Let $\tilde{\mathbf{u}}$ be a solution to a system of differential equations, $\mathbf{L}(\mathbf{u}) = \mathbf{0}$. Then, $\mathbf{L}(\tilde{\mathbf{u}}) = 0$, and we define the truncation error as

$$\tau_h(\tilde{\mathbf{u}}) = \mathbf{L}_h\left(\mathbf{I}^h\tilde{\mathbf{u}}\right). \tag{2.24}$$

The error transport equation (ETE) relates the truncation error to the local solution (discretization) error, defined as $\varepsilon_h = \mathbf{u}_h - \mathbf{I}^h\tilde{\mathbf{u}}$. We start the derivation of the ETE by expanding $\mathbf{L}_h(\mathbf{I}^h\tilde{\mathbf{u}})$ about the exact solution of the discrete equations, $\mathbf{u}_h$,

$$\mathbf{L}_h\left(\mathbf{I}^h\tilde{\mathbf{u}}\right) = \mathbf{L}_h(\mathbf{u}_h) - \left.\frac{\partial \mathbf{L}_h}{\partial \mathbf{u}}\right|_{\mathbf{u}_h} \varepsilon_h + O\left(\|\varepsilon_h\|^2\right). \tag{2.25}$$

Since $\mathbf{u}_h$ is the exact solution of the discretized problem, $\mathbf{L}_h(\mathbf{u}_h) = 0$, we get

$$\left.\frac{\partial \mathbf{L}_h}{\partial \mathbf{u}}\right|_{\mathbf{u}_h} \varepsilon_h = -\tau_h(\tilde{\mathbf{u}}) + O\left(\|\varepsilon_h\|^2\right). \tag{2.26}$$

To obtain a second order accuracy estimate of the truncation error, the Jacobian matrix $\left.\frac{\partial \mathbf{L}_h}{\partial \mathbf{u}}\right|_{\mathbf{u}_h}$ must be the full second order linearization of the discrete operator.

Adjoint methods and error estimation

Adjoint methods have originally been developed for design optimization problems (such as topology optimization), as a way to evaluate sensitivity of a functional to a set of design parameters. The dot product of the solution to the adjoint problem (which corresponds to the sensitivity of the functional to perturbations in the operator) and local truncation error can be used as an adaptation indicator; in this case adaptation is done only in the areas of the domain that contribute to the error in the functional of interest.

Let $\mathbf{J}_h$ be a discrete solution functional. We expand the functional about the exact discrete solution, $\mathbf{u}_h$,

$$\mathbf{J}_h(\mathbf{I}^h\tilde{\mathbf{u}}) = \mathbf{J}_h(\mathbf{u}_h) - \left.\frac{\partial \mathbf{J}_h}{\partial \mathbf{u}}\right|_{\mathbf{u}_h} \varepsilon_h + O\left(\|\varepsilon_h\|^2\right). \tag{2.27}$$

Let $\varepsilon_{\mathbf{J}_h}$ be the error in the functional: $\varepsilon_{\mathbf{J}_h} = \mathbf{J}_h(\mathbf{I}^h\tilde{\mathbf{u}}) - \mathbf{J}_h(\mathbf{u}_h)$. Using (2.27), this gives

$$\varepsilon_{\mathbf{J}_h} = \mathbf{J}_h(\mathbf{I}^h\tilde{\mathbf{u}}) - \mathbf{J}_h(\mathbf{u}_h) = -\left.\frac{\partial \mathbf{J}_h}{\partial \mathbf{u}}\right|_{\mathbf{u}_h} \varepsilon_h + O\left(\|\varepsilon_h\|^2\right). \tag{2.28}$$

Let λ be the discrete adjoint variables which satisfy linear adjoint equation

$$\left[\left.\frac{\partial \mathbf{L}_h}{\partial \mathbf{u}}\right|_{\mathbf{u}_h}\right]^T \lambda = \left[\left.\frac{\partial \mathbf{J}_h}{\partial \mathbf{u}}\right|_{\mathbf{u}_h}\right]^T. \tag{2.29}$$

Then, substituting (2.26) into (2.28), we obtain

$$\varepsilon_{\mathbf{J}_h} = \lambda^T \tau_h \left(\tilde{\mathbf{u}}\right) + O\left(\|\varepsilon_h\|^2\right). \tag{2.30}$$

$\frac{\partial \mathbf{L}_h}{\partial \mathbf{u}}\Big|_{\mathbf{u}_h}$ is already available if we are using an implicit solver; however, full linearization is needed to obtain a second order error estimate. Since the adjoint variables give the sensitivities to perturbations, the product of the local adjoint variable and the local truncation error is a good candidate for an adaptation indicator.

In CFD we often need estimates for the error in several functionals. Solving many adjoint problems might be costly. These errors, however, can be estimated *in bulk* using a clever mathematical approach. We first recall that the error in the functional can be approximated as

$$\varepsilon_{\mathbf{J}_h} \approx \lambda^T \tau_h \left(\tilde{\mathbf{u}}\right).$$

Since λ is a solution to (2.29), we can write

$$\varepsilon_{\mathbf{J}_h} \approx \frac{\partial \mathbf{J}_h}{\partial \mathbf{u}}\bigg|_{\mathbf{u}_h} \left[\frac{\partial \mathbf{L}_h}{\partial \mathbf{u}}\bigg|_{\mathbf{u}_h}\right]^{-1} \tau_h \left(\tilde{\mathbf{u}}\right). \tag{2.31}$$

Note that $\left[\frac{\partial \mathbf{L}_h}{\partial \mathbf{u}}\Big|_{\mathbf{u}_h}\right]^{-1} \tau_h \left(\tilde{\mathbf{u}}\right)$ is, in fact, the solution to the ETE. Hence, the the correction to the functional can be computed as the inner product of the adjoint solution with the truncation error or as the inner product of the ETE solution with the functional linearization,

$$\varepsilon_{\mathbf{J}_h} \approx \overbrace{-\frac{\partial \mathbf{J}_h}{\partial \mathbf{u}}\bigg|_{\mathbf{u}_h} \underbrace{\left[\frac{\partial \mathbf{L}_h}{\partial \mathbf{u}}\bigg|_{\mathbf{u}_h}\right]^{-1} \tau_h(\tilde{\mathbf{u}})}_{\text{Error Transport Equations}}}^{\text{Sol. to Adjoint Problem, } \lambda^T}. \tag{2.32}$$

From a mathematical point of view, the quantity represented by equation (2.32) is a matrix bilinear form, $\mathbf{c}^T\mathbf{a}^{-1}\mathbf{b}$, which can be evaluated using BiCG or another Krylov subspace solver that involves Lanczos bi-diagonalization. We present an adaptation indicator based on this bilinear form in Table 4.

2.5 GenIDLEST

GenIDLEST has been used to study propulsion, biological flows, and energy related applications that involve complex multi-physics flows. GenIDLEST provides various turbulence

modeling options, such as RANS (Reynolds-Averaged Navier Stokes), LES (Large Eddy Simulation), and DES (Detached Eddy Simulation), see [7] for details. In this thesis, we consider two important test problems in GenIDLEST.

Turbulent Channel Flow

Turbulent channel flow is a classic CFD problem that has been studied extensively. It is a standard problem for testing new codes and methods. In this problem, a fluid is moving through a channel with a rectangular cross-section. We model the problem using the three-dimensional Navier-Stokes equations. We create a Cartesian, three dimensional grid with a single grid block of $64 \times 64 \times 64$ grid cells. In the discretization, we use seven point stencil, which means that the state of the grid cell depends on the state of the neighbors that share a face with it. In order to resolve the boundary layer, we use a finer grid near the boundaries. We also fix the pressure gradient in the direction of the flow to balance wall friction. In order to simulate an infinitely long channel, we apply periodic boundary conditions in the flow direction. We start the simulation with given perturbations in the flow to trigger the onset of turbulence. The initial perturbations are based on the mean turbulent channel flow profile.

We run the simulation until the flow turbulence has become statistically stationary in time. We set the skin friction Reynolds number Re_τ to 180, which is based on channel half width and wall friction velocity. We use this problem to test the hybrid recycling approach that is the subject of Table 5.

Flow Through Porous Media

The Flow Through Porous Media test problem was implemented using the Immersed Boundary Method (IBM) to resolve the porous structure, which is constructed using the stochastic reconstruction procedure [66]. In this example, we use a 2D (background) grid with 2.56 million Cartesian computational cells, structured into 16 grid blocks ($100 \times 800 \times 2$ cells each) to simulate a bulk flow Reynolds number Re_b of 10^{-4}. We need only two neighboring computational cells in the horizontal direction to update flow values for a given cell. The porous medium has wall boundaries at the top and the bottom (max and min y). The inlet is placed on the left boundary and the outlet is placed on the right boundary. The pressure coefficient matrix remains constant for every time step, as the IBM related flux corrections at the immersed boundaries are not applied. Again, we use this problem to test our hybrid recycling approach in Table 5.

Chapter 3

Krylov Methods and Preconditioners

3.1 Krylov spaces

Let $\mathbf{x}_0$ be an initial guess for the solution of system (1.1), and let $\mathbf{r}_0 = \mathbf{b} - \mathbf{A}\mathbf{x}_0$ be the residual. Krylov subspace methods update the approximate solution at every iteration. In step j, we form $\mathbf{x}_j = \mathbf{x}_0 + \mathbf{z}_j$, with $\mathbf{z}_j \in \mathcal{K}^j(\mathbf{A}, \mathbf{r}_0)$, where $\mathcal{K}^j(\mathbf{A}, \mathbf{r}_0) = \text{span}\{\mathbf{r}_0, \mathbf{A}\mathbf{r}_0, \ldots, \mathbf{A}^{j-1}\mathbf{r}_0\}$ is the Krylov subspace associated with $\mathbf{A}$ and $\mathbf{r}_0$ [95]. We find $\mathbf{z}_j$ using projection. The projection is specific to the method. A standard choice is to select a projection that minimizes the residual norm $\|\mathbf{b} - \mathbf{A}\mathbf{x}_j\|$ over all $\mathbf{z}_j \in \mathcal{K}^j(\mathbf{A}, \mathbf{r}_0)$ in each step. We refer to this approach as *minimum residual norm approach*. It leads to methods such as MINRES [68], GCR [34],and GMRES [81]. Minimal residual norm is equivalent to

$$\mathbf{r}_j \perp \mathcal{K}^j(\mathbf{A}, \mathbf{A}\mathbf{r}_0). \tag{3.1}$$

If the matrix $\mathbf{A}$ is symmetric and positive definite, another popular approach is to minimize the $\mathbf{A}$-norm of the error, i.e., minimize $\|\mathbf{x}_j - \tilde{\mathbf{x}}\|_{\mathbf{A}}$ where $\tilde{\mathbf{x}}$ is the solution to (1.1). An example of a method that uses this approach is the Conjugate Gradient (CG) method [49, 57]. Finding $\mathbf{z}_j$ such that the error is minimized in the $\mathbf{A}$-norm is equivalent to enforcing

$$\mathbf{r}_j \perp \mathcal{K}^j(\mathbf{A}, \mathbf{r}_0), \tag{3.2}$$

see [80].

The convergence of the Krylov subspace methods depends in a large part on the eigenvalue distribution. For a detailed discussion, see [44, 43, 35, 92, 80]. The importance of the eigenvalues in convergence bounds is, however, decreased for non-normal matrices, see [44, 35, 63].

3.2 Krylov subspace methods

3.2.1 GMRES

GMRES (Generalized Minimal Residual Method) was first proposed in [81]. The method is based on the Arnoldi recurrence

$$\mathbf{A}\mathbf{V}_j = \mathbf{V}_{j+1}\mathbf{H}_{j+1,j}, \tag{3.3}$$

where $\mathbf{V}_j$ is a matrix with orthonormal columns, and the span of the columns of $\mathbf{V}_j$ is equal to $\text{span}\{\mathbf{r}_0, \mathbf{A}\mathbf{r}_0, \ldots, \mathbf{A}^{j-1}\mathbf{r}_0\}$, with $\mathbf{v}_1 = \frac{\mathbf{r}_0}{\|\mathbf{r}_0\|}$. The matrix $\mathbf{H}_{j+1,j}$ is a $(j+1) \times j$ upper Hessenberg matrix. Since $\mathbf{z}_j \in \text{span}\{\mathbf{v}_1, \mathbf{v}_2, \ldots, \mathbf{v}_j\}$, $\mathbf{z}_j = \mathbf{V}_j\mathbf{y}_j$ for the $j \times 1$ vector $\mathbf{y}_j$ that minimizes the norm of the residual $\mathbf{r}_j = \mathbf{b} - \mathbf{A}\mathbf{x}_j = \mathbf{b} - \mathbf{A}(\mathbf{x}_0 + \mathbf{V}_j\mathbf{y}_j)$. The resulting least squares problem, $\min_{\mathbf{y}_j} \|\|\mathbf{r}_0\|\mathbf{e}_1 - \mathbf{H}_{j+1,j}\mathbf{y}_j\|$, where $\mathbf{e}_1 = [1, 0, \ldots, 0]^T$, relies on the Arnoldi recurrence.

We compute the QR decomposition of $\mathbf{H}_{j+1,j}$. Givens rotations give $\mathbf{H}_{j+1,j} = \mathbf{Q}_{j+1,j}\mathbf{R}_{j,j}$. The approximate solution in step j is $\mathbf{x}_j = \mathbf{x}_0 + \mathbf{V}_j\mathbf{R}_{j,j}^{-1}\mathbf{Q}_{j,j+1}\|\mathbf{r}_0\|\mathbf{e}_1$.

GMRES comes with high computational and storage costs. The Arnoldi iteration uses the modified Gram-Schmidt procedure (denoted with red background in Algorithm 1) to generate a sequence of orthogonal vectors, which form the columns of $\mathbf{V}_j$. A new vector, $\mathbf{v}_{j+1}$, is explicitly orthogonalized against $\mathbf{v}_1, \mathbf{v}_2, \ldots, \mathbf{v}_j$. Thus, the computational cost increases quadratically with the iteration count, and the storage cost increases linearly until convergence is reached. To reduce the storage and computational requirements, GMRES is restarted after a prescribed number of iterations, and the last solution is used as the new initial guess; the restarted method is called GMRES(m), where m is the restart frequency. Restarting, however, often results in an increase in the total number of iterations [80].

A non-restarted (full) GMRES is guaranteed to converge (see [80, Proposition 6.10]). The sequence of residual norms must be non-increasing but does not have to be monotonically decreasing [44]. Stagnation is a known issue for the restarted version [80]. To speed up convergence, we use preconditioners, discussed later in the current chapter.

In Table 6, we discuss the parallel implementation of Krylov subspace methods, including GMRES. Because of the Arnoldi recurrence, it is difficult to implement GMRES efficiently for parallel architectures. In line 7 of Algorithm 1 we create a new vector $\mathbf{w} = \mathbf{A}\mathbf{v}_l$, where l is the current number of vectors in the Krylov subspace. In the k-loop (lines 8 to 11) we subtract the components of $\mathbf{w}$ in the direction of the vectors $\mathbf{v}_1, \ldots, \mathbf{v}_l$. This has to be done in order, one after the other. In line 9 we use $\mathbf{w}$ computed in line 10 in the previous iteration of the k-loop, and we cannot simply execute the k-iterations concurrently.

3.2.2 Conjugate Gradient

The Conjugate Gradient method (CG) [49, 57, 73] is used to find an approximate solution of (1.1), if the matrix $\mathbf{A}$ is symmetric positive definite. In this case $\mathbf{H}_{j+1,j}$ in (3.3) reduces to a tridiagonal matrix, for which the top left $j \times j$ minor is a symmetric matrix.

The recurrence relation is then written as

$$\mathbf{A}\mathbf{V}_j = \mathbf{V}_{j+1}\mathbf{T}_{j+1,j} = \mathbf{V}_j\mathbf{T}_{j,j} + t_{j+1,j}\mathbf{v}_{j+1}\mathbf{e}_j^T, \tag{3.4}$$

which is known as the Lanczos recurrence.

CG minimizes the $\mathbf{A}$-norm of the error. Based on (3.2), this is equivalent to finding $\mathbf{z}_j$ such that $\mathbf{r}_j = \mathbf{b} - \mathbf{A}(\mathbf{x}_0 + \mathbf{z}_j) \perp \mathcal{K}^j(\mathbf{A}, \mathbf{r}_0)$. Since $\mathbf{z}_j \in \text{span}\{\mathbf{v}_1, \mathbf{v}_2, \ldots, \mathbf{v}_j\}$, hence $\mathbf{z}_j = \mathbf{V}_j\mathbf{y}_j$ for a $j \times 1$ vector $\mathbf{y}_j$. Thus

$$\begin{aligned}
&\mathbf{V}_j^T(\mathbf{r}_0 - \mathbf{A}\mathbf{V}_j\mathbf{y}_j) = \mathbf{0} \\
&\Leftrightarrow \mathbf{V}_j^T(\|\mathbf{r}_0\|\mathbf{v}_1 - \mathbf{A}\mathbf{V}_j\mathbf{y}_j) = \mathbf{0} \\
&\Leftrightarrow \|\mathbf{r}_0\|\mathbf{e}_1 - \mathbf{V}_j^T\mathbf{A}\mathbf{V}_j\mathbf{y}_j = \mathbf{0} \\
&\Leftrightarrow \|\mathbf{r}_0\|\mathbf{e}_1 - \mathbf{V}_j^T\left(\mathbf{V}_j\mathbf{T}_{j,j} + t_{j+1,j}\mathbf{v}_{j+1}\mathbf{e}_j^T\right)\mathbf{y}_j = \mathbf{0} \\
&\Leftrightarrow \|\mathbf{r}_0\|\mathbf{e}_1 - \mathbf{T}_{j,j}\mathbf{y}_j = \mathbf{0}.
\end{aligned}$$

Hence, we obtain $\mathbf{y}_j = \|\mathbf{r}_0\|\mathbf{T}_{j,j}^{-1}\mathbf{e}_1$. In order to find $\mathbf{T}_{j,j}^{-1}$, we use the $\mathbf{LDL}^T$ decomposition of the matrix $\mathbf{T}_{j,j}$. The decomposition is not computed g explicitly [95]. The algorithm is shown in Algorithm 2.

In CG, we update the solution $\mathbf{x}_j$ based on short recurrences. Unlike for GMRES, in CG we do not store all the vectors $\mathbf{v}_1, \mathbf{v}_2, \ldots, \mathbf{v}_j$. In exact arithmetic, the columns of $\mathbf{V}_j$ would be orthogonal. However, due to round-off errors this is generally not the case in floating point arithmetic; the vectors can even become linearly dependent [87, 89].

3.2.3 BiCG and BiCGstab

In BiCG [36], we use an oblique projection associated with the Krylov subspace derived from an arbitrary dual (adjoint) system, where by arbitrary dual system we understand the system $\mathbf{A}^T\mathbf{y} = \mathbf{c}$ with an arbitrary vector $\mathbf{c}$. Let $\mathbf{y}_0$ be an initial guess for the dual system. Let $\mathbf{s}_0 = \mathbf{c} - \mathbf{A}^T\mathbf{y}_0$, be the initial residual for the dual system. Let the columns of $\mathbf{W}_j$ form a basis for $\text{span}\{\mathbf{s}_0, \mathbf{A}^T\mathbf{s}_0, (\mathbf{A}^T)^2, \ldots, (\mathbf{A}^T)^{j-1}\mathbf{s}_0\}$; there exist multiple choices for the normalization of $\mathbf{w}_1$.

BiCG is based on the two-sided Lanczos recurrence [76]. By construction, the oblique projection results in $\mathbf{W}_j^T\mathbf{V}_j = \mathbf{D}_j$, where $\mathbf{D}_j$ is a diagonal matrix. The two-sided Lanczos algorithm

computes the following recurrences

$$\begin{aligned} \mathbf{A}\mathbf{V}_j &= \mathbf{V}_{j+1}\mathbf{T}_{j+1,j} = \mathbf{V}_j\mathbf{T}_{j,j} + t_{j+1,j}\mathbf{v}_{j+1}\mathbf{e}_j^T, \\ \mathbf{A}^T\mathbf{W}_j &= \mathbf{W}_{j+1}\mathbf{S}_{j+1,j} = \mathbf{W}_j\mathbf{S}_{j,j} + s_{j+1,j}\mathbf{w}_{j+1}\mathbf{e}_j^T, \\ \mathbf{S}_{j,j} &= \mathbf{T}_{j,j}^T, \end{aligned} \tag{3.5}$$

where $\mathbf{T}_{j,j}$ and $\mathbf{S}_{j,j}$ are tridiagonal matrices .

If the matrix $\mathbf{A}$ is symmetric and $\mathbf{s}_0 = \mathbf{r}_0$, the recurrence (3.5) reduces to (3.4).

To compute an approximate solution in BiCG, we use a projection that makes $\mathbf{r}_j$ orthogonal to $\mathbf{W}_j$, which results in

$$\mathbf{0} = \mathbf{W}_j^T\mathbf{r}_j = \mathbf{W}_j^T\left(\mathbf{r}_0 - \mathbf{A}\mathbf{V}_j\mathbf{y}_j\right) = \mathbf{W}_j^T\mathbf{r}_0 - \mathbf{W}_j^T\mathbf{A}\mathbf{V}_j\mathbf{y}_j.$$

By (3.5)

$$\mathbf{D}_j\mathbf{T}_{j,j}\mathbf{y}_j = \|\mathbf{r}_0\|\mathbf{e}_1.$$

Solution updates in BiCG are computed using an implicit LDU decomposition of the matrix $\mathbf{T}_{j,j}$ [80, section 7.3.1].

In the CG algorithm, we do not explicitly enforce orthogonality of the columns of the matrix $\mathbf{V}_j$. In BiCG, we do not explicitly enforce $\mathbf{V}_j^T\mathbf{W}_j = \mathbf{D}_j$ (so-called *bi-orthogonality*), therefore, we do not need to keep all the columns of $\mathbf{V}_j$ and $\mathbf{W}_j$. In fact, to find the solution update in iteration j, we only need the vectors from the iteration $j-1$. However, the drawback is that in floating point arithmetic, we are likely to lose the bi-orthogonality of $\mathbf{V}_j$ and $\mathbf{W}_j$ due to round-off errors [11]. Similarly as for the Lanczos vectors in CG, the columns of $\mathbf{W}_j$ can become linearly dependent.

The choice of the right hand side $\mathbf{c}$ in the dual system is arbitrary. The only requirement for the dual system is $\mathbf{r}_0^T\mathbf{s}_0 \neq 0$, because the method breaks down if $\mathbf{r_j} \perp \mathbf{s_j}$ [95]. Hence, BiCG can solve the main system (1.1) and the dual system (1.2) simultaneously, i.e., we can choose $\mathbf{s}_0 = \mathbf{c} - \mathbf{A}^T\mathbf{y}_0$, where $\mathbf{y}_0$ is an initial guess for the dual system. In Table 4, we use BiCG in this way – we solve main and dual system simultaneously – and hence, we apply a non-standard convergence criteria to ensure that the solutions of both systems have low residual norms (see Algorithm 3). Next, we briefly explain what motivates the alternative convergence criteria.

Table 3.1 shows convergence for a main and a dual systems taken from the DOT problem. While computing $\mathbf{c}_1^T\mathbf{A}^{-1}\mathbf{b}_1$, we use BiCG to solve $\mathbf{A}\mathbf{x} = \mathbf{b}_1$ and $\mathbf{A}^T\mathbf{y} = \mathbf{c}_1$. We start with random initial guesses $\mathbf{x}_0$ and $\mathbf{y}_0$. Table 3.1 shows that the initial residual norms are of similar order (both in the interval $(10^{2.5}, 10^{3.6})$). The tolerance parameter is $\text{tol} = 10^{-12}$. The BiCG algorithm stops when

$$\|\mathbf{r}_j\|\|\mathbf{s}_j\| < \text{tol} \cdot \|\mathbf{b}\|\|\mathbf{c}\|. \tag{3.6}$$

The convergence pattern (Figure 3.1) suggests that both residual norms get smaller with approximately the same rate.

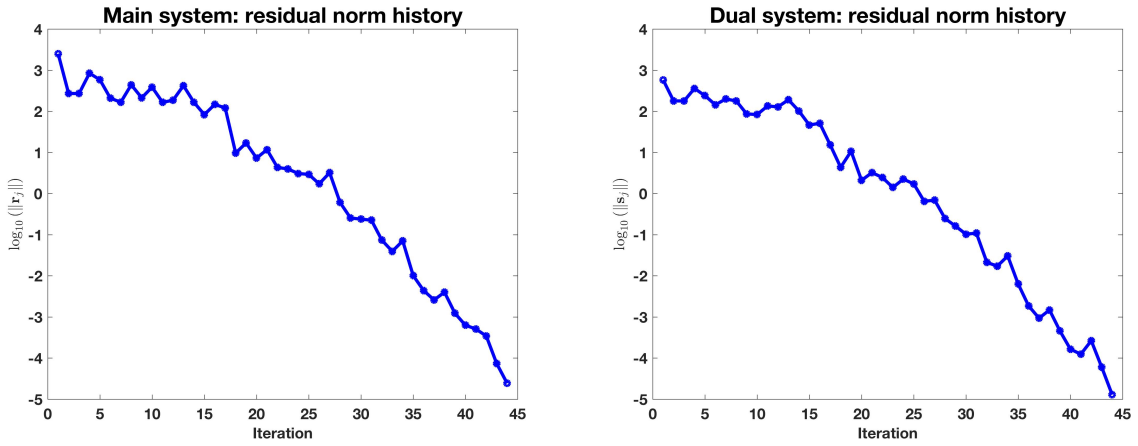


Figure 3.1: Convergence of BiCG used to solve main and dual systems simultaneously for a DOT problem with tolerance 10^{-12} (convergence criteria (3.6)). Matrix size is 40401×40401.

Next, we use BiCG to solve the systems $\mathbf{Ax} = \mathbf{b_1}$ and $\mathbf{A}^T\mathbf{y} = \mathbf{c}_2$ (the main system stays the same and the dual system changes). The algorithm stops when $\|r_j\|\|s_j\| < 10^{-12}\|\mathbf{c}_2\|\|\mathbf{b}_1\|$. Figure 3.2 shows the convergence curves. In this case, we already have a good initial guess for the main system (because we solved it once); the initial residuals are of different orders (10^{-4} and 10^{0}). The convergence criteria is (3.6) and the tolerance is tol = 10^{-12}. At the last iteration, the residual for the main system is smaller than 10^{-7} while the residual for the dual system is barely smaller than 10^{-1}. As mentioned in Table 2, we often need a solution with low residual norm for both, the main and the dual system. In particular, this is required for the DOT and Error Estimation and Grid Adaptation problems. Figure 3.2 shows that we can obtain solutions differing by six orders of magnitude with the convergence criteria (3.6). This motivates new convergence criteria. We stop BiCG when

$$\|\mathbf{r}_j\| < \text{tol} \cdot \|\mathbf{b}\| \quad \text{and} \quad \|\mathbf{s}_j\| < \text{tol} \cdot \|\mathbf{c}\|. \tag{3.7}$$

BiCGStab was proposed in [94]. In BiCG, we have $\mathbf{r}_j = P_j^{\text{BiCG}}(\mathbf{A})\mathbf{r}_0$, where $P_j^{\text{BiCG}}(\cdot)$ is a polynomial of degree j associated with BiCG. Since the dual residual is computed through the same recurrence relations, we also have $\mathbf{s}_j = P_j^{\text{BiCG}}(\mathbf{A}^T)\mathbf{s}_0$.

In BiCG we require that $\mathbf{r}_j \perp P_k^{\text{BiCG}}(\mathbf{A}^T)\mathbf{s}_0$, for all $k < j$ (because $\mathbf{r}_j \perp \mathcal{K}^{j-1}(\mathbf{A}^T, \mathbf{s}_0)$). For $k < j$ we obtain $0 = \langle P_j^{\text{BiCG}}(\mathbf{A})\mathbf{r}_0, P_k^{\text{BiCG}}(\mathbf{A}^T)\mathbf{s}_0\rangle = \langle P_j^{\text{BiCG}}(\mathbf{A})P_k^{\text{BiCG}}(\mathbf{A})\mathbf{r}_0, \mathbf{s}_0\rangle$. This new form of inner product does not involve multiplication by $\mathbf{A}^T$, and we do not need to compute $\mathbf{s}_j$ explicitly. Sonneveld proposed Conjugate Gradient Squared (CGS) algorithm [85], in which we construct a residual $\mathbf{r}_j^{\text{CGS}}$ such that $\mathbf{r}_j^{\text{CGS}} = \left(P_j^{\text{BiCG}}(\mathbf{A})\right)^2\mathbf{r}_0$. Since there exists a recursive formula for $P_j^{\text{BiCG}}(\mathbf{A})$ (see [80, 95, 95]), the CGS algorithm parameters are derived by squaring $P_j^{\text{BiCG}}(\mathbf{A})$ and using algebra [80].

In BiCGStab, we construct residuals such that $\mathbf{r}_j^{\text{BiCGSTab}} = \tilde{P}_j(\mathbf{A})P_j^{\text{BiCG}}(\mathbf{A})\mathbf{r}_0$, where the

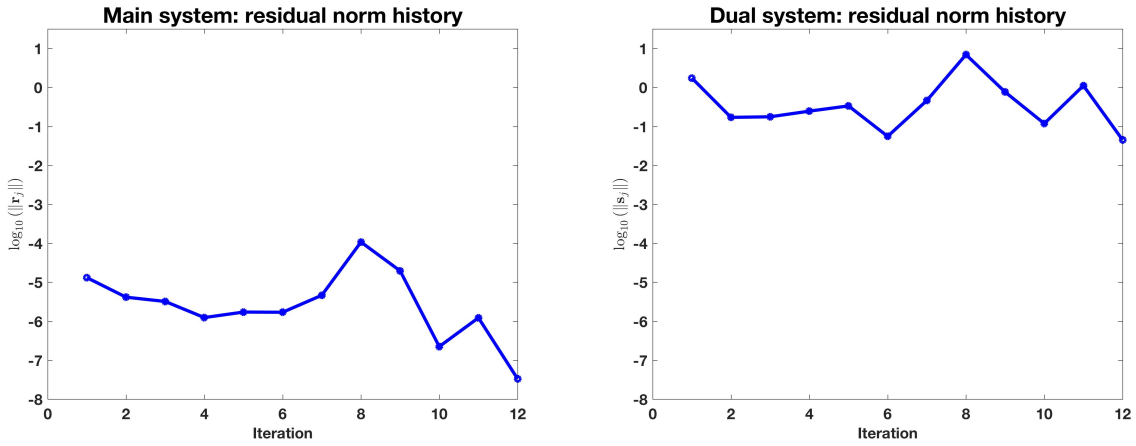


Figure 3.2: Convergence of BiCG used to solve a main and dual systems simultaneously for a DOT problem with tolerance 10^{-12} (convergence criteria (3.6)). Matrix size is 40401×40401.

degree j polynomial $\tilde{P}_j(z)$ is given by

$$\tilde{P}_{j+1}(z) = (1 - \omega_j z)\, \tilde{P}_j(z), \tag{3.8}$$

with the scalar ω_j that needs to be determined; we choose ω_j for which the quantity

$$\|\mathbf{r}_j\| = \|(\mathbf{I} - \omega_j \mathbf{A})\tilde{P}_{j-1}(\mathbf{A})P_j(\mathbf{A})\mathbf{r}_0\|$$

is minimized. The derivation of the algorithm parameters relies on the recursive formula for $P_j^{\mathrm{BiCG}}(\mathbf{A})$ and (3.8). The resulting algorithm is listed as Algorithm 4. BiCGStab has less oscillatory convergence behavior then CGS [43] and does not require multiplication by $\mathbf{A}^T$; however, it does require two matrix products with $\mathbf{A}$ per iteration (per j increment in Algorithm 4). BiCGStab breaks down in two cases: when $\mathbf{r}_j^T \mathbf{s}_j = 0$ and when $\omega_j = 0$.

BiCGStab has short recurrences. To update the residual $\mathbf{r}_j$ and the approximate solution $\mathbf{x}_j$, we only need $\mathbf{r}_{j-1}$, $\mathbf{x}_{j-1}$, a few vectors constructed in iteration $j-1$, and a few scalars computed in iterations $j-1$ and $j-2$. This makes the method computationally cheap (the iterations of the method are cheap). We use BiCGStab for the CFD problems in Table 5. We also use it in Table 6, where we test its performance on the GPUs for the LDC problem.

3.3 Preconditioners

The convergence of iterative solver can often be improved by using a *preconditioner*. Instead of solving (1.1), we solve

$$\mathbf{M}_1 \mathbf{A} \mathbf{M}_2 \mathbf{g} = \mathbf{M}_1 \mathbf{b}, \quad \mathbf{x} = \mathbf{M}_2 \mathbf{g}. \tag{3.9}$$

The matrix $\mathbf{M}_1$ is called a *left preconditioner* and the matrix $\mathbf{M}_2$ is called a *right preconitioner*. We can use a right preconditioner, a left preconditioner, or both. We would like preconditioners that are (1) cheap to compute, (2) cheap to apply (the preconditioner-vector product is fast), and (3) yield fast convergence (in terms of the number of iterations needed for convergence).

For non-symmetric matrices, incomplete LU (ILU) decomposition [62, 79] is often an effective preconditioner, and for symmetric matrices, a standard choice is the incomplete Cholesky decomposition [61]. Incomplete LU decomposition produces a lower triangular matrix $\mathbf{L}$ and an upper triangular matrix $\mathbf{U}$ such that $\mathbf{A} \approx \mathbf{LU}$. We typically do not want $\mathbf{L}$ nor $\mathbf{U}$ to be dense matrices, and there are multiple strategies for enforcing sparsity on $\mathbf{L}$ and $\mathbf{U}$, for example, ILU(0), ILU(k), ILUT, [79, 80]. We use $\mathbf{M}_1 = \mathbf{L}^{-1}$ and $\mathbf{M}_2 = \mathbf{U}^{-1}$ as a split preconditioner.

In Table 6, we analyze performance of various preconditioners implemented in parallel. It turns out that preconditioners from the ILU family can be expensive (characterized by long runtimes) if used in parallel. Since we do not form $\mathbf{L}^{-1}$ nor $\mathbf{U}^{-1}$ directly, multiplying by the preconditioner requires two triangular solves, and these solves are sequential.

The alternatives include, for example, block ILU(T) [84, 13, 52], which is a block version of ILU(T). In block ILU(T), we divide the matrix $\mathbf{A}$ into submatrices (blocks) along the main diagonal and ignore the selected non-diagonal components. Next, we perform the ILU(T) decomposition for each block. Since we ignored the non-diagonal parts of the matrix $\mathbf{A}$, block-wise triangular solves can be performed in parallel. Block ILUT is well suited for MPI-style coarse grain parallelism, but not well-suited for the GPUs. There are many other preconditioners, such as Jacobi, Sparse Approximate Inverse (SAI) [16, 18, 33], and Approximate Inverse (AINV) [23, 52]; for these preconditioners parallel setup and multiplication is more effective than for ILU. For instance, the computation of SAI can be easily done in parallel and the preconditioner multiplication does not involve triangular solves. However, for many linear systems SAI is not as effective in terms of reducing the number of iterations as ILU. For a detailed discussion on preconditioning techniques, we refer the reader to [17]. In Table 6, we extensively test various preconditioners (ILUT, block ILUT, SAI, Jacobi) coupled with either BiCGStab or GMRES(m) to assess their performance on the GPUs.

3.4 Recycling

We encounter long sequences of slowly changing linear systems in all the applications explored in this thesis. In most cases, the matrix $\mathbf{A}$ changes modestly from one system to another. This suggests that we can reduce the cost of solving these systems by recycling some part of the Krylov subspace between consecutive linear solves. Recycling subspaces helps with obtaining fast convergence right from the start for subsequent systems [69].

Recycling Krylov subspaces was originally developed for application between cycles for a single linear system (GMRES(m)) to avoid recomputing nearly the same, recurring subspace after restart, [27, 65, 26]. It was extended to be used with multiple linear systems in [69].

There are two important issues associated with recycling: (1) how do we select a space to recycle and (2) how is the selected space used in the Krylov subspace method. The answer varies with the method. Next, we briefly discuss recycling Krylov subspace solvers used in the latter parts of this thesis.

3.4.1 Recycling GMRES: rGCROT/rGCRO-DR

GCROT [27] is a truncated minimal residual method. The method preserves such a subspace between restarts of GMRES(m) that the loss of orthogonality with respect to the truncated (discarded) subspace is minimized. After a cycle of GMRES, we have computed a Krylov subspace. We would like to keep a part of this subspace and use it in the next cycle to maintain fast convergence. rGCROT [69] is an extension of GCROT that allows us to retain a Krylov subspace between linear solves, not just between GMRES(m) cycles.

GCRO-DR differs from GCROT in the way it selects the recycle space. GCROT measures the principal angles between successive spaces and selects the recurring one, indicated by small angles between successive Krylov subspaces, see [69, 27], while GCRO-DR uses harmonic Ritz vectors to approximate an invariant subspace.

Let $\mathbf{U} \in \mathbb{C}^{n\times k}$; we want to use range$(\mathbf{U})$ as a recycle space. We assume that we obtained $\mathbf{U}$ either from a previous GMRES(m) cycle, or from a previous solver run. Now we start a new cycle of GMRES(m).

Based on (3.1), the residual in iteration j of GMRES is minimized if $\mathbf{r}_j \perp \mathbf{A}\mathcal{K}^j(\mathbf{A}, \mathbf{r}_0)$. In addition, we want to keep the residuals orthogonal to $\mathbf{AU}$. First, we compute $\tilde{\mathbf{C}} = \mathbf{AU}$ and then we use QR decomposition of $\tilde{\mathbf{C}}$, $\tilde{\mathbf{C}} = \mathbf{CR}$ and $\mathbf{C}^T\mathbf{C} = \mathbf{I}$.

Let $\mathbf{x}_{-1}$ be arbitrary initial guess and let $\mathbf{r}_{-1} = \mathbf{b} - \mathbf{A}\mathbf{x}_{-1}$ be the associated residual. We start GMRES(m) with $\mathbf{r}_0 = (\mathbf{I} - \mathbf{C}\mathbf{C}^\mathbf{T})\,\mathbf{r}_{-1}$. This gives $\mathbf{r}_0 \perp \mathbf{C}$.

In GCRO [26] we build an augmented Arnoldi recurrence

$$\left(\mathbf{I} - \mathbf{C}\mathbf{C}^\mathbf{T}\right)\mathbf{A}\mathbf{W}_j = \mathbf{W}_{j+1}\mathbf{H}_{j+1,j}, \tag{3.10}$$

where $\mathbf{w}_1 = \frac{\mathbf{r}_0}{\|\mathbf{r}_0\|}$. Updates to $\mathbf{x}_j$ and $\mathbf{r}_j$ are based on (3.10). The resulting algorithm is listed as Algorithm 5

3.4.2 Recycling BiCG and BiCGStab

We use BiCG to solve the main and the dual system simultaneously, and thus, we compute two Krylov subspaces. Since we want to use recycling to help with the convergence, in rBiCG (recycled BiCG) [2, 3, 5] we use two separate recycle spaces, one for the main and one for the dual system. Let the column spaces of $\mathbf{U}$ and $\tilde{\mathbf{U}}$ be the recycle spaces we want to use for the

main and dual system, respectively. We compute $\mathbf{C} = \mathbf{A}\mathbf{U}$ and $\tilde{\mathbf{C}} = \mathbf{A}^T\tilde{\mathbf{U}}$. Neither $\mathbf{C}$ nor $\tilde{\mathbf{C}}$ will, in general, be an orthogonal matrix.

By construction (see [3, 5]), $\mathbf{C}_j^T\tilde{\mathbf{C}}_j$ is a diagonal matrix and we build two-sided Lanczos relation such that

$$\begin{aligned}\left(\mathbf{I} - \mathbf{C}\hat{\mathbf{C}}^T\right)\mathbf{A}\mathbf{V}_j &= \mathbf{V}_{j+1}\mathbf{T}_{j+1,j},\\ \left(\mathbf{I} - \tilde{\mathbf{C}}\check{\mathbf{C}}^T\right)\mathbf{A}^T\mathbf{W}_j &= \mathbf{W}_{j+1}\mathbf{S}_{j+1,j},\end{aligned} \tag{3.11}$$

where $\hat{\mathbf{C}} = \left[\frac{\tilde{\mathbf{c}}_1}{\tilde{\mathbf{c}}_1^T\mathbf{c}_1}, \frac{\tilde{\mathbf{c}}_2}{\tilde{\mathbf{c}}_2^T\mathbf{c}_2}, \ldots, \frac{\tilde{\mathbf{c}}_k}{\tilde{\mathbf{c}}_k^T\mathbf{c}_k}\right] = \tilde{\mathbf{C}}\mathbf{D}_C^{-1}$ and $\check{\mathbf{C}} = \left[\frac{\mathbf{c}_1}{\tilde{\mathbf{c}}_1^T\mathbf{c}_1}, \frac{\mathbf{c}_2}{\tilde{\mathbf{c}}_2^T\mathbf{c}_2}, \ldots, \frac{\mathbf{c}_k}{\tilde{\mathbf{c}}_k^T\mathbf{c}_k}\right] = \mathbf{C}\mathbf{D}_C^{-1}$. The matrix $\mathbf{D}_C$ is an invertible diagonal matrix and it is defined as $\mathbf{D}_C = \tilde{\mathbf{C}}^T\mathbf{C}$.

To compute (3.11), we use recurrences similar to these in the standard BiCG, see [2, 3, 5, 99]. Unlike in GMRES(m), in BiCG we do not save the Krylov subspace vectors; however, these vectors must be available when updating the recycle spaces. To limit memory usage, we choose a priori a maximum number of vectors to be kept in the memory, and once this maximum number is reached, we update the recycle spaces. In rBiCG, we select the recycle spaces using harmonic Ritz vectors. For details, see [2].

The derivation of the rBiCGStab [2, 4] is analogous to the derivation of BiCGStab. Let $\mathbf{r}_j$ and $\mathbf{s}_j$ be residuals at step j of rBiCG.

Let $\mathbf{B} = \left(\mathbf{I} - \mathbf{C}\hat{\mathbf{C}}^T\right)\mathbf{A}$ and $\tilde{\mathbf{B}} = \left(\mathbf{I} - \tilde{\mathbf{C}}\check{\mathbf{C}}^T\right)\mathbf{A}^T$. Then, there exists polynomials of degree j: $\Theta_j(\cdot)$ and $\tilde{\Theta}_j(\cdot)$ such that $\mathbf{r}_j^{\text{rBiCG}} = \Theta_j\left(\mathbf{B}\right)\mathbf{r}_0$, $\mathbf{s}_j^{\text{rBiCG}} = \tilde{\Theta}_j\left(\tilde{\mathbf{B}}\right)\mathbf{s}_0$. For proof, see [4, Theorem 4.1].

In rBiCG we have $\mathbf{s}_j \perp \mathbf{r}_k$ for $j < k$. This also implies [3] $\mathbf{s}_j \perp \mathcal{K}^k\left(\mathbf{B}, \mathbf{r}_0\right)$. Using the polynomial expressions, we get

$$0 = \left\langle \tilde{\Theta}_j\left(\tilde{\mathbf{B}}\right)\mathbf{s}_0, \Theta_k\left(\mathbf{B}\right)\mathbf{r}_0\right\rangle. \tag{3.12}$$

The polynomial $\tilde{\Theta}_j\left(\tilde{\mathbf{B}}\right)$ can be replaced by a different polynomial of degree j [85, 94]. Similarly as in the derivation of BiCGStab, we use

$$\bar{\Omega}_j\left(\tilde{\mathbf{B}}\right) = (1 - \bar{\omega}_j\tilde{\mathbf{B}})(1 - \bar{\omega}_{j-1}\tilde{\mathbf{B}})\ldots(1 - \bar{\omega}_1\tilde{\mathbf{B}}),$$

with $\bar{\omega}$s defined analogously as for rBiCGStab. After replacing $\tilde{\Theta}_j\left(\tilde{\mathbf{B}}\right)$ with $\bar{\Omega}_j\left(\tilde{\mathbf{B}}\right)$ in (3.12) we obtain

$$0 = \left\langle \bar{\Omega}_j\left(\tilde{\mathbf{B}}\right)\mathbf{s}_0, \Theta_k\left(\mathbf{B}\right)\mathbf{r}_0\right\rangle. \tag{3.13}$$

The inner product in (3.13) can be replaced by an alternative inner product (see [2, 4] for details; note $\tilde{\mathbf{B}}^T \neq \mathbf{B}$, also see [85, 94])

$$0 = \left\langle \mathbf{s}_0, \Omega_j\left(\mathbf{B}\right)\Theta_k\left(\mathbf{B}\right)\mathbf{r}_0\right\rangle, \tag{3.14}$$

with

$$\Omega_j(\mathbf{B}) = (1-\omega_j\mathbf{B})(1-\omega_{j-1}\mathbf{B})\dots(1-\omega_1\mathbf{B}).$$

The formula (3.14) does not require the transpose of $\mathbf{B}$. The rBiCGStab method is then derived based on (3.14). The resulting algorithm is presented as Algorithm 7.

rBiCGStab algorithm requires two recycle spaces, $\mathbf{U}$ and $\tilde{\mathbf{U}}$. In Table 5, we derive an alternative version, where we use only one recycle space. As a consequence, the rBiCGStab algorithm can be simplified and becomes computationally cheaper.

3.4.3 Recycling CG

The idea of deflated CG appeared for the first time in the paper by Nicolaides [67]. Saad et al. used deflated CG while solving a sequence of linear systems, where the matrix $\mathbf{A}$ does not change, but the right hand sides do [83]. Related work includes [86, 1]. In deflated CG, we keep the new vectors $\mathbf{A}$-orthogonal to a previously computed approximately invariant space. Parks et al. [70] further developed the idea in the context of Krylov subspace recycling. It was named *recycled CG (rCG)*.

Let $\mathbf{W}$ be an $n \times k$ matrix with orthonormal columns. We want to use range($\mathbf{W}$) as a recycle space, i.e., all the vectors $\mathbf{v}_j$ generated by rCG should be $\mathbf{A}$-orthogonal to $\mathbf{W}$.

We use $\mathbf{B} = \mathbf{A} - \mathbf{AW}(\mathbf{W}^T\mathbf{AW})^{-1}\mathbf{W}^T\mathbf{A}$ to obtain Lanczos recurrence

$$\mathbf{BV}_j = \mathbf{V}_j\mathbf{T}_{j+1,j}. \tag{3.15}$$

From (3.15) we get $\mathbf{W}^T\mathbf{B} = 0$. Because $\mathbf{A}$ is symmetric, so is $\mathbf{B}$. If $\mathbf{A}$ is an SPD matrix, $\mathbf{B}$ is an SSPD matrix.

Let $\mathbf{x}_{-1}$ be an arbitrary initial guess and let $\mathbf{r}_{-1}$ be the residual associated with $\mathbf{x}_{-1}$. We define $\mathbf{x}_0 = \mathbf{x}_{-1} + \mathbf{W}(\mathbf{W}^T\mathbf{AW})^{-1}\mathbf{W}^T\mathbf{r}_{-1}$, which gives $\mathbf{r}_0^T\mathbf{W} = 0$.

The derivation of the method is analogous to the derivation of CG and based on the Lanczos recurrence (3.15) and $\mathbf{r}_0$ defined above. The details of the derivation can be found in [83, 70]. The resulting algorithm is shown as Algorithm 8. We use recycled CG in Table 4, where we apply it to the topology optimization problem. In the same chapter, we show that the recycled CG algorithm can be extended to efficiently evaluate the quadratic form $\mathbf{b}^T\mathbf{A}^{-1}\mathbf{b}$.

3.5 Algorithm listings.

Algorithm 1 GMRES(m), adapted from [95]

1: $\mathbf{x}_0$ given initial guess;
2: $\mathbf{r} \leftarrow \mathbf{b} - \mathbf{A}\mathbf{x}_0$, $\mathbf{x} \leftarrow \mathbf{x}_0$;
3: Choose `maxit` and `tol`;
4: **for** $j = 1$ to maxit **do**
5: $\beta \leftarrow \|\mathbf{r}\|_2$, $\mathbf{v}_1 \leftarrow \frac{\mathbf{r}}{\beta}$, $\hat{\mathbf{b}} \leftarrow \beta \mathbf{e}_1$;
6: **for** $l = 1$ to m **do**
7: $\mathbf{w} \leftarrow \mathbf{A}\mathbf{v}_l$;
8: **for** $k = 1$ to l **do**
9: $h_{k,i} \leftarrow \mathbf{v}_k^T \mathbf{w}$;
10: $\mathbf{w} \leftarrow \mathbf{w} - h_{k,l}\mathbf{v}_k$;
11: **end for**
12: $h_{l+1,l} \leftarrow \|\mathbf{w}\|_2$, $\mathbf{v}_{l+1} \leftarrow \frac{\mathbf{w}}{h_{l+1,l}}$;
13: $r_{1,l} \leftarrow h_{1,l}$;
14: **for** $k = 2 \ldots$ to l **do**
15: $\gamma \leftarrow c_{k-1} r_{k-1,l} + s_{k-1} h_{k,l}$
16: $r_{k,l} \leftarrow -s_{k-1} r_{k-1,l} + c_{k-1} h_{k,l}$
17: $r_{k-1,l} \leftarrow \gamma$
18: **end for**
19: $\delta \leftarrow \sqrt{r_{l,l}^2 + h_{l+1,l}^2}$, $c_l = \frac{r_{l,i}}{\delta}$, $s_l \leftarrow \frac{h_{l+1,l}}{\delta}$;
20: $r_{l,l} \leftarrow c_l r_{l,l} + s_l h_{l+1,l}$;
21: $\hat{b_{l+1}} \leftarrow -s_l \hat{b_l}$, $\hat{b_l} \leftarrow c_l \hat{b_l}$;
22: $\rho \leftarrow |\hat{b_{l+1}}|$;
23: **if** ρ small enough **then**
24: $n_r \leftarrow l$ and go to line 28;
25: **end if**
26: **end for**
27: $n_r \leftarrow m, \mathbf{y}_{n_r} \leftarrow \frac{\hat{b_{n_r}}}{r_{n_r,n_r}}$;
28: **for** $k \leftarrow n_r - 1, \ldots$ to 1 **do**
29: $y_k \leftarrow \frac{\hat{b_k} - \sum_{l=k+1}^{n_r} r_{k,l} y_l}{r_{k,k}}$;
30: **end for**
31: $\mathbf{x} \leftarrow \mathbf{x} + \sum_{k=1}^{n_r} y_k \mathbf{v}_k$;
32: **if** $\rho < tol$ **then**
33: quit;
34: **end if**
35: **if** $j > maxit$ **then**
36: quit;
37: **end if**
38: $\mathbf{r} \leftarrow \mathbf{b} - \mathbf{A}\mathbf{x}$;
39: **end for**

Algorithm 2 CG, adapted from [95]

$\mathbf{x}_0$ given initial guess;
Compute $\mathbf{r} \leftarrow \mathbf{b} - \mathbf{A}\mathbf{x}_0$;
Choose `maxit` and `tol`;
for $k \leftarrow 1$ to maxit **do**
  $\rho_{k-1} \leftarrow \mathbf{r}_{k-1}^T \mathbf{r}_{k-1}$;
  if $k == 1$ **then**
    $\mathbf{p}_k \leftarrow \mathbf{r}_{k-1}$;
  else
    $\beta_{k-1} \leftarrow \frac{\rho_{k-1}}{\rho_{k-2}}$;
    $\mathbf{p}_k = \mathbf{r}_{k-1} + \beta_{k-1}\mathbf{p}_{k-1}$;
  end if
  $\mathbf{w}_k \leftarrow \mathbf{A}\mathbf{p}_k$;
  $\alpha_k \leftarrow \frac{\rho_{k-1}}{\mathbf{p}_k^T \mathbf{w}_k}$
  $\mathbf{x}_k \leftarrow \mathbf{x}_{k-1} + \alpha_k \mathbf{p}_k$;
  $\mathbf{r}_k \leftarrow \mathbf{r}_{k-1} - \alpha_k \mathbf{w}_k$;
  if $\|\mathbf{r}_k\| < tol$ **then**
    break
  end if
end for

Algorithm 3 BiCG, adapted from [90]

$\mathbf{x}_0$ given initial guess;
Compute $\mathbf{r} \leftarrow \mathbf{b} - \mathbf{A}\mathbf{x}_0$ and choose $\mathbf{s}_0$;
Set $\mathbf{p}_0 \leftarrow \mathbf{r}_0$ and $\mathbf{q}_0 \leftarrow \mathbf{s}_0$
Choose `maxit` and `tol`;
for $k \leftarrow 0$ to maxit **do**
  $\rho_k \leftarrow \mathbf{r}_k^T \mathbf{s}_k$;
  if $\rho_k == 0$ **then**
    FAILED; quit;
  end if
  $\alpha_k \leftarrow \frac{\rho_k}{\mathbf{q}_k^T \mathbf{A}\mathbf{p}_k}$;
  $\mathbf{x}_{k+1} \leftarrow \mathbf{x}_k + \alpha_k \mathbf{p}_k$;
  $\mathbf{y}_{k+1} \leftarrow \mathbf{y}_k + \alpha_k \mathbf{q}_k$;
  $\mathbf{r}_{k+1} \leftarrow \mathbf{r}_k - \alpha_k \mathbf{A}\mathbf{p}_k$;
  $\mathbf{s}_{k+1} \leftarrow \mathbf{s}_k - \alpha_k \mathbf{A}^\mathbf{T} \mathbf{q}_k$;
  $\eta_{k+1} \leftarrow \frac{\mathbf{s}_{k+1}^T \mathbf{r}_{k+1}}{\mathbf{s}_k^T \mathbf{r}_k}$;
  $\mathbf{p}_{k+1} \leftarrow \mathbf{r}_{k+1} + \eta_{k+1}\mathbf{p}_k$;
  $\mathbf{q}_{k+1} \leftarrow \mathbf{s}_{k+1} + \eta_{k+1}\mathbf{q}_k$;
  if $\|\mathbf{r}_{k+1}\| < tol$ AND $\|\mathbf{s}_{k+1}\| < tol$ **then**
    quit;
  end if
end for

Algorithm 4 BiCGStab, adapted from [95]

1: $\mathbf{x}_0$ given initial guess;
2: Compute $\mathbf{r}_0 \leftarrow \mathbf{b} - \mathbf{A}\mathbf{r}_0$ and choose $\mathbf{s}$, for example $\mathbf{s} \leftarrow \mathbf{r}_0$;
3: Choose `maxit` and `tol`;
4: **for** $j = 1$ to maxit **do**
5: $\rho_{j-1} \leftarrow \mathbf{s}^T \mathbf{r}_{j-1}$;
6: **if** $\rho_{j-1} == 0$ **then**
7: FAILED;
8: **end if**
9: **if** $j == 1$ **then**
10: $\mathbf{p}_j \leftarrow \mathbf{r}_{j-1}$;
11: **else**
12: $\beta_{j-1} \leftarrow \frac{\rho_{j-1}}{\rho_{j-2}} \cdot \frac{\alpha_{j-1}}{\omega_{j-1}}$;
13: $\mathbf{p}_j \leftarrow \mathbf{r}_{j-1} + \beta_{j-1}(\mathbf{p}_{j-1} - \omega_{j-1}\mathbf{v}_{j-1})$;
14: **end if**
15: $\mathbf{v}_j \leftarrow \mathbf{A}\mathbf{p}_j$;
16: $\alpha_j \leftarrow \frac{\rho_{j-1}}{\mathbf{s}^T \mathbf{v}_j}$;
17: $\mathbf{s} \leftarrow \mathbf{r}_{j-1} - \alpha_j \mathbf{v}_j$;
18: **if** $\|\mathbf{s}\| < tol$ **then**
19: $\mathbf{x}_j \leftarrow \mathbf{x}_{j-1} + \alpha_j \mathbf{p}$;
20: quit;
21: **end if**
22: $\mathbf{t} \leftarrow \mathbf{A}\mathbf{s}$;
23: $\omega_j \leftarrow \frac{\mathbf{t}^T \mathbf{s}}{\mathbf{t}^T \mathbf{t}}$;
24: $\mathbf{x}_j \leftarrow \mathbf{x}_{j-1} + \alpha_j \mathbf{p} + \omega_j \mathbf{s}$;
25: **if** $\mathbf{x}_j$ accurate enough **then**
26: quit;
27: **end if**
28: $\mathbf{r}_j \leftarrow \mathbf{s} - \omega_\mathbf{j}\mathbf{t}$;
29: **if** $\omega == 0$ **then**
30: error('for continuation ω must be different than 0');
31: **else**
32: go to line 5;
33: **end if**
34: **end for**

Algorithm 5 rGCROT(m,k)/rGCRO-DR(m,k), adapted from [7]

$\mathbf{x}_{-1}$ given initial guess;
Compute $\mathbf{r}_{-1} \leftarrow \mathbf{b} - \mathbf{A}\mathbf{r}_{-1}$
$\mathbf{U}$ given, compute $\tilde{\mathbf{C}} \leftarrow \mathbf{AU}$ and $\tilde{\mathbf{C}} \leftarrow \mathbf{CR}$ (thin QR decomposition);
$\xi \leftarrow \mathbf{C^T r}_{-1}$;
$\mathbf{r}_0 \leftarrow \mathbf{r}_{-1} - \mathbf{C}\xi$;
$\rho_0 \leftarrow \|\mathbf{r}_0\|$, $\mathbf{x}_0 = \mathbf{x}_{-1} + \mathbf{U}\left(\mathbf{R}^{-1}\xi\right)$;
Choose `maxit` and `tol`, set $k \leftarrow 0$;
while $\rho_k > tol * \|\mathbf{b}\|$ AND $k \leq maxit$ **do**
  $\mathbf{w}_1 \leftarrow \mathbf{r}_k/\rho_k$;
  for $j = 1 \dots m$ **do**
    $\mathbf{w}_{j+1} \leftarrow \mathbf{A}\mathbf{w}_j$;
    $k \leftarrow k + 1$;
    for $\ell = 1$ to k **do**
      $\mathbf{b}_{\ell,j} \leftarrow \mathbf{c}_\ell^T \mathbf{w}_{j+1}$;
      $\mathbf{w}_{j+1} \leftarrow \mathbf{w}_{j+1} - \mathbf{c}_\ell b_{\ell,j}$;
    end for
    for $\ell = 1$ to j **do**
      $h_{\ell,j} \leftarrow \mathbf{w}_\ell^T \mathbf{w}_{j+1}$;
      $\mathbf{w}_{j+1} \leftarrow \mathbf{w}_{j+1} - \mathbf{w}_\ell h_{\ell,j}$;
    end for
    $h_{j+1,j} \leftarrow \|\mathbf{w}_{j+1}\|_2$;
    $\mathbf{w}_{j+1} = h_{j+1,j}^{-1} \mathbf{w}_{j+1}$;
  end for
  Solve $\mathbf{y} \leftarrow \arg\min_{\tilde{\mathbf{y}} \in \mathbb{R}^m} \|\mathbf{e}_1 \rho_{k-m} - \underline{\mathrm{H}}_\mathbf{m} \tilde{\mathbf{y}}\|$;
  $\mathbf{z} \leftarrow -B\mathbf{y}$ where $B = (b_{\ell,j})$;
  $\mathbf{x}_k \leftarrow \mathbf{x}_{k-m} + \mathbf{W}_\mathbf{m}\mathbf{y}$;
  $\mathbf{x}_k \leftarrow \mathbf{x}_k - \mathbf{U}\left(\mathbf{R}^{-1}\mathbf{z}\right)$;
  $\mathbf{r}_k \leftarrow \mathbf{b} - \mathbf{A}\mathbf{x}_k$;
  $\rho_k \leftarrow \|\mathbf{r}_k\|$;
  Update U and C if desired;
end while

Algorithm 6 rBiCG, adapted from [5]

$\mathbf{x}_{-1}$ and $\mathbf{y}_{-1}$ given initial guesses;
$\mathbf{U}$ and $\tilde{\mathbf{U}}$ given, want to recycle range$(\mathbf{U})$ and range$\left(\tilde{\mathbf{U}}\right)$. if no spaces given, set to empty;
Compute $\mathbf{x}_0$ and $\mathbf{y}_0$, $\mathbf{r}_0$ and $\mathbf{s}_0$, $\hat{\mathbf{C}}$ and $\check{\mathbf{C}}$;
if $\langle \mathbf{r}_0, \mathbf{s}_0 \rangle = 0$ **then**
    $\mathbf{y}_0 \leftarrow$ random vector;
    Recompute $\mathbf{s}_0$;
end if
Set $\beta_0 \leftarrow 0$, $\mathbf{p}_0 \leftarrow 0$, $\tilde{\mathbf{p}} \leftarrow 0$;
Choose `maxit` and `tol`;
for $k \leftarrow 1$ to maxit **do**
    $\mathbf{p}_k \leftarrow \mathbf{r}_{k-1} + \beta_{k-1}\mathbf{p}_{k-1}$;
    $\tilde{\mathbf{p}}_k \leftarrow \mathbf{s}_{k-1} + \beta_{k-1}\tilde{\mathbf{p}}_{k-1}$;
    $\mathbf{q}_k \leftarrow \left(\mathbf{I} - \mathbf{U}\hat{\mathbf{C}}^T\mathbf{A}\right)\mathbf{p}_k$;
    $\tilde{\mathbf{q}}_k \leftarrow \left(\mathbf{I} - \tilde{\mathbf{U}}\check{\mathbf{C}}^T\mathbf{A}^T\right)\tilde{\mathbf{p}}_k$;
    $\alpha_k \leftarrow \frac{\langle \mathbf{s}_{k-1}, \mathbf{r}_{k-1} \rangle}{\langle \tilde{\mathbf{p}}_k, \mathbf{A}\mathbf{q}_k \rangle}$;
    $\mathbf{x}_k \leftarrow \mathbf{x}_{k-1} + \alpha_k \mathbf{q}_k$;
    $\mathbf{y}_k \leftarrow \mathbf{y}_{k-1} + \alpha_k \tilde{\mathbf{q}}_k$;
    $\mathbf{r}_k \leftarrow \mathbf{r}_{k-1} - \alpha_k A\mathbf{q}_k$;
    $\mathbf{s}_k \leftarrow \mathbf{s}_{k-1} - \alpha_k \mathbf{A}^T\tilde{\mathbf{q}}_k$;
    if $\|\mathbf{r}_k\|\|\mathbf{s}_k\| < tol$ **then**
        break
    end if
    $\beta_k \leftarrow \frac{\langle \mathbf{s}_k, \mathbf{r}_k \rangle}{\langle \mathbf{s}_{k-1}, \mathbf{r}_{k-1} \rangle}$;
end for

Algorithm 7 rBiCGStab, adapted from [4]

$\mathbf{U}$ and $\tilde{\mathbf{U}}$ given, want to recycle range$(\mathbf{U})$ and range$\left(\tilde{\mathbf{U}}\right)$. If no $\mathbf{C}$ and $\tilde{\mathbf{C}}$, compute $\mathbf{C} \leftarrow \mathbf{AU}$ and $\tilde{\mathbf{C}} \leftarrow \mathbf{A^T}\tilde{\mathbf{U}}$;
Compute $\mathbf{D}_C \leftarrow \tilde{\mathbf{C}}^{\mathbf{T}}\mathbf{C}$, $\hat{\mathbf{C}} \leftarrow \tilde{\mathbf{C}}\mathbf{D}_C^{-1}$, $\check{\mathbf{C}} \leftarrow \mathbf{C}\mathbf{D}_C^{-1}$;
$\mathbf{x}_{-1}$ given initial guess;
Compute $\mathbf{r}_{-1} \leftarrow \mathbf{b} - \mathbf{A}\mathbf{r}_{-1}$;
Set $\mathbf{s}_{-1}$ to a random vector;
$\mathbf{x}_0 \leftarrow \mathbf{x}_{-1} + \mathbf{U}\hat{\mathbf{C}}^{\mathbf{T}}\mathbf{r}_{-1}$, $\mathbf{r}_0 \leftarrow \left(\mathbf{I} - \mathbf{C}\hat{\mathbf{C}}^T\right)\mathbf{r}_{-1}$, $\mathbf{s}_0 \leftarrow \left(\mathbf{I} - \mathbf{C}\check{\mathbf{C}}^T\right)\mathbf{s}_{-1}$;
Set β_0 and ω_0 to 0. Set $\mathbf{p}_0$, $\mathbf{q}_0$ and $\mathbf{x}_c$ to zero vectors;
Choose `maxit` and `tol`;
for $k = 1, \ldots, maxit$ **do**
  $\mathbf{p}_k \leftarrow \mathbf{r}_{k-1} - \beta_{k-1}\omega_{k-1}\mathbf{q}_{k-1}$;
  $\mathbf{q_k} \leftarrow \mathbf{A}\mathbf{p}_k$;
  $\boldsymbol{\zeta}_k \leftarrow \hat{\mathbf{C}}^{\mathbf{T}}\mathbf{q}_k$;
  $\mathbf{q}_k \leftarrow \mathbf{q}_k - \mathbf{C}\boldsymbol{\zeta}_k$;
  $\alpha_k \leftarrow \frac{\langle \mathbf{s}_0, \mathbf{r}_{k-1} \rangle}{\langle \mathbf{s}_0, \mathbf{q}_k}\rangle$;
  $\mathbf{s}_k \leftarrow \mathbf{r}_{k-1} - \alpha_k\mathbf{q}_k$;
  $\mathbf{t}_k \leftarrow \mathbf{A}\mathbf{s}_k$;
  $\boldsymbol{\gamma}_k \leftarrow \hat{\mathbf{C}}^T\mathbf{t}_k$;
  $\mathbf{t}_k = \mathbf{t}_k - \mathbf{C}\boldsymbol{\gamma}_k$;
  $\omega_k \leftarrow \frac{\langle \mathbf{s}_k, \mathbf{t}_k \rangle}{\langle \mathbf{t}_k, \mathbf{t}_k \rangle}$;
  $\mathbf{x}_k \leftarrow \mathbf{x}_{k-1} + \alpha_k\mathbf{p}_k + \omega_k\mathbf{s}_k$;
  $\mathbf{x}_c \leftarrow \mathbf{x}_c + \alpha_k\boldsymbol{\zeta}_k + \omega_k\boldsymbol{\gamma}_k$;
  $\mathbf{r}_k \leftarrow \mathbf{r}_{k-1} - \alpha_k\mathbf{q}_k - \omega_k\mathbf{t}_k$;
  if $\|\mathbf{r}_k\| < tol$ **then**
    BREAK;
  end if
  $\beta_k \leftarrow \frac{\langle \mathbf{s}_0, \mathbf{r}_k \rangle}{\langle \mathbf{s}_0, \mathbf{r}_{k-1} \rangle} \cdot \frac{\alpha_k}{\omega_k}$;
end for
$\mathbf{x}_k \leftarrow \mathbf{x}_k - \mathbf{U}\mathbf{x}_c$;

Algorithm 8 rCG, adapted from [70]

$\mathbf{x}_{-1}$ given initial guess;
$\mathbf{U}$ given; range $(\mathbf{U})$ recycle space;
Compute $\mathbf{x}_0$, $\mathbf{r}_0$;
Set $\mathbf{p}_0 \leftarrow \mathbf{r}_0$;
Choose `maxit` and `tol`;
for $k \leftarrow 1$ to maxit **do**
  if $k > 1$ **then**
    $\alpha_{k-1} \leftarrow \frac{\langle \mathbf{r}_{k-1}, \mathbf{r}_{k-1} \rangle}{\langle \mathbf{p}_{k-1}, \mathbf{A}\mathbf{p}_{k-1} \rangle}$;
  else
    $\alpha_{k-1} \leftarrow 0$
  end if
  $\mathbf{x}_k \leftarrow \mathbf{x}_{k-1} + \alpha_{k-1}\mathbf{p}_{k-1}$;
  $\mathbf{r}_k \leftarrow \mathbf{r}_{k-1} - \alpha_{k-1}A\mathbf{p}_{k-1}$;
  $\beta_{k-1} \leftarrow \frac{\langle \mathbf{r}_k, \mathbf{r}_k \rangle}{\langle \mathbf{r}_{k-1}, \mathbf{r}_{k-1} \rangle}$;
  Solve for μ_k: $U^T AU\mu_k = (AU)^T \mathbf{r}_k$;
  $\mathbf{p}_k \leftarrow \beta_{k-1}\mathbf{p}_{k-1} + \mathbf{r}_k - U\mu_k$;
  if $\|\mathbf{r}_k\| < tol$ **then**
    break
  end if
end for

Chapter 4

Estimating Bilinear and Quadratic Forms

4.1 Motivation

Many important applications, such as Diffuse Optical Tomography (DOT), topology optimization, and functional-based error estimation discussed in Chapter 2, as well as quantum physics [72], CFD [37], and others, involve repeated evaluation of expensive bilinear and quadratic forms, for instance

$$\begin{aligned} \min_{\mathbf{p}} \|\mathbf{C}^T(\mathbf{A}(\mathbf{p}) + \sigma\mathbf{E})^{-1}\mathbf{B} - \mathbf{D}\|_F^2 \qquad & \textit{(DOT)} \\ \min_{\mathbf{p}} \operatorname{trace}\left(\mathbf{F}^T\mathbf{K}(\mathbf{p})^{-1}\mathbf{F}\right) \qquad & \textit{(topology opt.)} \\ \varepsilon_{\mathbf{J}_h} = -\mathbf{c}_k^T\mathbf{A}^{-1}\mathbf{b}, \quad k = 1, 2\ldots \qquad & \textit{(func.-based error est.)} \end{aligned}$$

These bilinear and quadratic forms may have to be evaluated many times. We would like to emphasize, which is important for the methods developed later in this chapter, that, if $\mathbf{c}^T\mathbf{A}^{-1}\mathbf{b}$, is a bilinear form, there are two linear systems associated with it (one for quadratic form). The system $\mathbf{Ax} = \mathbf{b}$ is called *the main problem* and the system $\mathbf{Ay} = \mathbf{c}$ is called the *dual system*. The bilinear/quadratic forms are usually computed by solving a large number of linear systems for either the main problem or the dual problem, using a direct solver or an iterative solver. For Newton-type optimization methods, a second set of systems needs to be solved in every time step because the Jacobian (for nonlinear least squares) or Hessian (more general) needs to be computed. In functional-based error estimation (Chapter 2), the dual problem represents the functional adjoint problem, and an accurate solution to this problem is needed.

The computational cost of the problems described above is enormous, and we need to develop highly efficient solution methods. The number of bilinear/quadratic forms might be substantially reduced through a randomized approach [100, 9]. However, even with the randomization, we still need to evaluate several such forms in every step of the optimization process. One possible method of evaluating the bilinear/quadratic form is to solve one of the systems (main or dual) and evaluate the bilinear form an using inner product. Instead, one might estimate the bilinear form directly (not through an inner product) using properties of a Krylov subspace solver. In Section 4.3 we expand the method developed in [90], which is based on properties of BiCG; using this method, we can obtain an estimate for bilinear/quadratic forms with much higher accuracy than using inner an product. The details are discussed in Section 4.3.

In addition, in optimization problems, we need an (approximate) solution to both linear systems (main and dual) in order to compute the expressions for derivatives. In the DOT problem, discussed in Chapter 2, these solutions are required for the Jacobian evaluation. In topology optimization (discussed in the same chapter) approximate solutions are needed to evaluate the gradient (sensitivities). In functional-based error estimation, the estimates for both solutions are needed. In this application, we need a solution with modest accuracy for the adjoint problem, and we need a highly accurate solution for the Error Transport Equation (ETE). In topology optimization we need a highly accurate estimate for the quadratic form, because it corresponds to the evaluation of compliance function, and a modestly accurate estimate for the solution of the main system (this solution is needed to approximate the gradient of compliance).

The BiCG-based approach (discussed in Section 4.3) to estimate bilinear forms, also computes approximate solutions to both problems. The accuracy of these solutions is lower than the accuracy of the bilinear form, and this is advantageous for our applications. Hence, we investigate this approach in Section 4.3, and we extend it to Krylov subspace recycling. In Section 4.4, we test this approach for the applications.

4.2 Background

The problem of computing $\mathbf{c}^T\mathbf{f}(\mathbf{A})\,\mathbf{b}$, where $\mathbf{f}(\mathbf{A})$ is a matrix function and $\mathbf{b}$ and $\mathbf{c}$ are vectors has been a topic of research for the last twenty-five years [39, 38]. In this thesis, we focus on the special case: $\mathbf{f}(\mathbf{A}) = \mathbf{A}^{-1}$.

Several authors focus on estimating quadratic forms $\mathbf{b}^T\mathbf{A}^{-1}\mathbf{b}$ using quadrature and the method of moments [42, 38, 12]. In these papers, the authors assume that $\mathbf{A}$ is SPD or HPD. This assumption ensures the existence of the spectral decomposition $\mathbf{A} = \mathbf{Q}\mathbf{\Lambda}\mathbf{Q}^T$, where $\mathbf{Q}$ is an orthogonal matrix and $\mathbf{\Lambda}$ is a diagonal matrix with eigenvalues on the diagonal (the eigenvalues are real and positive in this case); we assume $\lambda_1 \leq \lambda_2 \leq \ldots \leq \lambda_n$. The columns of $\mathbf{Q}$ are the

normalized eigenvectors. The decomposition leads to

$$\begin{aligned}\mathbf{c}^T\mathbf{f}\left(\mathbf{A}\right)\mathbf{b} &= \mathbf{c}^T\mathbf{f}\left(\mathbf{Q}\mathbf{\Lambda}\mathbf{Q}^T\right)\mathbf{b}\\ &= \underbrace{\mathbf{c}^T\mathbf{Q}}_{\boldsymbol{\alpha}^T}\,\mathbf{f}(\mathbf{\Lambda})\,\underbrace{\mathbf{Q}^T\mathbf{b}}_{\boldsymbol{\beta}}\\ &= \boldsymbol{\alpha}^T\mathbf{f}(\Lambda)\boldsymbol{\beta} = \sum_{i=1}^{n}\mathbf{f}(\lambda_i)\alpha_k\beta_k.\end{aligned}$$

We observe that f is a smooth function on $[\lambda_1, \lambda_n]$. Therefore, the summation is approximated by a Riemann-Stieltjes integral (see [40, Definition 2.1]), i.e. $\mathbf{c}^T\mathbf{f}\left(\mathbf{A}\right)\mathbf{b} = \int_a^b \mathbf{f}(\lambda)\, d\alpha(\lambda)$, with the non-decreasing measure function defined as [38, 40]

$$\alpha(\lambda) = \begin{cases} 0 & \text{if } \lambda < a = \lambda_1,\\ \sum_{j=1}^{k}\alpha_j\beta_j & \text{if } \lambda_k \le \lambda < \lambda_{k+1},\\ \sum_{j=1}^{n}\alpha_j\beta_j & \text{if } \lambda \ge \lambda_n = b.\end{cases}$$

In [39], Golub and Meurant use Gauss, Gauss-Lobatto or Gauss-Radau quadrature formulas to derive lower and upper bounds for the Riemann-Stieltjes integral estimate of $\mathbf{c}^T\mathbf{f}\left(\mathbf{A}\right)\mathbf{b}$, where $\mathbf{f}$ is an arbitrary, smooth function on some interval of the real line. These bounds are further developed using moments by Golub in [38]. In [42], the authors show how to use the integral estimates to evaluate a quadratic form $\min_{\mathbf{x}}\left(\mathbf{x}^T\mathbf{A}\mathbf{x} - 2\mathbf{b}^T\mathbf{x}\right)$ subject to $\|\mathbf{x}\| = \sigma$, where σ is a constant; in this quadratic form, $\mathbf{A}$ is a real, symmetric, positive define matrix. The authors also use the approach based on the Riemann-Stieltes integral and its quadrature approximation to estimate the error, defined as the difference between the initial guess to the solution, $\mathbf{x}_0$, and true solution to the linear system $\mathbf{A}\mathbf{x} = \mathbf{b}$, i.e., $\|\mathbf{x}_0 - \tilde{\mathbf{x}}\|$. In addition, the authors introduce a method for estimating the diagonal elements of the inverse, i.e., $\left(\mathbf{A}^{-1}\right)_{ii}$. The bounds for these estimates are discussed as well.

The assumption of $\mathbf{A}$ being an SPD matrix is lifted in [12]. In this paperm the authors evaluate the elements of the inverse $\left(\mathbf{A}^{-1}\right)_{ij}$, $\mathrm{Tr}\left(\mathbf{A}^{-1}\right)$, and $\det\left(\mathbf{A}\right)$ for non-symmetric matrices using the integral/quadrature methods derived for symmetric matrices. The method of moments is tied with conjugate gradient error bounds in [89]. In [88], Strakoš generalizes model reduction based on matching moments to the non-symmetric matrices, and shows how to use matching moments and two-sided Lanczos recurrence to estimate the bilinear form.

In the paper from 2008 [41], Golub, Stoll and Wathen consider estimating the scattering amplitude $\mathbf{c}^T\mathbf{A}^{-1}\mathbf{b}$, and show how to evaluate the bilinear form using QMR and generalized LSQR. The latter method is based on the properties of the Lanczos algorithm. This approximation relies on the assumption that the vectors in the Krylov subspace obtained from the Lanczos algorithm are orthogonal to each other. While this is true in exact arithmetic, due to round-off errors this is not likely to be the case in finite precision arithmetic, especially if many iterations are needed for convergence.

In addition, the authors of [41] introduce a method of estimating the scattering amplitude $\mathbf{c}^T\mathbf{A}^{-1}\mathbf{b}$, based on BiCG and that relies only on the orthogonality between vectors in two consecutive iterations of the BiCG method. In the paper by Strakoš and Tichý, [90] (unpublished version of the paper is referenced in [41]) this BiCG-based approximation is restated in the context of Vorobyev moments. The paper contains an overview of known approaches for the bilinear form estimation. The authors also introduce some error estimates for the new, BiCG-based approach. An advantage of the BiCG-based method is that the error in the bilinear form converges quadratically with the product of residual norms.

In this chapter, we extend the results of Strakoš and Tichý. First, we introduce their method and then, we adapt the method to be used with Krylov subspace recycling (recycled BiCG [3, 5], Algorithm 6). Next, we derive an analogous method for symmetric positive definite matrices using CG, and extend it to recycling CG [70] (Algorithm 8). In the last section, we show results for the topology optimization, DOT, and functional-based error estimation.

4.3 Analysis

4.3.1 Biconjugate gradient and bilinear form approximation

Consider BiCG algorithm (Algorithm 3). Let the vectors $\mathbf{x}_j$ and $\mathbf{y}_j$ be the approximate solutions to the main and to the dual system (to $\mathbf{Ax} = \mathbf{b}$ and $\mathbf{A}^T\mathbf{y} = \mathbf{c}$) computed in the step j of the BiCG iteration. Furthermore, let $\mathbf{r}_j = \mathbf{b} - \mathbf{A}\mathbf{x}_j$ and $\mathbf{s}_j = \mathbf{c} - \mathbf{A}^T\mathbf{y}_j$ denote the main and the dual residuals, respectively.

Lemma 4.1. *For $\mathbf{x}_j$, $\mathbf{y}_j$, $\mathbf{r}_j$ and $\mathbf{s}_j$ defined above, we have*

$$\mathbf{s}_j\mathbf{A}^{-1}\mathbf{r}_j - \mathbf{s}_{j+1}\mathbf{A}^{-1}\mathbf{r}_{j+1} = \alpha_j\mathbf{s}_j^T\mathbf{r}_j,$$

where α_j is the quantity computed in Line 10 of Algorithm 3.

Proof. [90] ∎

Next, we observe that $\mathbf{c}^T\mathbf{A}^{-1}\mathbf{b}$ can be written as

$$\mathbf{c}^T\mathbf{A}^{-1}\mathbf{b} = \left(\mathbf{s}_0 + \mathbf{A}^T\mathbf{y}_0\right)^T\mathbf{A}^{-1}\left(\mathbf{r}_0 + \mathbf{A}\mathbf{x}_0\right) = \mathbf{s}_0^T\mathbf{A}^{-1}\mathbf{r}_0 + \mathbf{s}_0^T\mathbf{x}_0 + \mathbf{y}_0^T\mathbf{b}. \tag{4.1}$$

Thus for any $N > 0$, we have, using Lemma 4.1,

$$\begin{aligned}\mathbf{c}^T\mathbf{A}^{-1}\mathbf{b} &= \mathbf{s}_0^T\mathbf{A}^{-1}\mathbf{r}_0 + \mathbf{s}_0^T\mathbf{r}_0 + \mathbf{s}_0^T\mathbf{x}_0 + \mathbf{y}_0^T\mathbf{b}\\ &= \left(\mathbf{s}_0^T\mathbf{x}_0 + \mathbf{y}_0^T\mathbf{b}\right) + \left(\mathbf{s}_0^T\mathbf{A}^{-1}\mathbf{r}_0 - \mathbf{s}_1^T\mathbf{A}^{-1}\mathbf{r}_1\right)\\ &+ \left(\mathbf{s}_1^T\mathbf{A}^{-1}\mathbf{r}_1 - \mathbf{s}_2^T\mathbf{A}^{-1}\mathbf{r}_2\right) + \dots\\ &+ \left(\mathbf{s}_{N-1}^T\mathbf{A}^{-1}\mathbf{r}_{N-1} - \mathbf{s}_N^T\mathbf{A}^{-1}\mathbf{r}_N\right)\\ &+ \mathbf{s}_N^T\mathbf{A}^{-1}\mathbf{r}_N\\ &= \underbrace{\left(\mathbf{s}_0^T\mathbf{x}_0 + \mathbf{y}_0^T\mathbf{b}\right)}_{\Phi_0} + \sum_{j=0}^{N-1}\alpha_j\mathbf{s}_j^T\mathbf{r}_j + \mathbf{s}_N^T\mathbf{A}^{-1}\mathbf{r}_N. \end{aligned} \tag{4.2}$$

The derivation above leads to the expression

$$\mathbf{c}^T\mathbf{A}^{-1}\mathbf{b} - \mathbf{s}_N^T\mathbf{A}^{-1}\mathbf{r}_N = \Phi_0 + \sum_{j=0}^{N-1}\alpha_j\mathbf{s}_j^T\mathbf{r}_j. \tag{4.3}$$

If the method converges, we have $\left|\mathbf{s}_N\mathbf{A}^{-1}\mathbf{r}_N\right| < \varepsilon$ for some N. This gives the inequality at iteration N of the BiCG method [41, 90]

$$\left|\mathbf{c}^T\mathbf{A}^{-1}\mathbf{b} - \left(\Phi_0 + \sum_{j=0}^{N-1}\alpha_j\mathbf{s}_j^T\mathbf{r}_j\right)\right| < \varepsilon, \tag{4.4}$$

where Φ_0 denotes the initial approximation. Note that the error in the approximation equals $\mathbf{s}_N^T\mathbf{A}^{-1}\mathbf{r}_N = \mathbf{s}_N^T\mathbf{e}_N$, where $\mathbf{e}_N = \tilde{\mathbf{x}} - \mathbf{x}_N$ with $\tilde{\mathbf{x}}$ being the exact solution to the main system $\mathbf{A}\mathbf{x} = \mathbf{b}$.

4.3.2 Extension of the method to recycled BiCG

In recycled BiCG (Algorithm 6), we start with arbitrary initial guesses $\mathbf{x}_{-1}$ and $\mathbf{y}_{-1}$, and construct $\mathbf{x}_0$ and $\mathbf{y}_0$ through projection to ensure $\mathbf{r}_0 \perp \tilde{\mathbf{C}}$ and $\mathbf{s}_0 \perp \mathbf{C}$, where range$(\mathbf{U})$ and range$\left(\tilde{\mathbf{U}}\right)$ are the recycle spaces and $\mathbf{C} = \mathbf{A}\mathbf{U}$, $\tilde{\mathbf{C}} = \mathbf{A}^T\tilde{\mathbf{U}}$, and $\mathbf{C}^T\tilde{\mathbf{C}} = \mathbf{D}_\mathbf{C}$. $\mathbf{D}_\mathbf{C}$ is a diagonal matrix with real positive coefficients and $\hat{\mathbf{C}} = \tilde{\mathbf{C}}\mathbf{D}_\mathbf{C}^{-1}$, $\check{\mathbf{C}} = \mathbf{C}\mathbf{D}_\mathbf{C}^{-1}$.

We define the initial residuals

$$\begin{aligned}\mathbf{r}_{-1} &= \mathbf{b} - \mathbf{A}\mathbf{x}_{-1},\\ \mathbf{s}_{-1} &= \mathbf{c} - \mathbf{A}^T\mathbf{y}_{-1},\end{aligned} \tag{4.5}$$

and define $\mathbf{x}_0$ and $\mathbf{y}_0$ as

$$\begin{aligned}\mathbf{x}_0 &= \mathbf{x}_{-1} + \mathbf{U}\hat{\mathbf{C}}^T\mathbf{r}_{-1},\\ \mathbf{y}_0 &= \mathbf{y}_{-1} + \tilde{\mathbf{U}}\check{\mathbf{C}}^T\mathbf{s}_{-1},\end{aligned} \tag{4.6}$$

which leads to

$$\begin{aligned}\mathbf{r}_0 &= (\mathbf{I} - \mathbf{C}\hat{\mathbf{C}}^T)\mathbf{r}_{-1},\\ \mathbf{s}_0 &= (\mathbf{I} - \tilde{\mathbf{C}}\check{\mathbf{C}}^T)\mathbf{s}_{-1}.\end{aligned}$$

Lemma 4.2. *If* $\mathbf{x}_0$ *and* $\mathbf{y}_0$ *are defined as in (4.6), then* $\mathbf{r}_0 \perp \tilde{\mathbf{C}}$ *and* $\mathbf{s}_0 \perp \mathbf{C}$.

Proof. (See [2]).

$$\begin{aligned}\mathbf{r}_0^T\tilde{\mathbf{C}} &= \left((\mathbf{I} - \mathbf{C}\hat{\mathbf{C}}^T)\mathbf{r}_{-1}\right)^T\tilde{\mathbf{C}} = \mathbf{r}_{-1}^T\left(\tilde{\mathbf{C}} - \hat{\mathbf{C}}\mathbf{D}_{\mathbf{C}}\right) = 0.\\ \mathbf{s}_0^T\mathbf{C} &= \left((\mathbf{I} - \tilde{\mathbf{C}}\check{\mathbf{C}}^T)\mathbf{s}_{-1}\right)^T\mathbf{C} = \mathbf{s}_{-1}^T\left(\mathbf{C} - \check{\mathbf{C}}\mathbf{D}_{\mathbf{C}}\right) = 0.\end{aligned}$$

■

Next, we follow the derivations in (4.1) and (4.2) to obtain the initial approximation to the bilinear form using rBiCG, and to obtain a method for approximating the bilinear form using properties of rBiCG.

In our derivation we use $\mathbf{p}_k$, $\tilde{\mathbf{p}}_k$, $\mathbf{q}_k$, $\tilde{\mathbf{q}}_k$ defined by the recurrences in Algorithm 6 (lines 11 – 14)

$$\begin{aligned}\mathbf{p}_j &= \mathbf{r}_{j-1} + \beta_{j-1}\mathbf{p}_{j-1},\\ \tilde{\mathbf{p}}_j &= \mathbf{s}_{j-1} + \beta_{j-1}\tilde{\mathbf{p}}_{j-1},\\ \mathbf{q}_j &= \left(\mathbf{I} - \mathbf{U}\hat{\mathbf{C}}^T\mathbf{A}\right)\mathbf{p}_j,\\ \tilde{\mathbf{q}}_j &= \left(\mathbf{I} - \mathbf{U}\check{\mathbf{C}}^T\mathbf{A}^T\right)\tilde{\mathbf{p}}_j,\end{aligned} \tag{4.7}$$

and

$$\beta_j = \frac{\langle \mathbf{s}_j, \mathbf{r}_j\rangle}{\langle \mathbf{s}_{j-1}, \mathbf{r}_{j-1}\rangle}.$$

Using (4.6) we have

$$\begin{aligned}\mathbf{b} &= \mathbf{r}_0 + \mathbf{A}\mathbf{x}_{-1} + \mathbf{C}\hat{\mathbf{C}}^T\mathbf{r}_{-1},\\ \mathbf{c} &= \mathbf{s}_0 + \mathbf{A}^T\mathbf{y}_{-1} + \tilde{\mathbf{C}}\check{\mathbf{C}}^T\mathbf{s}_{-1}.\end{aligned}$$

Theorem 4.3. *Using the recycled BiCG algorithm (Algorithm 6), we have*

(1) $\mathbf{c}^T\mathbf{A}^{-1}\mathbf{b} = \mathbf{s}_0^T\mathbf{A}^{-1}\mathbf{r}_0 + \mathbf{s}_0^T\mathbf{x}_0 + \mathbf{y}_0^T\mathbf{b}$,

(2) $\mathbf{s}_j\mathbf{A}^{-1}\mathbf{r}_j - \mathbf{s}_{j+1}\mathbf{A}^{-1}\mathbf{r}_{j+1} = \alpha_{j+1}\mathbf{s}_j^T\mathbf{r}_j + \alpha_{j+1}\mathbf{s}_{j+1}^T\mathbf{q}_{j+1} + \alpha_{j+1}\left(\tilde{\mathbf{q}}_{j+1} - \tilde{\mathbf{p}}_{j+1}\right)^T\mathbf{r}_j$.

Proof. (1) We first use the definition of $\mathbf{x}_0$ and $\mathbf{y}_0$

$$\begin{aligned}\mathbf{c}^T\mathbf{A}^{-1}\mathbf{b} &= \left(\mathbf{s}_0 + \mathbf{A}^T\mathbf{y}_{-1} + \tilde{\mathbf{C}}\check{\mathbf{C}}^T\mathbf{s}_{-1}\right)^T\mathbf{A}^{-1}\left(\mathbf{r}_0 + \mathbf{A}\mathbf{x}_{-1} + \mathbf{C}\hat{\mathbf{C}}^T\mathbf{r}_{-1}\right)\\ &= \mathbf{s}_0^T\mathbf{A}^{-1}\mathbf{r}_0 + \mathbf{s}_0^T\mathbf{x}_{-1} + \mathbf{s}_0^T\mathbf{A}^{-1}\mathbf{C}\hat{\mathbf{C}}^T\mathbf{r}_{-1}\\ &+ \mathbf{y}_{-1}^T\mathbf{r}_0 + \mathbf{y}_{-1}^T\mathbf{A}\mathbf{x}_{-1} + \mathbf{y}_{-1}^T\mathbf{C}\hat{\mathbf{C}}^T\mathbf{r}_{-1}\\ &+ \mathbf{s}_{-1}^T\check{\mathbf{C}}\tilde{\mathbf{C}}^T\mathbf{A}^{-1}\mathbf{r}_0 + \mathbf{s}_{-1}^T\check{\mathbf{C}}\tilde{\mathbf{C}}^T\mathbf{x}_{-1} + \mathbf{s}_{-1}^T\check{\mathbf{C}}\tilde{\mathbf{C}}^T\mathbf{A}^{-1}\mathbf{C}\hat{\mathbf{C}}^T\mathbf{r}_{-1}.\end{aligned}$$

We group the terms into five groups.

$$\begin{aligned}\mathbf{c}^T\mathbf{A}^{-1}\mathbf{b} &= \mathbf{s}_0^T\mathbf{A}^{-1}\mathbf{r}_0 + \left(\mathbf{s}_0^T\mathbf{x}_{-1} + \mathbf{s}_0^T\mathbf{A}^{-1}\mathbf{C}\hat{\mathbf{C}}^T\mathbf{r}_{-1}\right) + \left(\mathbf{y}_{-1}^T\mathbf{r}_0 + \mathbf{s}_{-1}^T\check{\mathbf{C}}\tilde{\mathbf{C}}^T\mathbf{A}^{-1}\mathbf{r}_0\right)\\ &+ \left(\mathbf{y}_{-1}^T\mathbf{A}\mathbf{x}_{-1} + \mathbf{s}_{-1}^T\check{\mathbf{C}}\tilde{\mathbf{C}}^T\mathbf{x}_{-1}\right) + \left(\mathbf{y}_{-1}^T\mathbf{C}\hat{\mathbf{C}}^T\mathbf{r}_{-1} + \mathbf{s}_{-1}^T\check{\mathbf{C}}\tilde{\mathbf{C}}^T\mathbf{A}^{-1}\mathbf{C}\hat{\mathbf{C}}^T\mathbf{r}_{-1}\right).\end{aligned}$$

Next, we use the definitions of $\mathbf{C} = \mathbf{A}\mathbf{U}$ and $\tilde{\mathbf{C}} = \mathbf{A}^T\tilde{\mathbf{U}}$ and the definitions of $\mathbf{x}_0$ and $\mathbf{y}_0$.

$$\begin{aligned}&\mathbf{s}_0^T\mathbf{A}^{-1}\mathbf{r}_0 + \mathbf{s}_0^T\underbrace{\left(\mathbf{x}_{-1} + \mathbf{U}\hat{\mathbf{C}}^T\mathbf{r}_{-1}\right)}_{\mathbf{x}_0} + \underbrace{\left(\mathbf{y}_{-1} + \tilde{\mathbf{U}}\check{\mathbf{C}}^T\mathbf{s}_{-1}\right)^T}_{\mathbf{y}_0^T}\mathbf{r}_0\\ &+ \underbrace{\left(\mathbf{y}_{-1} + \tilde{\mathbf{U}}\check{\mathbf{C}}^T\mathbf{s}_{-1}\right)^T}_{\mathbf{y}_0^T}\mathbf{A}\mathbf{x}_{-1} + \underbrace{\left(\mathbf{y}_{-1} + \tilde{\mathbf{U}}\check{\mathbf{C}}^T\mathbf{s}_{-1}\right)^T\mathbf{C}\hat{\mathbf{C}}^T}_{\mathbf{y}_0^T}\mathbf{s}_{-1}.\end{aligned}$$

We simplify further to obtain a shortened expression.

$$\begin{aligned}&\mathbf{s}_0^T\mathbf{A}^{-1}\mathbf{r}_0 + \mathbf{s}_0^T\mathbf{x}_0 + \mathbf{y}_0^T\mathbf{r}_0 + \mathbf{y}_0^T\mathbf{A}\mathbf{x}_{-1} + \mathbf{y}_0^T\mathbf{C}\hat{\mathbf{C}}^T\mathbf{s}_{-1}\\ &= \mathbf{s}_0^T\mathbf{A}^{-1}\mathbf{r}_0 + \mathbf{s}_0^T\mathbf{x}_0 + \mathbf{y}_0^T\underbrace{\left(\mathbf{r}_0 + \mathbf{A}\mathbf{x}_{-1} + \mathbf{C}\hat{\mathbf{C}}^T\mathbf{s}_{-1}\right)}_{\mathbf{b}} = \boxed{\mathbf{s}_0^T\mathbf{A}^{-1}\mathbf{r}_0 + \mathbf{s}_0^T\mathbf{x}_0 + \mathbf{y}_0^T\mathbf{b}}\end{aligned}$$

We use $\Phi_0 = \mathbf{s}_0^T\mathbf{x}_0 + \mathbf{y}_0^T\mathbf{b}$ as our initial approximation.

(2) Next, we derive an equation for $\mathbf{s}_j\mathbf{A}^{-1}\mathbf{r}_j - \mathbf{s}_{j+1}\mathbf{A}^{-1}\mathbf{r}_{j+1}$, similar to Lemma 4.1. In the rBiCG algorithm (Algorithm 6) from lines 8 and 9 we get

$$\begin{aligned}
&\mathbf{r}_j = \mathbf{r}_{j-1} - \alpha_j\mathbf{A}\mathbf{q}_j \quad \Leftrightarrow \quad \mathbf{r}_{j-1} = \mathbf{r}_j + \alpha_j\mathbf{A}\mathbf{q}_j \quad \Rightarrow \quad \mathbf{r}_j = \mathbf{r}_{j+1} + \alpha_{j+1}\mathbf{A}\mathbf{q}_{j+1},\\
&\mathbf{s}_j = \mathbf{s}_{j-1} - \alpha_j\mathbf{A}^T\tilde{\mathbf{q}}_j \quad \Leftrightarrow \quad \mathbf{s}_{j-1} = \mathbf{s}_j + \alpha_j\mathbf{A}^T\tilde{\mathbf{q}}_j \quad \Rightarrow \quad \mathbf{s}_j = \mathbf{s}_{j+1} + \alpha_{j+1}\mathbf{A}^T\tilde{\mathbf{q}}_{j+1}.
\end{aligned} \tag{4.8}$$

Replacing $\mathbf{s}_j$ and $\mathbf{r}_j$ with the expressions given above, we get

$$\begin{aligned}
\mathbf{s}_j\mathbf{A}^{-1}\mathbf{r}_j - \mathbf{s}_{j+1}\mathbf{A}^{-1}\mathbf{r}_{j+1} &= \left(\mathbf{s}_{j+1} + \alpha_{j+1}\mathbf{A}^T\tilde{\mathbf{q}}_{j+1}\right)^T\mathbf{A}^{-1}\left(\mathbf{r}_{j+1} + \alpha_{j+1}\mathbf{A}\mathbf{q}_{j+1}\right) - \mathbf{s}_{j+1}\mathbf{A}^{-1}\mathbf{r}_{j+1}\\
&= \alpha_{j+1}\mathbf{s}_{j+1}^T\mathbf{q}_{j+1} + \alpha_{j+1}\tilde{\mathbf{q}}_{j+1}^T\mathbf{r}_{j+1} + \alpha_{j+1}^2\tilde{\mathbf{q}}_{j+1}^T\mathbf{A}\mathbf{q}_{j+1}.
\end{aligned}$$

Substituting for $\tilde{\mathbf{q}}_j = \left(\mathbf{I} - \mathbf{U}\check{\mathbf{C}}^T\mathbf{A}^T\right)\tilde{\mathbf{p}}_j$ (Algorithm 6, line 14), we can simplify this expression

$$\begin{aligned}
&\alpha_{j+1}\mathbf{s}_{j+1}^T\mathbf{q}_{j+1} + \alpha_{j+1}\tilde{\mathbf{p}}_{j+1}^T\left(\mathbf{I} - \tilde{\mathbf{U}}\check{\mathbf{C}}^T\mathbf{A}^T\right)^T\mathbf{r}_{j+1} + \alpha_{j+1}^2\tilde{\mathbf{p}}_{j+1}^T\left(\mathbf{I} - \tilde{\mathbf{U}}\check{\mathbf{C}}^T\mathbf{A}^T\right)^T\mathbf{A}\mathbf{q}_{j+1}\\
&= \alpha_{j+1}\mathbf{s}_{j+1}^T\mathbf{q}_{j+1} + \alpha_{j+1}\tilde{\mathbf{p}}_{j+1}^T\left(\mathbf{I} - A\check{\mathbf{C}}\tilde{\mathbf{U}}^T\right)\mathbf{r}_{j+1} + \alpha_{j+1}^2\tilde{\mathbf{p}}_{j+1}^T\left(\mathbf{I} - \mathbf{A}\check{\mathbf{C}}\tilde{\mathbf{U}}^T\right)\mathbf{A}\mathbf{q}_{j+1}.
\end{aligned}$$

By properties of BiCG (see [80, 90]), we have $\tilde{\mathbf{p}}_{j+1} \perp \mathbf{r}_{j+1}$. From Algorithm 6, line 15, it follows that

$$\alpha_{j+1} = \frac{\langle \mathbf{s}_j, \mathbf{r}_j\rangle}{\langle \tilde{\mathbf{p}}_{j+1}, \mathbf{A}\mathbf{q}_{j+1}\rangle},$$

which implies $\mathbf{s}_j^T\mathbf{r}_j = \alpha_{j+1}\tilde{\mathbf{p}}_{j+1}^T\mathbf{A}\mathbf{q}_{j+1}$.

Hence, we obtain

$$\begin{aligned}
&\alpha_{j+1}\mathbf{s}_{j+1}^T\mathbf{q}_{j+1} + \underbrace{\alpha_{j+1}\tilde{\mathbf{p}}_{j+1}^T\mathbf{r}_{j+1}}_{=0} - \alpha_{j+1}\tilde{\mathbf{p}}_{j+1}^T\mathbf{A}\check{\mathbf{C}}\tilde{\mathbf{U}}^T\mathbf{r}_{j+1}\\
&+ \alpha_{j+1}\underbrace{\left(\alpha_{j+1}\tilde{\mathbf{p}}_{j+1}^T\mathbf{A}\mathbf{q}_{j+1}\right)}_{=\mathbf{s}_j^T\mathbf{r}_j} - \alpha_{j+1}^2\tilde{\mathbf{p}}_{j+1}^T\mathbf{A}\check{\mathbf{C}}\tilde{\mathbf{U}}^T\mathbf{A}\mathbf{q}_{j+1}.
\end{aligned}$$

From (4.8) we have $\mathbf{r}_j - \mathbf{r}_{j+1} = \alpha_{j+1}\mathbf{A}\mathbf{q}_{j+1}$, and thus

$$\begin{aligned}
&\alpha_{j+1}\mathbf{s}_j^T\mathbf{r}_j + \alpha_{j+1}\mathbf{s}_{j+1}^T\mathbf{q}_{j+1} - \alpha_{j+1}\tilde{\mathbf{p}}_{j+1}^T\mathbf{A}\check{\mathbf{C}}\tilde{\mathbf{U}}^T\mathbf{r}_{j+1} - \alpha_{j+1}\tilde{\mathbf{p}}_{j+1}^T\mathbf{A}\check{\mathbf{C}}\tilde{\mathbf{U}}^T\left(\mathbf{r}_j - \mathbf{r}_{j+1}\right)\\
&= \alpha_{j+1}\mathbf{s}_j^T\mathbf{r}_j + \alpha_{j+1}\mathbf{s}_{j+1}^T\mathbf{q}_{j+1} - \alpha_{j+1}\left(\tilde{\mathbf{U}}\check{\mathbf{C}}^T\mathbf{A}^T\tilde{\mathbf{p}}_{j+1}\right)^T\mathbf{r}_j.
\end{aligned}$$

The expressions in (4.7) lead to $\tilde{\mathbf{q}}_{j+1} - \tilde{\mathbf{p}}_{j+1} = -\tilde{\mathbf{U}}\check{\mathbf{C}}^T\mathbf{A}^T\tilde{\mathbf{p}}_{j+1}$.

$$\boxed{\alpha_{j+1}\mathbf{s}_j^T\mathbf{r}_j + \alpha_{j+1}\mathbf{s}_{j+1}^T\mathbf{q}_{j+1} + \alpha_{j+1}\left(\tilde{\mathbf{q}}_{j+1} - \tilde{\mathbf{p}}_{j+1}\right)^T\mathbf{r}_j}.$$

■

Next, we follow a derivation analogous to (4.2). Hence, for any $N > 0$ we get

$$\begin{aligned}
\mathbf{c}^T\mathbf{A}^{-1}\mathbf{b} =& \mathbf{s}_0^T\mathbf{A}^{-1}\mathbf{r}_0 + \mathbf{s}_0^T\mathbf{x}_0 + \mathbf{y}_0^T\mathbf{b} \\
=& \left(\mathbf{s}_0^T\mathbf{x}_0 + \mathbf{y}_0^T\mathbf{b}\right) + \left(\mathbf{s}_0^T\mathbf{A}^{-1}\mathbf{r}_0 - \mathbf{s}_1^T\mathbf{A}^{-1}\mathbf{r}_1\right) \\
&+ \left(\mathbf{s}_1^T\mathbf{A}^{-1}\mathbf{r}_1 - \mathbf{s}_2^T\mathbf{A}^{-1}\mathbf{r}_2\right) + \dots \\
&+ \left(\mathbf{s}_{N-1}^T\mathbf{A}^{-1}\mathbf{r}_{N-1} - \mathbf{s}_N^T\mathbf{A}^{-1}\mathbf{r}_N\right) \\
&+ \mathbf{s}_N^T\mathbf{A}^{-1}\mathbf{r}_N \\
=& \underbrace{\left(\mathbf{s}_0^T\mathbf{x}_0 + \mathbf{y}_0^T\mathbf{b}\right)}_{\Phi_0} + \sum_{j=0}^{N-1}\left(\alpha_{j+1}\mathbf{s}_j^T\mathbf{r}_j + \alpha_{j+1}\mathbf{s}_{j+1}^T\mathbf{q}_{j+1} + \alpha_{j+1}\left(\tilde{\mathbf{q}}_{j+1} - \tilde{\mathbf{p}}_{j+1}\right)^T\mathbf{r}_j\right) + \mathbf{s}_N\mathbf{A}^{-1}\mathbf{r}_N
\end{aligned} \tag{4.9}$$

If the method converges, we have $\left|\mathbf{s}_N\mathbf{A}^{-1}\mathbf{r}_N\right| < \varepsilon$ for some N. This gives

$$\left|\mathbf{c}^T\mathbf{A}^{-1}\mathbf{b} - \Phi_0 - \sum_{j=0}^{N-1}\left(\alpha_{j+1}\mathbf{s}_j^T\mathbf{r}_j + \alpha_{j+1}\mathbf{s}_{j+1}^T\mathbf{q}_{j+1} + \alpha_{j+1}\left(\tilde{\mathbf{q}}_{j+1} - \tilde{\mathbf{p}}_{j+1}\right)^T\mathbf{r}_j\right)\right| < \varepsilon. \tag{4.10}$$

We define Φ_N as the estimate to the bilinear form computed in N iterations of BiCG, i.e.,

$$\Phi_N = \sum_{j=0}^{N-1}\left(\alpha_{j+1}\mathbf{s}_j^T\mathbf{r}_j + \alpha_{j+1}\mathbf{s}_{j+1}^T\mathbf{q}_{j+1} + \alpha_{j+1}\left(\tilde{\mathbf{q}}_{j+1} - \tilde{\mathbf{p}}_{j+1}\right)^T\mathbf{r}_j\right) \tag{4.11}$$

The recycled BiCG algorithm equipped with the bilinear form estimate is given in Algorithm 9. This new algorithm is called rBiCG-bf. The highlighted lines indicate where rBiCG-bf differs from rBiCG. The starting point is the rBiCG algorithm in [3], p. 32.

4.3.3 Quadratic form estimates using CG and recycling CG.

For the CG algorithm, we perform similar derivations as for rBiCG. Most of the derivations are the same for CG and rCG.

Let $\mathbf{r}_{j+1}$ denote the residual computed at iteration $j+1$ of CG (Algorithm 2, line 15) and $\mathbf{r}_j$ denote the residual computed at iteration j of CG.

For both CG and rCG we have

$$\mathbf{r}_{j+1} = \mathbf{r}_j - \alpha_j\mathbf{A}\mathbf{p}_j \quad \Rightarrow \quad \mathbf{r}_j = \mathbf{r}_{j+1} + \alpha_j\mathbf{A}\mathbf{p}_j. \tag{4.12}$$

Lemma 4.4. *Using the recurrences in rCG (Algorithm 8), we have*

$$\mathbf{r}_{j+1}^T\mathbf{p}_j = 0.$$

Algorithm 9 rBiCG-bf

$\mathbf{x}_{-1}$ and $\mathbf{y}_{-1}$ given initial guesses;
$\mathbf{U}$, $\tilde{\mathbf{U}}$ given; we want to use $range(\mathbf{U})$ and $range(\tilde{\mathbf{U}})$ as recycle spaces; if no $\mathbf{U}$, $\tilde{\mathbf{U}}$, set to empty;
Compute $\mathbf{x}_0$ and $\mathbf{y}_0$, $\mathbf{r}_0$ and $\mathbf{s}_0$, $\hat{\mathbf{C}}$ and $\check{\mathbf{C}}$;
if $\langle \mathbf{r}_0, \mathbf{s}_0 \rangle = 0$ **then**
  $\mathbf{y}_0 \leftarrow$ random vector;
  Recompute $\mathbf{s}_0$;
end if
Set $\Phi_0 \leftarrow \mathbf{s}_0^T \mathbf{x}_0 + \mathbf{y}_0^T \mathbf{b}$;
Set $\beta_0 \leftarrow 0$, $\mathbf{p}_0 \leftarrow 0$, $\tilde{\mathbf{p}} \leftarrow 0$;
Choose `maxit` and `tol`;
for $k \leftarrow 1$ to maxit **do**
  $\mathbf{p}_k \leftarrow \mathbf{r}_{k-1} + \beta_{k-1} \mathbf{p}_{k-1}$;
  $\tilde{\mathbf{p}}_k \leftarrow \tilde{r}_{k-1} + \beta_{i-1} \tilde{\mathbf{p}}_{k-1}$;
  $\mathbf{q}_k \leftarrow \left(\mathbf{I} - \mathbf{U}\hat{\mathbf{C}}^T \mathbf{A}\right) \mathbf{p}_k$;
  $\tilde{\mathbf{q}}_k \leftarrow \left(\mathbf{I} - \tilde{\mathbf{U}}\check{\mathbf{C}}^T \mathbf{A}^T\right) \tilde{\mathbf{p}}_k$;
  $\alpha_k \leftarrow \frac{\langle \mathbf{s}_{k-1}, \mathbf{r}_{k-1} \rangle}{\langle \tilde{\mathbf{p}}_k, \mathbf{A}\mathbf{q}_k \rangle}$;
  $\mathbf{x}_k \leftarrow \mathbf{x}_{k-1} + \alpha_k \mathbf{q}_k$;
  $\mathbf{y}_k \leftarrow \mathbf{y}_{k-1} + \alpha_k \tilde{\mathbf{q}}_k$;
  $\mathbf{r}_k \leftarrow \mathbf{r}_{k-1} - \alpha_k A \mathbf{q}_k$;
  $\mathbf{s}_k \leftarrow \mathbf{s}_{k-1} - \alpha_k \mathbf{A}^T \tilde{\mathbf{q}}_k$;
  $\Phi_k \leftarrow \Phi_{k-1} + \alpha_k \mathbf{s}_{k-1}^T \mathbf{r}_{k-1} + \alpha_k \mathbf{s}_k^T \mathbf{q}_k + \alpha_k * (\tilde{\mathbf{q}}_k - \tilde{\mathbf{p}}_k)^T \mathbf{r}_{k-1}$;
  if $\|\mathbf{r}_k\| \|\mathbf{s}_k\| < tol$ **then**
    break
  end if
  $\beta_k \leftarrow \frac{\langle \mathbf{s}_k, \mathbf{r}_k \rangle}{\langle \mathbf{s}_{k-1}, \mathbf{r}_{k-1} \rangle}$;
end for

Proof. (1) We show $\mathbf{r}_1^T \mathbf{p}_0 = 0$. Note $\alpha_0 = \frac{\mathbf{r}_0^T \mathbf{r}_0}{\mathbf{p}_0^T \mathbf{A} \mathbf{p}_0}$ and by construction, $\mathbf{r}_0^T \mathbf{U} = 0$. Thus

$$\begin{aligned}
\mathbf{r}_1^T \mathbf{p}_0 &= \left(\mathbf{r}_0 - \alpha_0 \mathbf{A}\mathbf{p}_0\right)^T \mathbf{p}_0 = \mathbf{r}_0^T \mathbf{p}_0 - \frac{\mathbf{r}_0^T \mathbf{r}_0}{\cancel{\mathbf{p}_0^T \mathbf{A}\mathbf{p}_0}} \cancel{\mathbf{p}_0^T \mathbf{A}\mathbf{p}_0} = \mathbf{r}_0^T \mathbf{p}_0 - \mathbf{r}_0^T \mathbf{r}_0 \\
&= \mathbf{r}_0^T \left(\mathbf{r}_0 - \mathbf{U}\left(\mathbf{U}^T \mathbf{A}\mathbf{U}\right)^{-1} \mathbf{r}_0\right) - \mathbf{r}_0^T \mathbf{r}_0 \\
&= \mathbf{r}_0^T \mathbf{r}_0 - \mathbf{r}_0^T \mathbf{r}_0 = 0.
\end{aligned}$$

(2) Induction hypothesis. We assume $\mathbf{r}_{k+1}^T \mathbf{p}_k = 0$ for all $k < j$.

(3) Inductive step.

$$\begin{aligned}\mathbf{r}_{j+1}^T\mathbf{p}_j &= \left(\mathbf{r}_j - \alpha_0\mathbf{A}\mathbf{p}_j\right)^T\mathbf{p}_j = \mathbf{r}_j^T\mathbf{p}_j - \frac{\mathbf{r}_j^T\mathbf{r}_j}{\cancel{\mathbf{p}_j^T\mathbf{A}\mathbf{p}_j}}\cancel{\mathbf{p}_j^T\mathbf{A}\mathbf{p}_j}\\ &= \mathbf{r}_j^T\mathbf{p}_j - \mathbf{r}_j^T\mathbf{r}_j = \mathbf{r}_j^T\left(\beta_{j-1}\mathbf{p}_{j-1} + \mathbf{r}_j - \mathbf{U}\mu_j\right) - \mathbf{r}_j^T\mathbf{r}_j\\ &= \underbrace{\beta_{j-1}\mathbf{r}_j^T\mathbf{p}_{j-1}}_{=0 \text{ by ind. assumption}} + \mathbf{r}_j^T\mathbf{r}_j - \mathbf{r}_j^T\mathbf{U}\mu_j - \mathbf{r}_j^T\mathbf{r}_j = 0.\end{aligned}$$

■

Next, we apply derivations similar to those for BiCG. Hence, we need to derive a formula for $\mathbf{r}_j^T\mathbf{A}^{-1}\mathbf{r}_j - \mathbf{r}_{j+1}^T\mathbf{A}^{-1}\mathbf{r}_{j+1}$, similar to the one in Lemma 4.1. The difference between CG and rCG lies in the way we define $\mathbf{p}_j$ vectors. However, the definition of $\mathbf{p}_j$ is not used in the derivations of the formula for $\mathbf{r}_j^T\mathbf{A}^{-1}\mathbf{r}_j - \mathbf{r}_{j+1}^T\mathbf{A}^{-1}\mathbf{r}_{j+1}$. Therefore, the derivations hold for both CG and rCG.

Lemma 4.5. *For both, CG and rCG we have*

(1) $\mathbf{b}^T\mathbf{A}^{-1}\mathbf{b} = \mathbf{r}_0^T\mathbf{A}^{-1}\mathbf{r}_0 + \mathbf{r}_0^T\mathbf{x}_0 + \mathbf{x}_0^T\mathbf{b}$,

(2) $\mathbf{r}_j^T\mathbf{A}^{-1}\mathbf{r}_j - \mathbf{r}_{j+1}^T\mathbf{A}^{-1}\mathbf{r}_{j+1} = \alpha_j\mathbf{r}_j^T\mathbf{r}_j$.

Proof. For rCG $\mathbf{x}_{-1}$ is arbitrary but $\mathbf{x}_0$ is not; we choose $\mathbf{r}_0$ to satisfy $\mathbf{r}_0 \perp \mathbf{U}$. However, we start rCG with $\mathbf{x}_0$ ($\mathbf{x}_0$ it is not generated by rCG algorithm).

(1) $\mathbf{b} = \mathbf{r}_0 + \mathbf{A}\mathbf{x}_0$, hence

$$\mathbf{b}^T\mathbf{A}^{-1}\mathbf{b} = \left(\mathbf{r}_0 + \mathbf{A}\mathbf{x}_0\right)^T\mathbf{A}^{-1}\left(\mathbf{r}_0 + \mathbf{A}\mathbf{x}_0\right) = \mathbf{r}_0^T\mathbf{A}^{-1}\mathbf{r}_0 + \mathbf{r}_0^T\mathbf{x}_0 + \mathbf{x}_0^T\mathbf{r}_0 + \mathbf{x}_0^T\mathbf{A}\mathbf{x}_0 = \mathbf{r}_0^T\mathbf{A}^{-1}\mathbf{r}_0 + \mathbf{r}_0^T\mathbf{x}_0 + \mathbf{x}_0^T\mathbf{b}$$

(2) We use $\mathbf{r}_j = \mathbf{r}_{j+1} + \alpha_j\mathbf{A}\mathbf{p}_j$ and we compute

$$\begin{aligned}&\mathbf{r}_j^T\mathbf{A}^{-1}\mathbf{r}_j - \mathbf{r}_{j+1}^T\mathbf{A}^{-1}\mathbf{r}_{j+1}\\ &= \left(\mathbf{r}_{j+1} + \alpha_j\mathbf{A}\mathbf{p}_j\right)^T\mathbf{A}^{-1}\left(\mathbf{r}_{j+1} + \alpha_j\mathbf{A}\mathbf{p}_j\right) - \mathbf{r}_{j+1}^T\mathbf{A}^{-1}\mathbf{r}_{j+1}\\ &= \underbrace{\alpha_j\mathbf{r}_{j+1}^T\mathbf{p}_j}_{=0} + \underbrace{\alpha_j\mathbf{p}_j^T\mathbf{r}_{j+1}}_{=0} + \alpha_j^2\mathbf{p}_j^T\mathbf{A}\mathbf{p}_j = \alpha_j\frac{\mathbf{r}_j^T\mathbf{r}_j}{\mathbf{p}_j\mathbf{A}\mathbf{p}_j}\mathbf{p}_j^T\mathbf{A}\mathbf{p}_j\\ &= \boxed{\alpha_j\mathbf{r}_j^T\mathbf{r}_j}\end{aligned}$$

■

Now, we can derive a formula to estimate the quadratic form. Let $N > 0$

$$\begin{aligned}\mathbf{b}^T\mathbf{A}^{-1}\mathbf{b} &= \mathbf{r}_0^T\mathbf{A}^{-1}\mathbf{r}_0 + \underbrace{\mathbf{r}_0^T\mathbf{x}_0 + \mathbf{x}_0^T\mathbf{b}}_{\Psi_0} \\ &= \mathbf{r}_0^T\mathbf{A}^{-1}\mathbf{r}_0 - \mathbf{r}_1^T\mathbf{A}^{-1}\mathbf{r}_1 + \mathbf{r}_1^T\mathbf{A}^{-1}\mathbf{r}_1 - \mathbf{r}_2^T\mathbf{A}^{-1}\mathbf{r}_2 \\ &+ \ldots + \mathbf{r}_{N-1}^T\mathbf{A}^{-1}\mathbf{r}_{N-1} - \mathbf{r}_N^T\mathbf{A}^{-1}\mathbf{r}_N + \mathbf{r}_N^T\mathbf{A}^{-1}\mathbf{r}_N + \Psi_0 \\ &= \Psi_0 + \mathbf{r}_N^T\mathbf{A}^{-1}\mathbf{r}_N + \sum_{j=1}^{N-1}\alpha_j\mathbf{r}_j^T\mathbf{r}_j\end{aligned}$$

In the derivation above, we use $\alpha_0 = 0$. If the method converges, for some N we have $\left|\mathbf{s}_N\mathbf{A}^{-1}\mathbf{r}_N\right| < \varepsilon$, which gives

$$\left|\mathbf{b}^T\mathbf{A}^{-1}\mathbf{b} - \Psi_0 - \sum_{j=1}^{N-1}\alpha_j\mathbf{r}_j^T\mathbf{r}_j\right| < \varepsilon. \tag{4.13}$$

The resulting algorithm is listed as Algorithm 10 (for CG) and Algorithm 11 (for rCG). We call the first algorithm CG-qf and the second algorithm rCG-qf. If the recycle space is not used, the rCG-qf is equivalent to CG-qf. The new lines, in which the algorithm computes the approximation to the quadratic form, have been highlighted. The starting point for CG-qf is the CG algorithm from [95] and the starting point for the rCG-qf is the algorithm from [70].

4.3.4 Remarks on error estimates

A natural question is whether estimating the bilinear form through BiCG and rBiCG (formulae (4.4) and (4.10)) rather than using the inner product of the vector $\mathbf{c}$ with the BiCG solution $\mathbf{x}_N$ is worth the effort. It turns out that we can expect a better estimate with the (r)BiCG-bf method than with inner product [90].

The error in the bilinear form approximations (4.4) and (4.10) can be expressed as

$$\mathbf{c}^T\mathbf{A}^{-1}\mathbf{b} - \Psi_N = \mathbf{s}_N^T\mathbf{A}^{-1}\mathbf{r}_N = \mathbf{s}_N^T\mathbf{A}^{-1}\left(\mathbf{b} - \mathbf{A}\mathbf{x}_N\right) = \mathbf{s}_N^T\left(\tilde{\mathbf{x}} - \mathbf{x}_N\right), \tag{4.14}$$

where $\tilde{\mathbf{x}} = \mathbf{A}^{-1}\mathbf{b}$ and Ψ_N is the approximation to the bilinear form computed in N iterations of (r)BiCG-bf. For BiCG and rBiCG, the difference between the bilinear form and the approximation is given by $\mathbf{s}_N^T\mathbf{A}^{-1}\mathbf{r}_N$. In (4.14), we use only the definition of the error and the residual, hence (4.14) holds for both methods.

Lemma 4.6. *[90] Let $\mathbf{x}_N$ and $\mathbf{y}_N$ be the approximate solutions at the Nth iteration of (r)BiCG-bf applied to the problems $\mathbf{A}\mathbf{x} = \mathbf{b}$, $\mathbf{A}^T\mathbf{y} = \mathbf{c}$*

For both, BiCG and rBiCG we have

$$\mathbf{c}^T\mathbf{A}^{-1}\mathbf{b} - \mathbf{c}^T\mathbf{x}_N = \mathbf{s}_N^T\left(\tilde{\mathbf{x}} - \mathbf{x}_N\right) + \mathbf{y}_N^T\mathbf{r}_N, \tag{4.15}$$

where $\tilde{\mathbf{x}} = \mathbf{A}^{-1}\mathbf{b}$.

Algorithm 10 CG-qf

$\mathbf{x}_0$ given initial guess;
Compute $\mathbf{r}_0$;
Set $\Psi_0 \leftarrow \mathbf{r}_0^T\mathbf{x}_0 + \mathbf{x}_0^T\mathbf{b}$;
Choose `maxit` and `tol`;
for $k \leftarrow 1$ to maxit **do**
  $\rho_{k-1} \leftarrow \mathbf{r}_{k-1}^T\mathbf{r}_{k-1}$;
  if $k == 1$ **then**
    $\mathbf{p}_k \leftarrow \mathbf{r}_{k-1}$;
  else
    $\beta_{k-1} \leftarrow \frac{\rho_{k-1}}{\rho_{k-2}}$;
    $\mathbf{p}_k = \mathbf{r}_{k-1} + \beta_{k-1}\mathbf{p}_{k-1}$;
  end if
  $\mathbf{w}_k \leftarrow \mathbf{A}\mathbf{p}_k$;
  $\alpha_k \leftarrow \frac{\rho_{k-1}}{\mathbf{p}_k^T\mathbf{w}_k}$
  $\mathbf{x}_k \leftarrow \mathbf{x}_{k-1} + \alpha_k\mathbf{p}_k$;
  $\mathbf{r}_k \leftarrow \mathbf{r}_{k-1} - \alpha_k\mathbf{w}_k$;
  $\Psi_k \leftarrow \Psi_{k-1} + \alpha_k\mathbf{r}_k^T\mathbf{r}_k$
  if $\|\mathbf{r}_k\| < tol$ **then**
    break
  end if
end for

Proof.

$$\begin{aligned}\mathbf{c}^T\mathbf{A}^{-1}\mathbf{b} - \mathbf{c}^T\mathbf{x}_N &= \mathbf{c}^T\left(\mathbf{A}^{-1}\mathbf{b} - \mathbf{x}_N\right)\\ &= \left(\mathbf{s}_N + \mathbf{A}^T\mathbf{y}_N\right)^T\left(\mathbf{A}^{-1}\mathbf{b} - \mathbf{x}_N\right)\\ &= \mathbf{s}_N^T\left(\tilde{\mathbf{x}} - \mathbf{x}_N\right) + \mathbf{y}_N^T\left(\mathbf{b} - \mathbf{A}\mathbf{x}_N\right)\\ &= \mathbf{s}_N^T\left(\tilde{\mathbf{x}} - \mathbf{x}_N\right) + \mathbf{y}_N^T\mathbf{r}_N.\end{aligned}$$

■

The proof holds for both methods because we use the definitions of residuals, which are not affected by the method formulation.

In an analogous way we can derive

$$\mathbf{c}^T\mathbf{A}^{-1}\mathbf{b} - \mathbf{y}_N^T\mathbf{b} = \left(\mathbf{y}_N - \tilde{\mathbf{y}}_N\right)^T\mathbf{r}_N + \mathbf{s}_N^T\mathbf{x}_N.$$

Corrollary 4.7. *(1) The difference between the approximation to the bilinear form* $\mathbf{c}^T\mathbf{A}^{-1}\mathbf{b}$ *computed in* N *iterations of (r)BiCG, using (4.4) or (4.10), and the approximation computed using* $\mathbf{c}^T\mathbf{x}_N$*, is* $\mathbf{y}_N^T\mathbf{r}_N$.

Algorithm 11 rCG-qf

$\mathbf{x}_{-1}$ given initial guess;
$\mathbf{U}$ given; we want to use $range(\mathbf{U})$ as a recycle space;
Compute $\mathbf{x}_0$, $\mathbf{r}_0$;
Set $\Psi \leftarrow \mathbf{r}_0^T\mathbf{x}_0 + \mathbf{x}_0^T\mathbf{b}$;
Set $\mathbf{p}_0 \leftarrow \mathbf{r}_0$;
Choose `maxit` and `tol`;
for $i \leftarrow 1$ to maxit **do**
  if $i > 1$ **then**
    $\alpha_{i-1} \leftarrow \frac{\langle \mathbf{r}_{k-1}, \mathbf{r}_{k-1} \rangle}{\langle \mathbf{p}_{k-1}, \mathbf{A}\mathbf{p}_{k-1} \rangle}$;
  else
    $\alpha_{i-1} \leftarrow 0$
  end if
  $\mathbf{x}_k \leftarrow \mathbf{x}_{k-1} + \alpha_{i-1}\mathbf{p}_{k-1}$;
  $\mathbf{r}_k \leftarrow \mathbf{r}_{k-1} - \alpha_{i-1}A\mathbf{p}_{k-1}$;
  $\beta_{i-1} \leftarrow \frac{\langle \mathbf{r}_k, \mathbf{r}_k \rangle}{\langle \mathbf{r}_{k-1}, \mathbf{r}_{k-1} \rangle}$;
  Solve for μ_i: $U^T\mathbf{A}U\mu_i = (AU)^T\mathbf{r}_k$;
  $\mathbf{p}_k \leftarrow \beta_{i-1}\mathbf{p}_{k-1} + \mathbf{r}_k - U\mu_i$;
  $\Psi \leftarrow \Psi + \alpha_{i-1}\mathbf{r}_{k-1}^T\mathbf{r}_{k-1}$;
  if $\|\mathbf{r}_k\| < tol$ **then**
    break
  end if
end for

(2) The difference between the approximation to bilinear form $\mathbf{c}^T\mathbf{A}^{-1}\mathbf{b}$ *computed in* N *iterations of (r)BiCG, using (4.4) or (4.10), and the approximation computed using* $\mathbf{y}_N^T\mathbf{b}$, *is* $\mathbf{s}_N^T\mathbf{x}_N$.

In exact arithmetic, we have $\mathbf{r}_N \perp \mathcal{K}^N(\mathbf{A}^T; \mathbf{s}_0)$ and $\mathbf{s}_N \perp \mathcal{K}^N(\mathbf{A}; \mathbf{r}_0)$. By construction, we also have $\mathbf{y}_N = \mathbf{y}_0 + \mathbf{z}_N$, with $\mathbf{z}_N \in \mathcal{K}^N(\mathbf{A}^T; \mathbf{s}_0)$. Let range$(\mathbf{W}_N) = \mathcal{K}^N(\mathbf{A}^T; \mathbf{s}_0)$. Hence, there exists a vector $\mathbf{g}$ such that $\mathbf{y}_N = \mathbf{y}_0 + \mathbf{W}_N\mathbf{g}$. It follows that

$$\mathbf{y}_N^T\mathbf{r}_N = (\mathbf{y}_0 + \mathbf{W}_N\mathbf{g})^T\mathbf{r}_N = \mathbf{y}_0^T\mathbf{r}_N.$$

In exact arithmetic, if $\mathbf{y}_0$ is a zero vector, then both approaches (the (r)BiCG-qf approach and the inner product approach) are equivalent. According to [90], similar analysis for (r)CG is not easy and both estimates are close.

4.4 Results

4.4.1 Topology optimization

In Chapter 2, we discussed the optimization process associated with the topology optimization. In the method from Chapter 2, we need to perform many optimization steps to obtain an optimal design. In every optimization step, we evaluate the compliance function and its gradient.

We define compliance using (2.13), i.e.,

$$\mathbf{C}(\boldsymbol{\rho}) = \mathrm{Tr}\left(\mathbf{F}^T \mathbf{K}^{-1}(\boldsymbol{\rho})\,\mathbf{F}\right), \tag{4.16}$$

where $\mathbf{K}(\boldsymbol{\rho})$ is the stiffness matrix and $\mathbf{F} = [\mathbf{f}_1\,\mathbf{f}_2, \ldots\,\mathbf{f}_m]$, with $\mathbf{f}_1, \ldots, \mathbf{f}_m$ being the load cases. We can write (4.16) as

$$\sum_{j=1}^{m} \mathbf{f}_j \mathbf{K}^{-1}(\boldsymbol{\rho})\,\mathbf{f}_j. \tag{4.17}$$

Therefore, the compliance is a sum of m quadratic forms, where m is the number of load cases. We also need an approximate solutions to the systems

$$\mathbf{K}(\boldsymbol{\rho})\,\mathbf{u} = \mathbf{f}_j, \quad j = 1, 2, \ldots, m, \tag{4.18}$$

because these solutions are used to evaluate the gradient (see (2.14)).

Typically, generating an optimal design requires large number of steps in the optimization process. A common approach is to use a direct solver or CG to approximately solve the systems (4.18) and to estimate the compliance using inner products, i.e., $\sum_{j=1}^{m} \mathbf{f}_j^T \mathbf{u}_j$, where $\mathbf{u_j}$ is an approximate solution to $\mathbf{K}(\boldsymbol{\rho})\,\mathbf{u} = \mathbf{f}_j$, $j = 1, 2, \ldots, m$. This approach is very expensive and impossible to apply for large 3D problems.

The number of linear solves can be reduced using randomized approach, see Section 2.2. However, with the randomized approach we are still required to solve multiple linear systems per optimization step. In topology optimization problems, we need an accurate estimate for the compliance and a modest accuracy estimate for the solutions to (4.18). We can further reduce the computational cost by using recycling CG and the method of evaluating quadratic forms described in Section 4.3.3.

As a test case, we use a 3D bridge problem whose discretization contains 270000 elements. The corresponding stiffness matrix has size 844325×844325, and we use 36 load cases. The optimization process requires 941 steps to produce an optimal design. Hence, we need to evaluate 33876 quadratic forms. The resulting optimal design is shown in Figure 4.1

The Table 4.1 shows the results we obtained for the 3D bridge problem. If we want to use direct solver, in every optimization step we compute a Cholesky decomposition of the stiffness matrix,

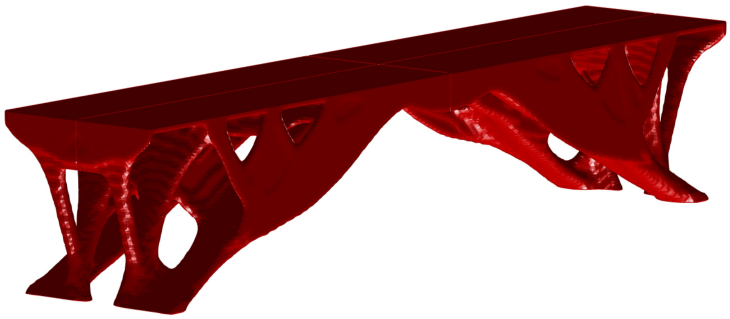


Figure 4.1: Large bridge (844325×844325 stiffness matrix) generated using the stochastic approach combined with rCG-qf with incomplete Cholesky preconditioner. The bridge was generated using a MATLAB implementation. The optimization process took 941 optimization steps and approximately 30h on a server with 96 GB RAM.

Method	**# of qfs**	**Time per qf**	**Time for all qfs**	**Speedup**
Direct solve	33876	368 seconds	144.2 days (approx.)	1×
Direct solve w/rand	5646	368 seconds	23 days (approx.)	6×
rCG-qf w/rand	5646	19.7 seconds	30 hours (approx.)	112×

Table 4.1: Results for a 3D bridge problem. The stiffness matrix has size 844325×844325, and there are 36 load cases in the original problem. The results were obtained using matlab, on a server with 96 GB RAM. In the table, **qf** denotes 'quadratic form'.

i.e., $\mathbf{K(p)} = \mathbf{LL}^T$. Next, we compute $\mathbf{u}_j = \mathbf{K(p)}^{-1}\mathbf{f}_j = \left(\mathbf{LL}^T\right)^{-1}\mathbf{f}_j$ for $j = 1, 2, \ldots, 36$, which requires two triangular solves. The Cholesky decomposition takes $640s$ and two triangular solves take 350s (total). Time per optimization step is then $360s + 36 \cdot 350s = 156min$, and the average time per quadratic form is $368s$. Because we need to perform 941 optimization steps, the process would take approximately 144.2 days.

The number of load cases (and quadratic forms) can be reduced by a stochastic approach derived in [100] and discussed in Chapter 2. In our case, we reduce the number of load cases to 6. The process still would take 23 days.

Instead, we can use iterative solver with fairly low tolerance (relative tolerance $10e-2$). Since we combined the iterative solver with quadratic form estimation (see Section 4.3.3), rCG-qf produces the quadratic form with the error of approximately $10e-4$, which is enough for making the optimization process converge. We accelerate the rCG-qf solver using an incomplete Cholesky preconditioner. The optimization process takes around 30h, which gives us 20× speedup, compared with the direct solver with the randomized approach, and 112× speedup compared with the brute-force approach.

The results for this problem show that the combination of randomization, quadratic form

estimation (which allows us to solve the linear systems using CG with low tolerance), and recycling (rCG), lets us substantially reduce the cost of the optimization process for the 3D bridge test problem. The process can be speed up further if we use parallel computing (all the sensitivities can be computed in parallel).

The test problem is relatively small. The problems arising in topology optimization are often much larger, and have many more load cases. Our combined strategy (randomization, quadratic form estimation and recycling) is even more advantageous for such problems.

4.4.2 Diffuse Optical Tomography

In Chapter 2, we provide the details of the optimization process associated with the Diffuse Optical Tomography (DOT) problem. In the current section, we consider the DOT problem with 32 detectors, 32 sources, and one frequency. In the non-stochastic variant of the DOT, we need to evaluate $\mathbf{c}_k^T \mathbf{A}^{-1} \mathbf{b}_j$ where $j = 1, \ldots, 32$ and $k = 1, \ldots, 32$, which leads to 1024 bilinear forms per optimization step. TREGS (see Chapter 2) does not estimate the Jacobian at every optimization step but, if the Jacobian is required, we also need high-accuracy estimates for $\mathbf{x}_{(j)}$ and $\mathbf{y}_{(k)}$, where $\mathbf{x}_{(j)}$ is an approximate solution to $\mathbf{A}\mathbf{x} = \mathbf{b}_j$ and $\mathbf{y}_{(k)}$ is an approximate solution to $\mathbf{A}^T\mathbf{y} = \mathbf{c}_k$, $j = 1, \ldots, 32$ and $k = 1, \ldots, 32$.

One can approach this problem in at least two different ways. First, we can solve all the systems

$$\mathbf{A}\mathbf{x} = \mathbf{b}_j, \tag{4.19}$$

for $j = 1, 2, \ldots, 32$ to high relative tolerance, using a method such as GMRES(m) (Chapter 3). Let $\mathbf{x}_{(j)}$ denote the approximate solution to (4.19) (Note: we use brackets to differentiate between an approximate solution to the system (j) and an approximate solution computed in iteration j). In that case, we estimate bilinear forms using $\Psi_{k,j} = \mathbf{c}_k^T \mathbf{x}_{(j)}$, $k = 1, \ldots, 32$. Let $\|\mathbf{r}_{(j)}\| = \|\mathbf{b_j} - \mathbf{A}\mathbf{x}_{(j)}\|$ be the residual. Let $\tilde{\mathbf{x}}_{(j)}$ be the true solution to (4.19). Then

$$|\mathbf{c}_k^T \mathbf{x}_{(j)} - \mathbf{c}_k^T \tilde{\mathbf{x}}_{(j)}| = |\mathbf{c}_k^T \left(\mathbf{x}_{(j)} - \tilde{\mathbf{x}}_{(j)}\right)| = |\mathbf{c}_k \mathbf{A}^{-1} \mathbf{r}_{(\mathbf{j})}|. \tag{4.20}$$

Since solutions to the adjoint systems are needed for the Jacobian, we perform an additional sequence of solves $\mathbf{A}^T\mathbf{y} = \mathbf{c}_k$ for $k = 1, \ldots, 32$, so 64 total solves are then needed.

The second method relies on (4.4) and (4.10). As mentioned before, we are required to evaluate 1024 bilinear forms at each optimization step. There are two systems associated with the bilinear form $\mathbf{c}_k \mathbf{A}^{-1} \mathbf{b}_j$: the main system $A\mathbf{x} = \mathbf{b}_j$ and the dual system $\mathbf{A}^T\mathbf{y} = \mathbf{c}_k$. We can evaluate each bilinear forms using (r)BiCG-bf. This solver also computes approximate solutions $\mathbf{x}_{(j)}$ and $\mathbf{y}_{(k)}$. Each main and each dual system appears 32 times. The first method might be more efficient as it requires only 64, not 1024 solves. However, because in the second method the systems repeat, we can save approximate solutions to these systems, and use them as initial guesses when the same system arises again. Hence, using the second method we obtain better and better initial guesses with every solve, hence, a couple iterations of

(r)BiCG-bf might yield a more accurate estimate for the bilinear form than the inner product method (the first approach). The issue changes when we switch to a stochastic approach, because the number of systems reduces (in our case, $j = 1, \ldots, 10$ and $k = 1, \ldots, 10$), and the systems repeat fewer times.

Test 1: rBiCG-bf(18,15) vs. GMRES(40)

In the first test, we compare the performance of rBiCG-bf and GMRES(40). The size of the recycle spaces in rBiCG-bf is 15 and the spaces are updated every 18 iterations (we refer to this solver as rBiCG-bf(18,15)). For both methods, we test various tolerance parameters. While testing these two approaches, our goal is to find the lowest tolerance that leads to convergence of TREGS (in our case, relative tolerance for TREGS is 10^{-9} and the maximum number of function evaluations is 200). We also compare the number of matrix-vector products for the best choice of tolerance parameter(s) for each method. As a measure of comparison, we use relative residual norms and average error in bilinear form approximation. We also collect data on the number of preconditioned matrix-vector products (one iteration of rBiCG-bf contains two preconditioned matrix-vector products and one iteration of GMRES contains one matrix vector product). The number of matrix-vector products corresponds to the amount of computations we perform.

Next, we discuss the details of the testing setup.

rBiCG-bf(15,18). We use the randomized approach described in Chapter 2. The number of sources and the number of detectors are both reduced to 10. Hence, we need to evaluate 100 bilinear forms in each optimization step.

To further speed up the computations, we save the solutions to the main and the dual systems. For example, while evaluating the bilinear form $\mathbf{c}_k\mathbf{A}^{-1}\mathbf{b}_j$, we also compute approximate solutions $\mathbf{x}_{(j)}$ and $\mathbf{y}_{(k)}$; we save $\mathbf{x}_{(j)}$ and $\mathbf{y}_{(k)}$, and use them as initial guesses when $\mathbf{Ax} = \mathbf{b}_j$ or $\mathbf{A}^T\mathbf{y} = \mathbf{c}_k$ needs to be solved again. What is more, because the matrix changes slowly, we also keep these approximate solutions from one optimization step to the next. For each bilinear form, before we start rBiCG-bf, we check whether either the solution to the main system, or the solution to the dual system, has high accuracy. If this is the case, we estimate the bilinear form with an appropriate inner product.

We use the convergence criteria (3.7), i.e., rBiCG-bf converges if $\|\mathbf{r}\| < \varepsilon\|\mathbf{b}\|$ and $\|\mathbf{s}\| < \varepsilon\|\mathbf{c}\|$, where ε is a relative tolerance parameter. This criteria is different than for the topology optimization, where there is only one system. We test several choices for ε, ranging from 10^{-5} to 10^{-9}. In case, when we need an estimate for the Jacobian, we need more accurate estimates for $\mathbf{x}_{(j)}$ and $\mathbf{y}_{(k)}$ than in case, when we do not need to estimate Jacobian. Hence, we use ε_1 if we do not need Jacobian and ε_2 if we do; we always assume $\varepsilon_1 \geq \varepsilon_2$. To accelerate the convergence of rBiCG-bf, we use ILUTP preconditioner as a split preconditioner. We compute the preconditioner once per optimization step.

There are multiple possible orders of evaluating $\mathbf{c}_k\mathbf{A}^{-1}\mathbf{b}_j$ for $j = 1, \ldots, 10$ and $k = 1, \ldots 10$. For instance, we can fix k, and vary j, or fix j and vary k, see Figure 4.7. In this test, the bilinear

forms are computed in the order shown in Figure 4.9 (Pattern 1).

GMRES(40). We use the same randomized approach as for rBiCG-bf (10 simultaneous sources and 10 simultaneous detectors). We restart GMRES every 40 iterations. We assume GMRES converged when $\|\mathbf{r}\| < \varepsilon\|\mathbf{b}\|$. We vary the choice for ε. To accelerate the convergence, we use the ILUTP split preconditioner.

For GMRES(40), we also keep the solutions $\mathbf{x}_{(j)}$ and $\mathbf{y}_{(k)}$. Unlike for rBiCG-bf, the systems do not repeat within one optimization step; however, the solutions are used as initial guesses in appropriate systems for the consecutive optimization steps. Moreover, we always solve the adjoint systems, and solve the main systems only when the approximation to the Jacobian is needed. Our extensive testing shows that solving the main systems and solving the adjoint systems only in cases when the approximation to the Jacobian is required is less advantageous (it results in more GMRES iterations on average).

Results of the comparison.

We show the result of the comparison in Table 4.2. The relative tolerance settings for the cases when the Jacobian is needed and when it is not needed are shown in the second and third column of Table 4.2, respectively.

rBiCG-bf

Tolerance (with Jacobian)	10^{-7}	10^{-7}	10^{-7}	10^{-6}[1], [2]	10^{-5} [1], [2]
Tolerance (no Jacobian)	10^{-7}	10^{-6}	10^{-5}	10^{-6}	10^{-5}
# opt. steps (w Jac)	28(8)	31(10)	28(9)	200(16)	200(23)
$\|\mathbf{R}\|^2$ at the end	3.19e-09	2.44e-09	3.12e-09	4.17e-09	5.57e-09
Total # matvecs	4284	4344	3106	11600	8480
Matvecs per opt. step	153.0	151.2	110.9	58.9	42.4
Av. error in BF	3.97e-05	1.53e-04	1.80e-03	1.94e-04	1.94e-03
Av. error norm in $\mathbf{Ax} = \mathbf{b}_j$	4.38e-08	7.15e-08	4.39e-06	6.60e-07	8.40e-06
Av. error norm in $\mathbf{A}^T\mathbf{y} = \mathbf{c}_k$	9.40e-08	5.42e-07	8.88e-04	9.23e-06	1.63e-06

GMRES(40)

Tolerance (with Jacobian)	10^{-10}	10^{-10}	10^{-9}	10^{-8}	10^{-7} [2]
Tolerance (no Jacobian)	10^{-10}	10^{-9}	10^{-9}	10^{-8}	10^{-7}
# opt. steps (w Jac)	35(12)	39(13)	69(24)	45(16)	200(13)
$\|\mathbf{R}\|^2$ at the end	2.50e-09	3.12e-09	3.11e-09	3.03e-09	2.66e-08
Total # matvecs	11851	13320	19492	10595	7533
Matvecs per opt. step	338.6	341.5	282.5	235.4	37.3
Av. error in BF	4.36e-05	2.77e-04	4.58e-04	3.49e-02	1.89e-02
Av. error norm in $\mathbf{Ax} = \mathbf{b}_j$	4.53e-04	3.32e-04	1.41e-04	2.88e-04	7.59e-05
Av. error norm in $\mathbf{A}^T\mathbf{y} = \mathbf{c}_k$	4.75e-09	2.88e-08	3.85e-08	3.69e-07	3.36e-06

Table 4.2: Results of the comparison of different tolerance settings for evaluating the objective function $\|\mathbf{R}\|^2$ in DOT. The errors in the three bottom rows of each table are relative and averaged out over multiple optimization steps. The presented values are relative errors computed using the solution obtained through direct solve. In the third row, the first number is the number of objective functions evaluations, and the number in the brackets indicates how many times we evaluated the Jacobian. BF = bilinear form.

Based on Table 4.2, we conclude that the overall best method in terms of low (total) number of objective function evaluations and low number of (preconditioned) matrix-vector products is rBiCG-bf with tolerance 10^{-7} for the cases, when we need to evaluate the Jacobian and 10^{-5} for the cases when we do not. The average error in bilinear form for rBiCG-bf is worse than expected based on the theory. In most cases rBiCG-bf is applied only to 10 bilinear forms

[1] No recycle spaces formed.
[2] Optimization problem did not converge for 200 function evaluations

($\mathbf{c}_k^T \mathbf{A}^{-1} \mathbf{b}_j$ with $j = k$). Then, the remaining 90 are computed through the inner product. If ε is the tolerance for the linear solver, the error in the bilinear forms computed using rBiCG-bf is in the order of ε^2. For bilinear forms computed through the inner product, the error in the order of ε. While taking the average, we get a number close to ε, not ε^2.

The GMRES(40)-based optimization process in best case (tolerance 10^{-8}) requires almost 4 times as much preconditioned matrix-vector products. However, in our current MATLAB implementation GMRES tends to be faster in terms of runtimes.

In Figures 4.2, 4.3, 4.4, 4.5, and 4.6 we show visual results (i.e., reconstruction of the shape). The plots on the left side of the figures show decreasing number of iterations of rBiCG-bf/GMRES(40) per optimization step. This demonstrates that saving the best solutions for repeating the main and dual systems (and using recycling in case of rBiCG-bf) has a positive effect on reducing the amount of computations in consecutive optimization steps/objective function evaluations.

Table 4.2 shows that we can use lower tolerance for rBiCG-bf than for GMRES(40), and TREGS would converge. We point out that we obtain less accurate estimates for the bilinear form using GMRES(40), even if we solve the systems to a higher accuracy than for rBiCG-bf. Using GMRES(40) also results in more optimization steps. However, for this problem, the difference between performance of a standard method (GMRES(40)) and performance of a new method (rBiCG-bf) is not as dramatic as it is for the topology optimization.

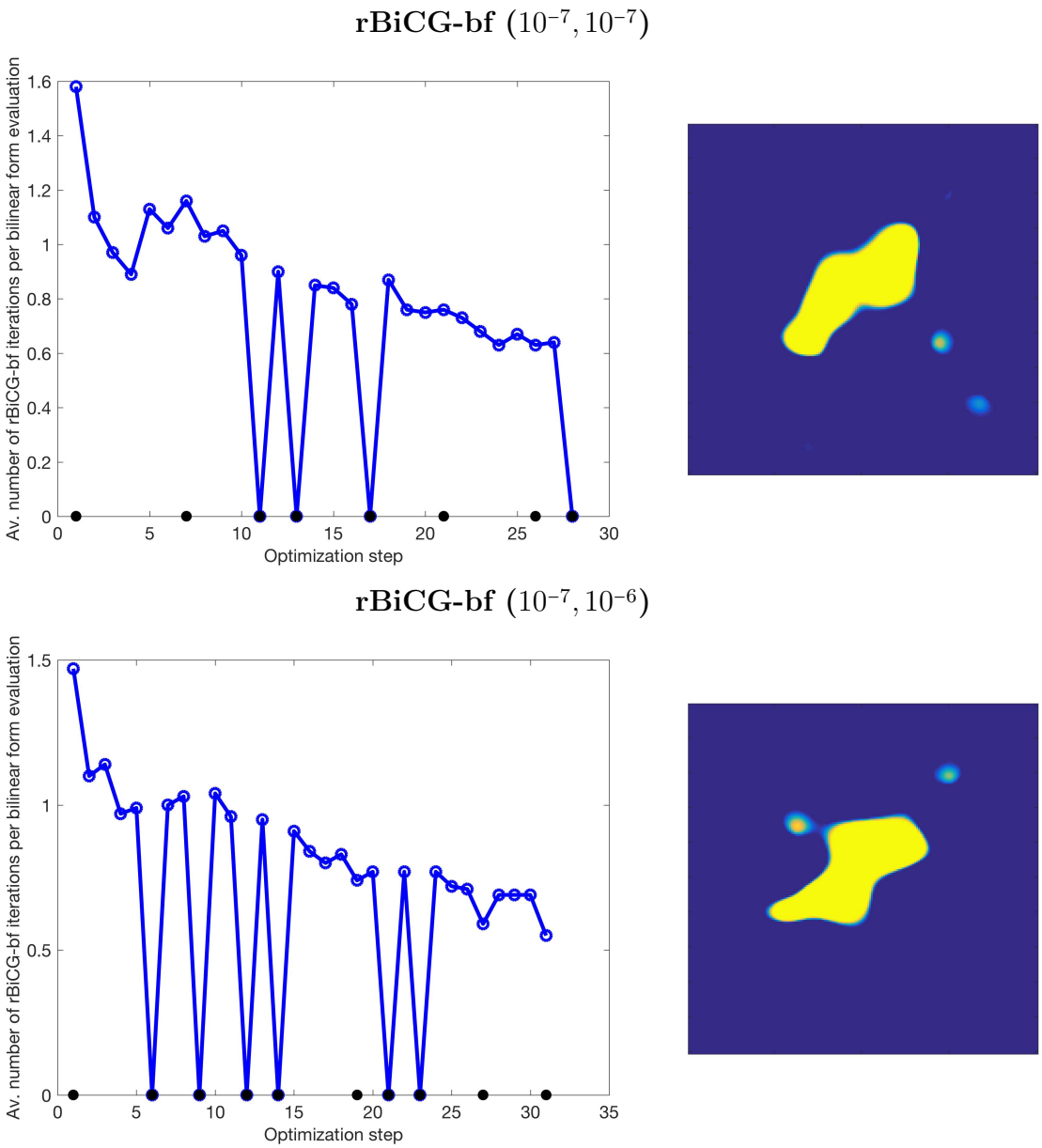
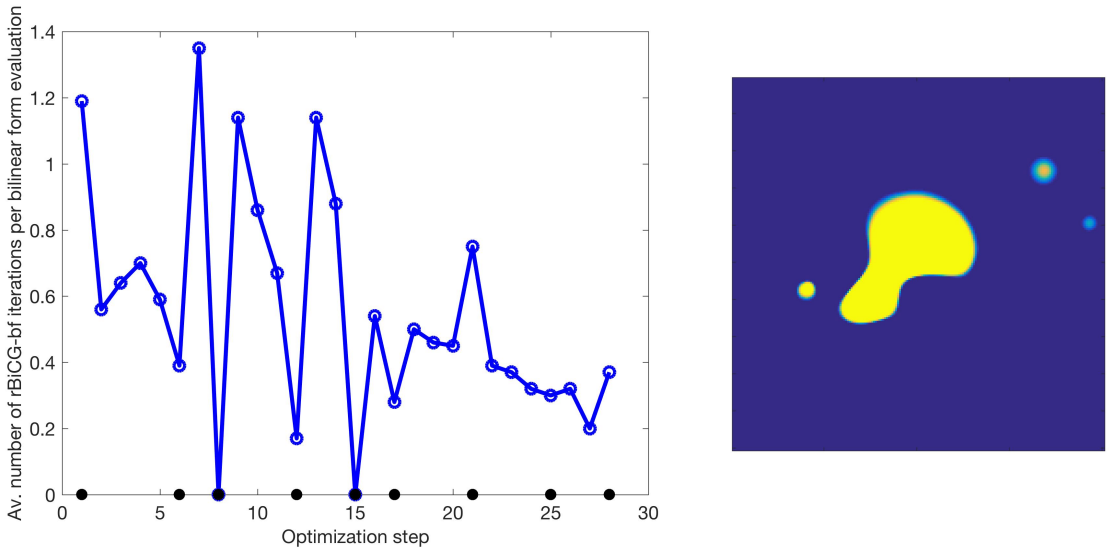


Figure 4.2: Left: number of iterations per linear system (function evaluation) in each optimization step; black dots on the bottom point out the optimization step where the Jacobian is computed. Right: reconstructed shape (amoeba). The first number in the titles is the (relative) tolerance used for the linear solver for cases, when an approximation to the Jacobian is needed and the second number is the (relative) tolerance for cases, when the approximation to the Jacobian is not needed.

rBiCG-bf $(10^{-7}, 10^{-5})$



rBiCG-bf $(10^{-6}, 10^{-6})$

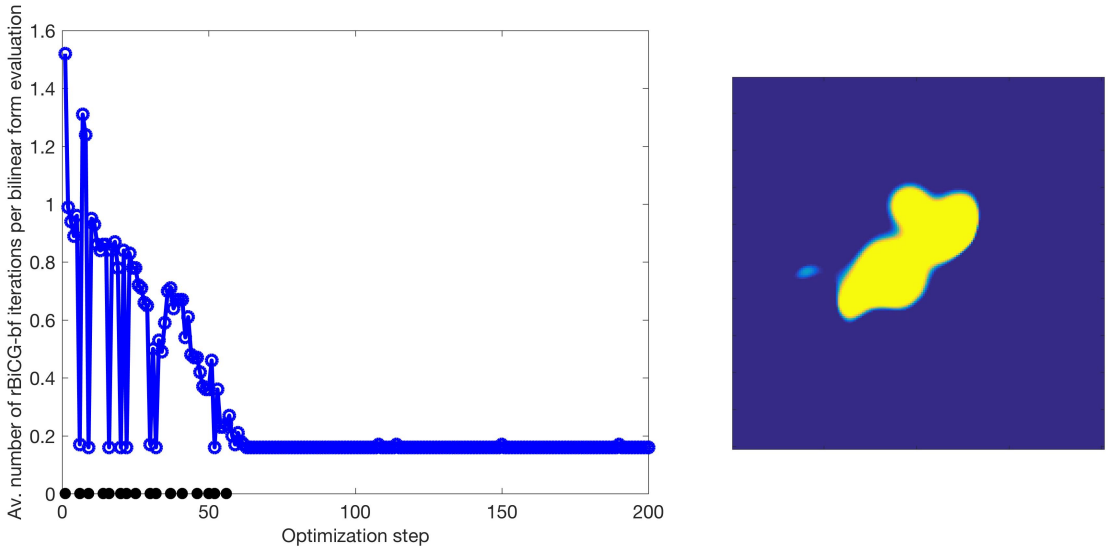


Figure 4.3: Left: number of iterations per linear system (function evaluation) in each optimization step; black dots on the bottom point out the optimization step where the Jacobian is computed. Right: reconstructed shape (amoeba). The first number in the titles is the (relative) tolerance used for the linear solver for cases, when an approximation to the Jacobian is needed and the second number is the (relative) tolerance for cases, when the approximation to the Jacobian is not needed.

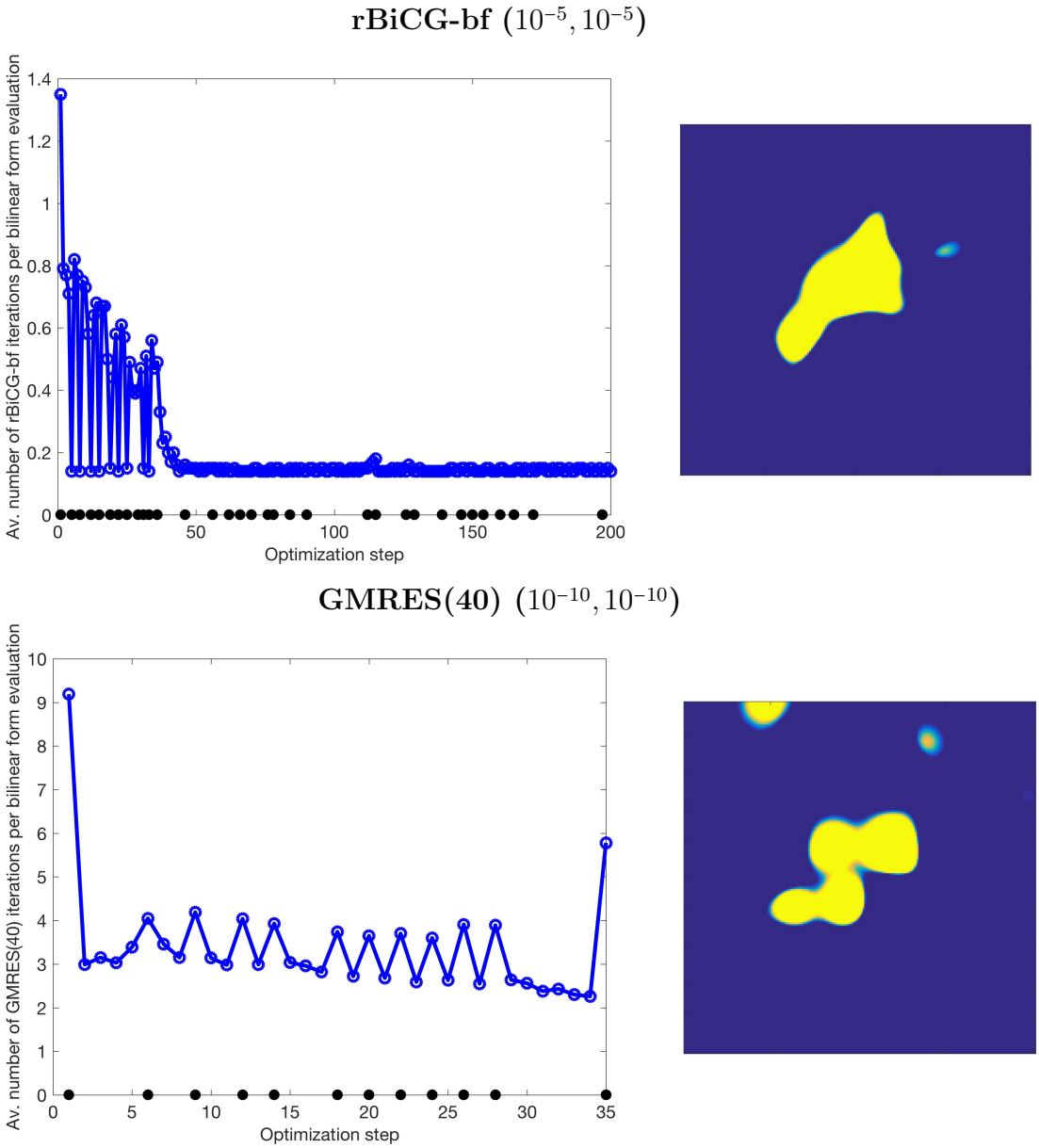
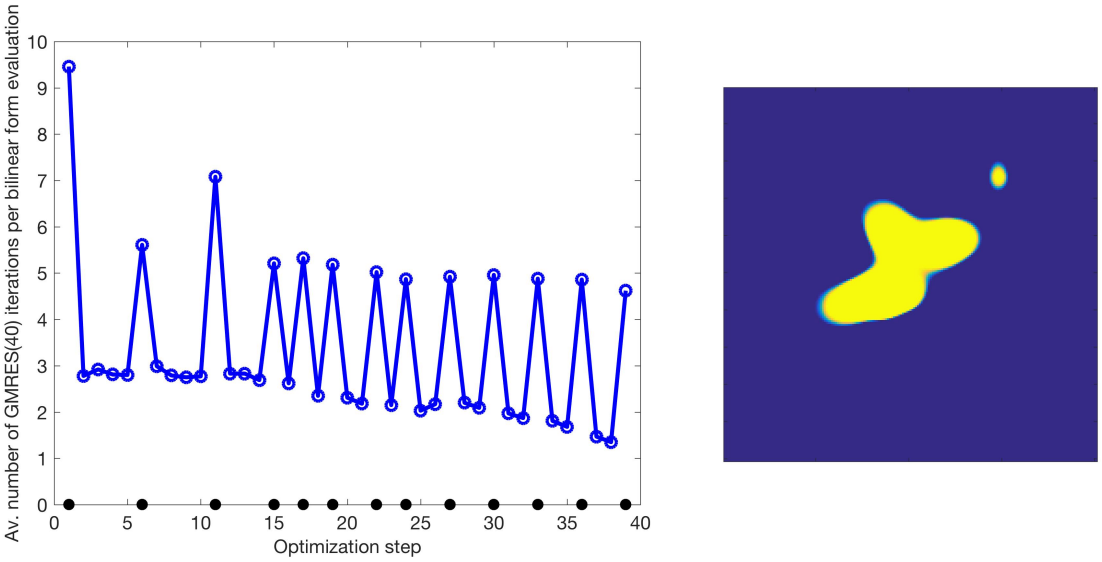


Figure 4.4: Left: number of iterations per linear system (function evaluation) in each optimization step; black dots on the bottom point out the optimization step where the Jacobian is computed. Right: reconstructed shape (amoeba). The first number in the titles is the (relative) tolerance used for the linear solver for cases, when an approximation to the Jacobian is needed and the second number is the (relative) tolerance for cases, when the approximation to the Jacobian is not needed.

GMRES(40) $(10^{-10}, 10^{-9})$



GMRES(40) $(10^{-9}, 10^{-9})$

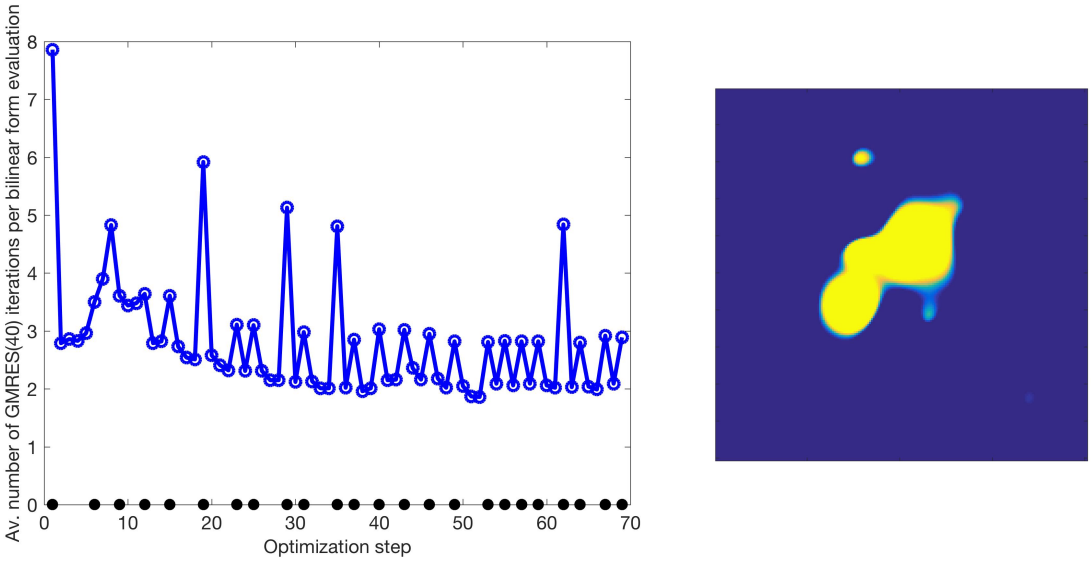


Figure 4.5: Left: number of iterations per linear system (function evaluation) in each optimization step; black dots on the bottom point out the optimization step where Jacobian is computed. Right: reconstructed shape (amoeba). The first number in the titles is the (relative) tolerance used for the linear solver for cases, when an approximation to the Jacobian is needed and the second number is the (relative) tolerance for cases, when the approximation to the Jacobian is not needed.

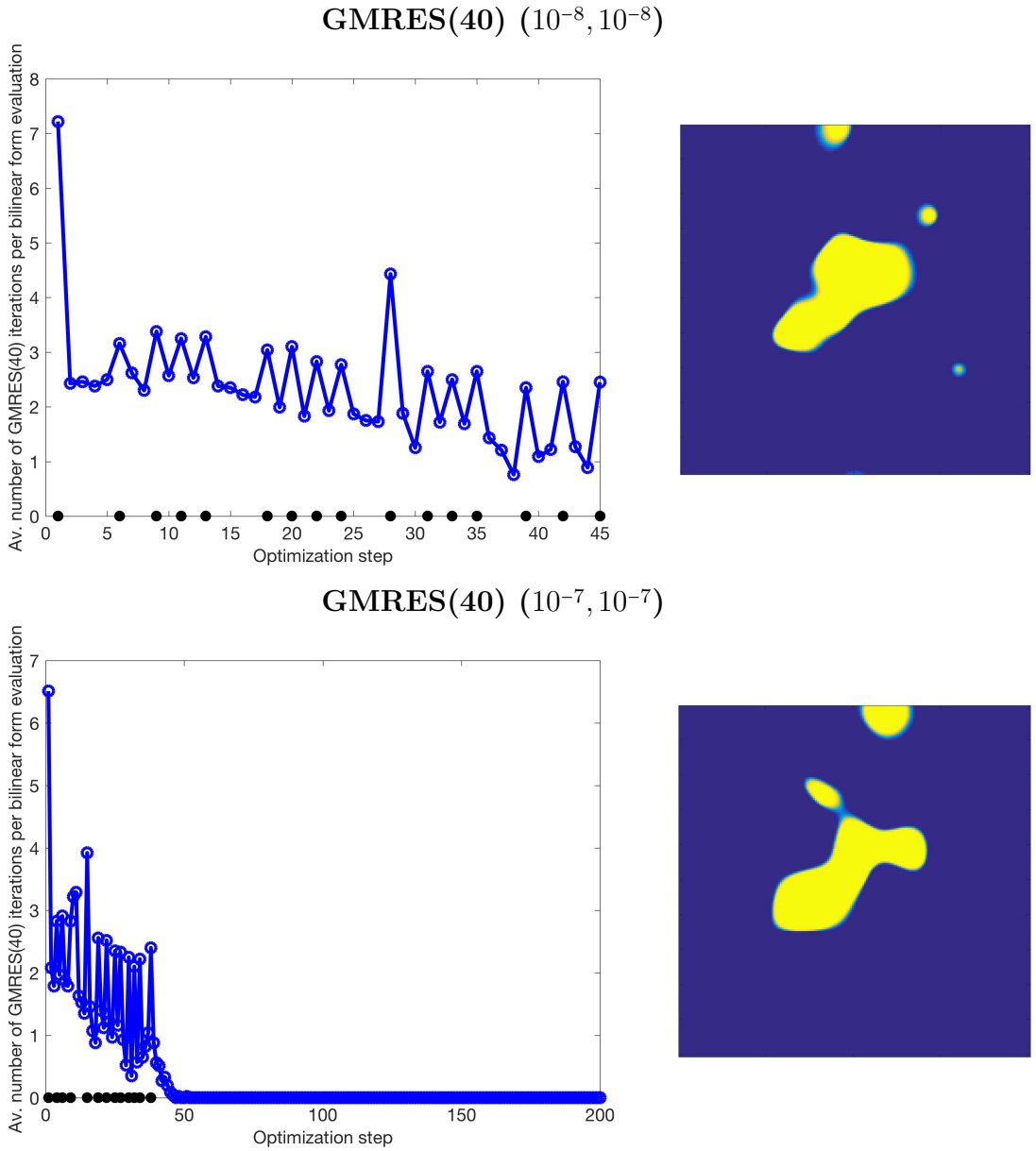


Figure 4.6: Left: number of iterations per linear system (function evaluation) in each optimization step; black dots on the bottom point out the optimization step where the Jacobian is computed. Right: reconstructed shape (amoeba). The first number in the titles is the (relative) tolerance used for the linear solver for cases, when an approximation to the Jacobian is needed and the second number is the (relative) tolerance for cases, when the approximation to the Jacobian is not needed.

Test 2: order of evaluation

The DOT problem is a perfect problem for applying Krylov subspace recycling. We have

many systems per optimization step, the matrix does not change within one optimization step, and changes only slightly from one step to another. Because we need to evaluate many bilinear forms, and the main and dual systems associated with the bilinear forms repeat, next we analyze what is the best order of evaluating these bilinear form and solving related main and dual systems.

Figure 4.7 represents all the bilinear forms that we need to compute as a table. To answer the main question, we test multiple orders.

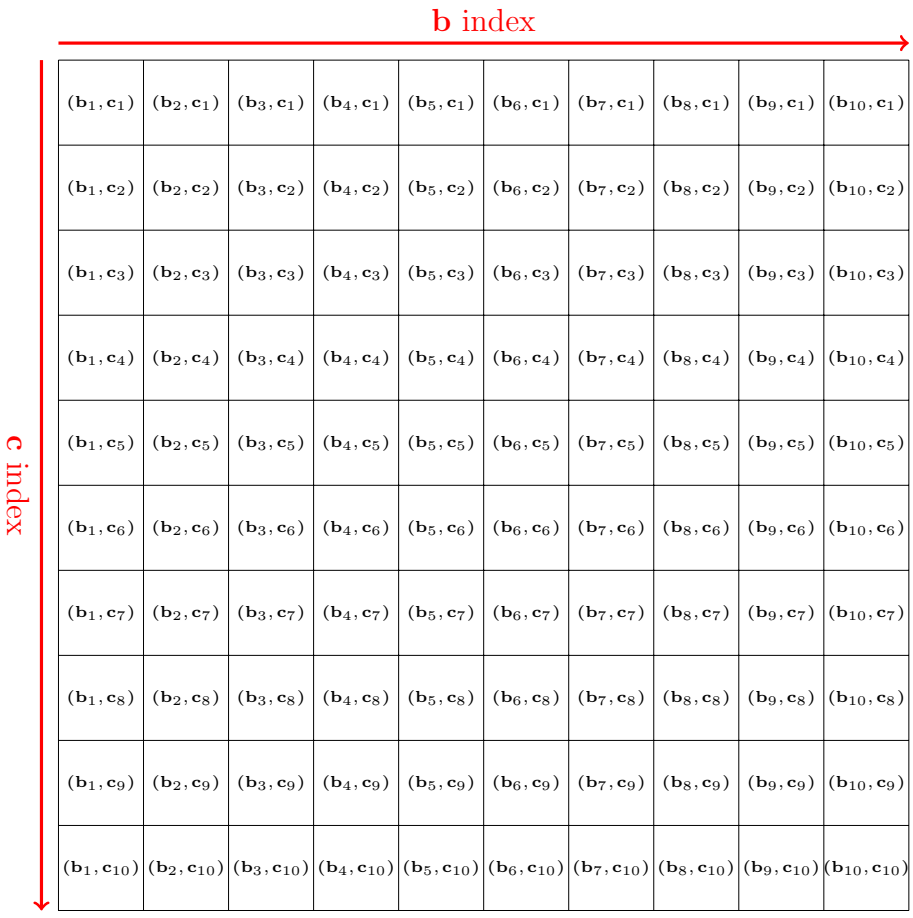


Figure 4.7: Graphical representation of the bilinear forms $\mathbf{c}_j\mathbf{A}^{-1}\mathbf{b}_k$, $j,k = 1,\ldots,10$

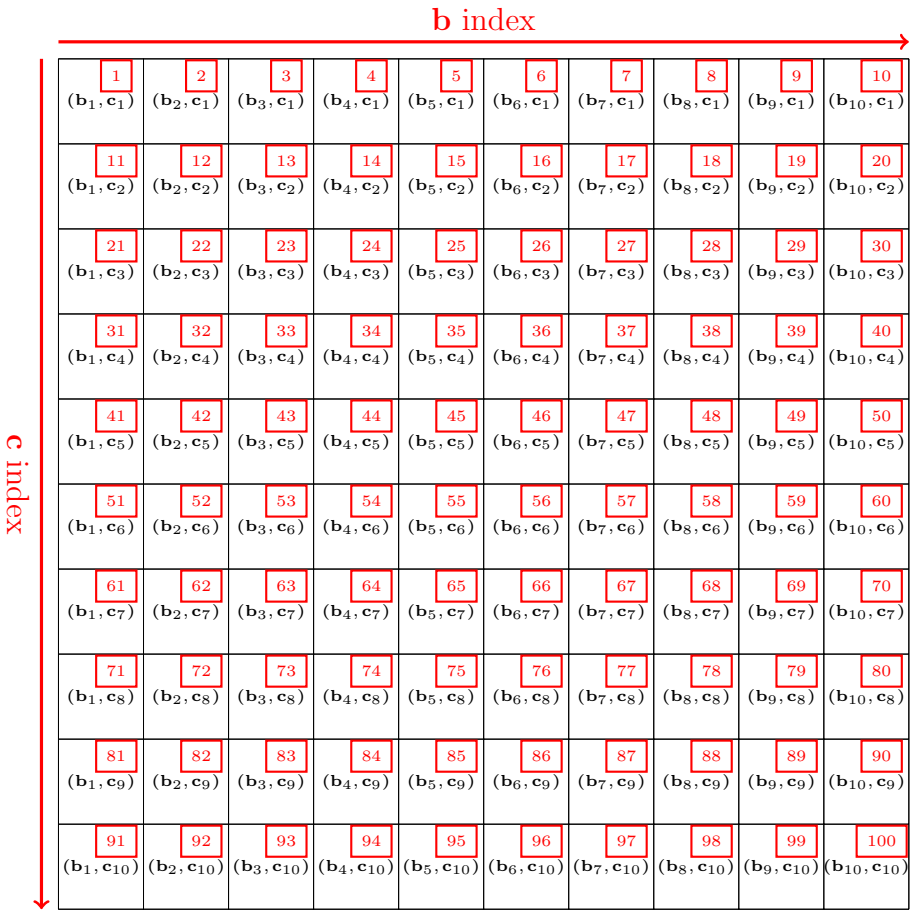


Figure 4.8: Graphical representation of an order of computation. The numbers in the red boxes indicate the order in which the bilinear forms are evaluated. We start with $\mathbf{b}_1\mathbf{A}^{-1}\mathbf{c}_1$, we evaluate $\mathbf{b}_2\mathbf{A}^{-1}\mathbf{c}_1$ next, then $\mathbf{b}_3\mathbf{A}^{-1}\mathbf{c}_1$, etc.

Figure 4.8 ties the bilinear forms shown in Figure 4.7 with the idea of the order of computations. We can use multiple different orders, for example, instead of processing the bilinear forms in the order shown in Figure 4.8, we can evaluate the bilinear form $\mathbf{b}_1\mathbf{A}^{-1}\mathbf{c}_1$ first, then $\mathbf{b}_1\mathbf{A}^{-1}\mathbf{c}_2$, then $\mathbf{b}_1\mathbf{A}^{-1}\mathbf{c}_3$, etc. Figure 4.9 shows the patterns we test. The order of computations shown in Figure 4.8 corresponds to Pattern 3 in Figure 4.9.

In the test, we set the tolerance to $\varepsilon = 10^{-7}$ for cases when the Jacobian is needed and we set the tolerance to $\varepsilon = 10^{-5}$ for cases when the Jacobian is not needed. The tolerance parameters were chosen based on the results we obtained in Test 1 in this section.

The average number of iterations per 100 bilinear forms is 79.85 with Pattern 1, 96 with Pattern 2, 114.8 with Pattern 3, and with 96.85 Pattern 4. It turns out that by choosing the *right* order we might reduce the number of iterations by 30% while keeping other parameters, such as size of recycle space, the same.

Pattern 1

1	11	12	13	14	15	16	17	18	19
20	2	21	22	23	24	25	26	27	28
29	30	3	31	32	33	34	35	36	37
38	39	40	4	41	42	43	44	45	46
47	48	49	50	5	51	52	53	54	55
56	57	58	59	60	6	61	62	63	64
65	66	67	68	69	70	7	71	72	73
74	75	76	77	78	79	80	8	81	82
83	84	85	86	87	88	89	90	9	91
92	93	94	95	96	97	98	99	100	10

Pattern 2

1	20	29	38	47	56	65	74	83	92
11	2	30	39	48	57	66	75	84	93
12	21	3	40	49	58	67	76	85	94
13	22	31	4	50	59	68	77	86	95
14	23	32	41	5	60	69	78	87	96
15	24	33	42	51	6	70	79	88	97
16	25	34	43	52	61	7	80	89	98
17	26	35	44	53	62	71	8	90	99
18	27	36	45	54	63	72	81	9	100
19	28	37	46	55	64	73	82	91	10

Pattern 3

1	2	3	4	5	6	7	8	9	10
11	12	13	14	15	16	17	18	19	20
21	22	23	24	25	26	27	28	29	30
31	32	33	34	35	36	37	38	39	40
41	42	43	44	45	46	47	48	49	50
51	52	53	54	55	56	57	58	59	60
61	62	63	64	65	66	67	68	69	70
71	72	73	74	75	76	77	78	79	80
81	82	83	84	85	86	87	88	89	90
91	92	93	94	95	96	97	98	99	100

Pattern 4

1	11	21	31	41	51	61	71	81	91
2	12	22	32	42	52	62	72	82	92
3	13	23	33	43	53	63	73	83	93
4	14	24	34	44	54	64	74	84	94
5	15	25	35	45	55	65	75	85	95
6	16	26	36	46	56	66	76	86	96
7	17	27	37	47	57	67	77	87	97
8	18	28	38	48	58	68	78	88	98
9	19	29	39	49	59	69	79	89	99
10	20	30	40	50	60	70	80	90	100

Figure 4.9: Four patterns (orders) for computing bilinear forms $\mathbf{c}_k\mathbf{A}^{-1}\mathbf{b}_j$ for $j = 1, \ldots, 10$ and $k = 1, \ldots 10$

Pattern 2 results in bilinear forms with smallest average error $10^{-5.23}$, whereas Pattern 4 results in bilinear form with largest average error 10^{-4}.

The relative error in the approximate solutions to the main systems $\mathbf{Ax} = \mathbf{b}_{(j)}$, $j = 1, \ldots, 10$ is similar for all patterns, except Pattern 4. However, for the solutions to the dual systems, the error for Pattern 3 is the the worst and for Pattern 4 the error is almost two orders of magnitude worse than for Patterns 1 and 2. These results are shown in Figure 4.10.

We conclude that in our case, we should either use Pattern 1 or Pattern 2 and we should avoid Patterns 3 and 4.

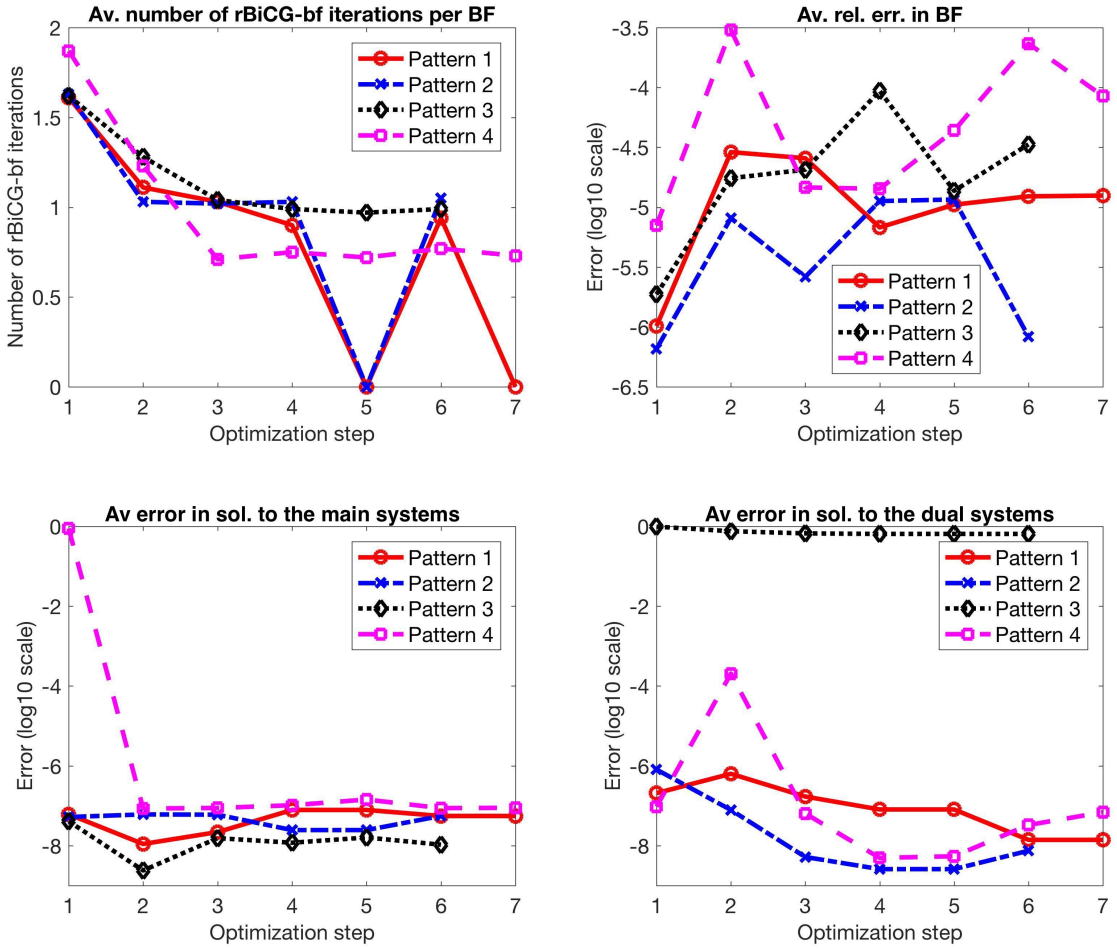


Figure 4.10: rBiCG-bf and the influence of order of recycling on errors and the number of iterations.

4.4.3 Error estimation and grid adaptation

In Chapter 2, we discuss functional-based error estimation and grid adaptation in CFD. In this problem, the bilinear form arises in the formula (2.31), which describes error in the functional of interest. In the current subsection, we test the rBiCG-bf approach for estimating error formulas of a type (2.31) in a Quasi-1D nozzle problem, which is a one-dimensional approximation to the compressible flow through a nozzle with varying cross-sectional area.

Quasi-1D nozzle: problem description

The quasi-1D nozzle problem has a known analytic solution, which makes it an ideal test scenario. The nozzle is a tube characterized by varying cross-sectional area. The flow through

the tube is very rapid with negligible frictional loses. In the initial section of the nozzle, the diameter of the nozzle contracts causing the flow to accelerate. In the subsequent section of the nozzle, the diameter increases, which allows the flow to expand subsonically or supersonically depending upon the pressure at the nozzle exit. Even though we refer to the cross-section using terms such as "diamete" or "area", it is treated like a one-dimensional quantity in the problem setup.

The geometry of the nozzle is set up using a Gaussian area distribution [54]. We examine fully supersonic flow through the diverging section. We fix the stagnation temperature to 600K and the stagnation pressure to 300kPa at the inflow of the nozzle. The Mach number is extrapolated from the interior to set the inflow state at the boundary face. The outflow boundary conditions depend upon the local character of the flow in the diverging section. In our case, all variables in the interior are extrapolated to the face to set the outflow flux

The governing equations for this problem are the Euler equations (conservation of mass, momentum, and energy) in weak form. We use $(\cdot)^+$ to denote dimensional quantity (in our tests, we convert the equations to dimensionless form to avoid numerical instability). For control volume Ω^+, this gives

$$\frac{\partial}{\partial t^+}\int_{\Omega^+}\mathbf{Q}^+\,d\Omega^+ + \oint_{\partial\Omega^+}\mathbf{F}^+\,dS^+ = \int_{\Omega^+}\mathbf{S}^+\,d\Omega^+ \tag{4.21}$$

where $\mathbf{Q}^+$ is the vector of conserved variables, $\mathbf{F}^+$ is the vector of inviscid fluxes, and $\mathbf{S}^+$ is a source term. These vectors are defined as

$$\mathbf{Q}^+ = \begin{bmatrix}\rho^+\\ \rho^+u^+\\ \rho^+e_t^+\end{bmatrix},\quad \mathbf{F}^+ = \begin{bmatrix}\rho^+u^+\\ \rho u^{+2}+p^+\\ \rho^+u^+h_t^+\end{bmatrix},\quad \mathbf{S}^+ = \begin{bmatrix}0\\ p^+\frac{dA^+}{dx^+}\\ 0\end{bmatrix},$$

where u^+ is the fluid velocity, ρ^+ is the fluid density, p^+ is the static pressure, e_t^+ is the total energy, h_t^+ is the total enthalpy, and $\frac{dA^+}{dx^+}$ is the change in the cross-section area in respect to change in nozzle length.

In addition, we close the system by using the equation of state for a perfect gas i.e.,

$$e_t^+ = \frac{p^+}{\rho^+(\gamma-1)} + \frac{u^{+2}}{2},$$

$$h_t^+ = \frac{\gamma p^+}{\rho^+(\gamma-1)} + \frac{u^{+2}}{2},$$

where γ is the ratio of specific heats for a perfect gas – for air it is $\gamma = 1.4$.

We discretize (4.21) using a second order, cell-centered finite-volume stencil [93]. The discretized version of (4.21) in the non-dimensional form is

$$\frac{A_j\Delta x_j}{\Delta t}\left[\mathbf{Q}_j^{n+1}-\mathbf{Q}_j^n\right] + \mathbf{R}_j^{n+1} = 0, \tag{4.22}$$

where the non-dimensional steady-state residual $\mathbf{R}_j^{n+1}$ is given by

$$\mathbf{R}_j^{n+1} = \mathbf{F}_{k+\frac{1}{2}}^{n+1} A_{i+\frac{1}{2}} - \mathbf{F}_{k-\frac{1}{2}}^{n+1} A_{i-\frac{1}{2}} - \Delta x_j \mathbf{S}_j^{n+1}. \tag{4.23}$$

We solve for a steady state using pseudo time stepping [93]. The resulting matrix equation is

$$\left[\frac{\Omega}{\Delta t} I + \frac{\partial \mathbf{R}}{\partial \mathbf{Q}}\right]^{(n)} \Delta \mathbf{Q}^n = -\mathbf{R}^n, \tag{4.24}$$

where $\frac{\Omega}{\Delta t} I$ is a matrix with the volume and time step contribution from each cell placed along the main diagonal, $\frac{\partial \mathbf{R}}{\partial \mathbf{Q}}$ is the residual Jacobian matrix, $\Delta \mathbf{Q}^n$ is a forward difference of the conserved variable vector given by $\Delta \mathbf{Q}^n = \mathbf{Q}^{n+1} - \mathbf{Q}^n$, and $\mathbf{R}^n$ is the steady-state residual evaluated at time step n.

The associated adjoint problem is given as [93]

$$\left[\frac{\Omega}{\Delta t}\frac{\partial \mathbf{Q}}{\partial \mathbf{q}} + \frac{\partial \mathbf{R}}{\partial \mathbf{q}}\right]^T \Delta \lambda^n = -\mathbf{R}_{adj}^n, \tag{4.25}$$

where $\mathbf{q} = [\rho, u, p]^T$ is the vector of primitive variables. In (4.25), $\mathbf{R}_{adj}^n$ is the adjoint residual defined as

$$\mathbf{R}_{adj}^n = -\left[\frac{\partial J_h}{\partial \mathbf{q}}\right]^T + \left[\frac{\partial \mathbf{R}}{\partial \mathbf{q}}\right]^T \lambda^n \tag{4.26}$$

and $\Delta \lambda^n$ is a forward difference of the adjoint variables, given by $\Delta \lambda^n = \lambda^{n+1} - \lambda^n$.

We solve (4.25) using time marching; we solve for steady state.

The ETEs are solved in a similar manner using

$$\left[\frac{\Omega}{\Delta t}\frac{\partial \mathbf{Q}}{\partial \mathbf{q}} + \frac{\partial \mathbf{R}}{\partial \mathbf{q}}\right] \Delta \varepsilon_h^n = -\mathbf{R}_{ETE}^n, \tag{4.27}$$

$\mathbf{R}_{ETE}^n$ is the ETE residual given by

$$\mathbf{R}_{ETE}^n = \tau_h(\tilde{u}) + \frac{\partial \mathbf{R}}{\partial \mathbf{q}} \varepsilon_h^n, \tag{4.28}$$

where $\frac{\partial \mathbf{Q}}{\partial \mathbf{q}}$ is a conversion matrix, and $\Delta \varepsilon_h^n$ is a forward difference of the discretization error given by $\Delta \varepsilon_h^n = \varepsilon_h^{n+1} - \varepsilon_h^n$.

We choose five functionals of interest,

	Continuous Functional	Discrete Functional
Integral of Pressure :	$\mathbf{J}(\tilde{u}) = \int_{-1}^{1} p\, dx$	$\mathbf{J}_h(u_h) = \sum_{i=1}^{N_{cells}} p_i \Delta x_i$
Integral of Entropy :	$\mathbf{J}(\tilde{u}) = \int_{-1}^{1} \log\left(\frac{p}{\rho^\gamma}\right) dx$	$\mathbf{J}_h(u_h) = \sum_{i=1}^{N_{cells}} \log\left(\frac{p_i}{\rho_i^\gamma}\right) \Delta x_i$
Total Mass :	$\mathbf{J}(\tilde{u}) = \int_{-1}^{1} \rho A\, dx$	$\mathbf{J}_h(u_h) = \sum_{i=1}^{N_{cells}} \rho_i A_i \Delta x_i$
Integral of Mass Flow :	$\mathbf{J}(\tilde{u}) = \int_{-1}^{1} \rho u A\, dx$	$\mathbf{J}_h(u_h) = \sum_{i=1}^{N_{cells}} \rho_i u_i A_i \Delta x_i$
Static Thrust :	$\mathbf{J}(\tilde{u}) = \rho_e u_e^2 A_e + (p_e - p_a) A_e$	$\mathbf{J}_h(u_h) = \rho_e u_e^2 A_e + (p_e - p_a) A_e$

Here $p_a = 5.5kPa$ is the ambient pressure.

Quasi-1D nozzle: grid adaptation

There are many methods of grid adaptation. We adapt our one-dimensional grid by $r-$adaptation. In this case, the number of grid nodes remains the same but we shift the nodes as needed. This strategy has computational advantages, as it does not involve much data movement. We shift the nodes to make the product of cell size Δx and a weight function approximately equal in all cells across the domain. This strategy is illustrated in Figure 4.11. The details of the weighting strategy are given in [93].

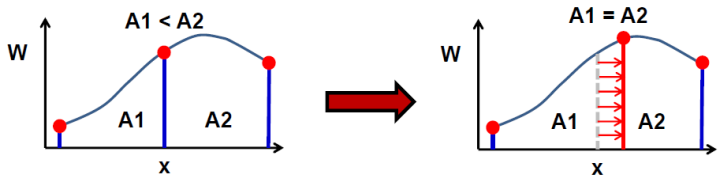


Figure 4.11: Grid adaptation for Quasi-1D nozzle problem using node shifting. Figure comes from [93]

Quasi-1D nozzle: results

We perform two tests. In the first test, we solve the ETE and the adjoint problem (used for adaptation) together and compute the functional correction as a bilinear form using BiCG-bf with SAI as a preconditioner. We vary the tolerances parameter, $\varepsilon = 10^{-1}, 10^{-2}, 10^{-3}$, and we use the standard convergence criteria (3.6) for BiCG-bf. We apply BiCG-bf to six uniformly refined grids. Since the problem is one-dimensional, ILU is very close to the true

LU factorization, and does show the true benefit of BiCG-bf approach, thus, to provide a more realistic example, we use SAI. We choose the integral of pressure as the functional of interest in the ETE. The results are shown in Table 4.3.

rel. tol. for rBiCG-bf ε = **1e – 1**

$\mathbf{N_{cells}}$	**rel. res. norm ETE**	**rel. res. norm AP**	**rel. err. BF**
32	9.93e-02	3.26e-01	3.04e+00
64	1.58e-02	3.97e-01	3.33e-01
128	4.15e-03	2.64e-01	2.04e+00
256	3.41e-04	3.60e-02	5.71e-07
512	1.49e-04	3.31e-02	1.89e-05

rel. tol. for rBiCG-bf ε = **1e – 2**

$\mathbf{N_{cells}}$	**rel. res. norm ETE**	**rel. res. norm AP**	**rel. err. BF**
32	4.64e-02	1.01e-01	1.21e-01
64	1.12e-03	5.79e-02	1.63e-03
128	8.31e-04	4.96e-02	9.72e-04
256	3.30e-04	1.42e-02	2.53e-07
512	3.82e-05	4.47e-03	1.89e-05

rel. tol. for rBiCG-bf ε = **1e – 3**

$\mathbf{N_{cells}}$	**rel. res. norm ETE**	**rel. res. norm AP**	**rel. err. BF**
32	2.38e-03	9.51e-03	5.04e-05
64	6.97e-04	1.33e-02	9.70e-06
128	3.61e-04	1.49e-02	1.91e-05
256	4.35e-05	3.88e-03	2.50e-07
512	1.41e-05	3.84e-03	1.89e-05

Table 4.3: Relative residual norms in the ETE and the adjoint problem (AP), and the relative error in the functional correction (BF) computed with rBiCG-bf solved with tolerance parameter shown on the very to of each individual table.

From Table 4.3, it is clear that the residual norm for the ETE solve is small for all tolerance parameters. This is not surprising, because we solve ETE multiple times. The residual norm is much larger for the adjoint problems. However, as concluded in [93], the adaptation indicators performed very well while compared to the standard approach. We do not need the adaptation indicators to be very exact, and for performing adaptation, it is just enough if they capture the main features. These adaptation indicators change drastically when the grid is moved (the weight function is highly non-linear and we perform smoothing to maintain a stable adaptation process), so there is no reason to spend time on computing accurate adaptation indicators.

The bilinear form, used as a functional correction, differs from the true value (approximated with direct solve) by less than 0.003% for $\varepsilon = 10e-3$. We conclude that overall, the approach works well.

Since we use a standard convergence criteria in rBiCG-bf, the solver stops when $\frac{\|\mathbf{r}_k\|}{\|\mathbf{b}\|}\frac{\|\mathbf{s}_k\|}{\|\mathbf{c}\|} < \nu \leq \varepsilon$ in iteration k. Hence, the relative error in bilinear form is smaller than $\nu\frac{\|c\|\|b\|\|\mathbf{A}^{-1}\|}{\mathbf{c}^T\mathbf{A}^{-1}\mathbf{b}}$. The quantity $\frac{\|c\|\|b\|\|\mathbf{A}^{-1}\|}{\mathbf{c}^T\mathbf{A}^{-1}\mathbf{b}} > 1$, thus we might obtain relative error in bilinear form that is bigger than ε, which happens for a few grid sizes with $\varepsilon = 10e-1$ and $\varepsilon = 10e-2$.

In the second test, we consider runtime. The BiCG-bf approach produces local error estimates, adjoint variables for adaptation, and a functional correction with just one run. The traditional approach is to solve the the adjoint problems and the ETE independently using a solver such as GMRES. We perform the comparison between GMRES(45) and BiCG-bf. The comparison was done using Matlab. The results are shown in Table 4.4. The results indicate speedups in the order of 30 – 50 times.

Runtimes for GMRES(45)

$\mathbf{N_{cells}}$	Adjoint solve(s)	ETE solve(s)	Adjoint+ETE solve(s)
32	0.0877	0.0515	0.1392
64	0.3581	0.9034	1.2615[3]
128	1.5549	1.5054	3.0602[3]
256	5.6034	5.8143	11.4177[3]
512	13.7266	14.3274	28.0540[3]
1024	53.2667[4]	35.0792	88.3459[3,4]

Runtimes for BiCG-bf approach

$\mathbf{N_{cells}}$	BiCG-bf(s) ε = 1e-1	Speedup	BiCG-bf(s) ε = 1e-2	Speedup	BiCG-bf(s) ε = 1e-3	Speedup
32	0.0305	4.57	0.0390	3.57	0.0302	4.62
64	0.0376	33.57	0.0431	29.28	0.0438	28.78
32	0.0305	4.57	0.0390	3.57	0.0302	4.62
64	0.0376	33.57	0.0431	29.28	0.0438	28.78
128	0.0599	51.08	0.0830	36.85	0.0731	41.85
256	0.1479	77.21	0.1502	76.02	0.1512	75.49
512	0.5213	53.82	0.5443	51.54	0.5490	51.10
1024	1.7163	51.48	2.0403[5]	43.30	1.9616[5]	45.04

Table 4.4: Time comparison (in seconds) between traditional (GMRES(45)) and BiCG-bf approach for Quasi-1D nozzle problem with different size grids.

The rBiCG-bf approach used to approximate functional correction works well, both in terms of runtime and accuracy.

[3] Required better preconditioner for both main and adjoint solve.
[4] Did not converge.
[5] Required better preconditioner (ILU or mSAI).

Chapter 5

Hybrid method using Krylov subspace recycling

5.1 Introduction

5.1.1 Motivation

The motivation for the hybrid solver and rBiCGstab variant presented in this chapter is the pressure Poisson equation that arises in the incompressible Navier-Stokes problems solved in GenIDLEST. As discussed in Table 2, the discretized pressure Poisson equation results in a non-symmetric linear system for both the turbulent channel flow problem and the flow through porous media problem.

A typical approach is to use either GMRES(m) or BiCGStab to solve pressure the Poisson equation. The advantage of using BiCGStab are its cheap iterations. However, at the start of the simulation, when the initial guess is far from the solution, the linear systems are the hardest to solve and BiCGStab fails to converge. In this phase, a better choice is GMRES(m), because of its robustness. But even for GMRES(m), the convergence is often slow, because of the restarts. Hence, to reduce the adverse effects of restarting, we investigate the rGCROT solver (Algorithm 5) for these systems. rGCROT is a recycling solver. It recycles a selected subspace and this strategy often leads to faster convergence. In this case, the matrix in the pressure Poisson equation does not change for subsequent systems. The right hand side changes,

This chapter is the result of a collaborative effort with Dr. Amit Amritkar (former postdoc at Virginia Tech, now University of Houston), Dr. Eric de Sturler, Dr. Danesh Tafti (Virginia Tech, Mechanical Engineering), and Dr. Kapil Ahuja (IIT Indore). The collaboration resulted in [7].

though, because it is based on the local balance of the intermediate volume fluxes at cell faces surrounding each finite volume. Thus, we also use the recycling subspace from the system N to accelerate the convergence for system $N + 1$. In Section 5.3, we show that the outcome is a drastic reduction in the number of iterations compared with GMRES(m). However, the rGCROT method is expensive in terms of storage and computation time compared with BiCGStab. This motivates the development of a *hybrid* solver. This hybrid solver combines rGCROT and rBiCGStab. In the initial time steps, we solve the systems with rGCROT. Over few time steps, rGCROT builds an effective recycle space, which we recycle further in rBiCGStab solver, after switching solvers.

In addition to the hybrid method, we also derive a simplified version of rBiCGStab. The proposed new version reduces the amount of computation compared with the original method (Algorithm 7) and allows us to use only one recycle space.

5.1.2 Background

Using Krylov subspace recycling in CFD is not a new idea. A simplified version of GCROT(m,k) has been successfully applied to aerodynamic shape optimization problems [51, 50, 58] and shown to accelerate the convergence. Carpenter and collaborators [19] used enriched GMRES for steady convection-diffusion problems and for flow over a wind turbine problem. They demonstrated findings similar to [51, 50, 58]. In their application, recycling is able to eliminate stagnation in some cases. However, there exist instances when recycling does not work well. For example, Mohamed et al. [64] demonstrated that GMRES(m) performs better than GCRO-DR for both restarting and sequences of systems for a variety of aerodynamic flows.

In the literature, the authors use either GCRO-DR, (r)CGROT(m,k) or enriched GMRES. These solvers can be thought of as variants of GMRES. To our knowledge, there exists no literature that discusses recycling Krylov subspace solvers based on the Lanczos bi-orthogonalization (rBiCG or rBiCGStab) applied to CFD simulations. These solvers have been used for the linear systems arising in model reduction [2, 3, 5, 4].

5.2 Method

5.2.1 Hybrid method

The results in Section 5.3 show that the rGCROT solver is very robust and often converges in fewer iterations than BiCGStab. However, the iterations of rGCROT are expensive compared with BiCGStab, so the runtime of rGCROT might be higher.

Since the robustness of the solver is crucial mainly in the initial time steps, we use rGCROT only for these time steps; afterwards, we switch to BiCGStab or rBiCGStab. The rBiCGStab

solver is characterized by faster convergence and better robustness compared with BiCGStab. In our approach, we start the simulation by solving the pressure Poisson equation with rGCROT for a few time steps, and afterwards, we switch to rBiCGStab, recycling the space selected in rGCROT. We do not update the recycle space in rBiCGStab. In theory, one might want to switch back to rGCROT for one or more time steps if the matrix in the pressure Poisson equation changes, or if the convergence becomes poor, but we found no need for switching back in the applications presented here.

5.2.2 rBiCGStab with one recycle space

In Algorithm 7, we use two recycle spaces, range($\mathbf{U}$) and range$\left(\tilde{\mathbf{U}}\right)$. However, if we apply the hybrid approach described in the previous subsection, we only get a single recycle space, computed by rGCROT.

Let range($\mathbf{U}$) be the recycle space we would like to use in rBiCGStab. As in rGCRO, we compute $\mathbf{C_2} = \mathbf{AU}$, and we use the QR decomposition to obtain $\mathbf{C}$ such that $\mathbf{C_2} = \mathbf{CR}$ and $\mathbf{C}^T\mathbf{C} = \mathbf{I}$.

Let $\mathbf{A_C} = \left(\mathbf{I} - \mathbf{CC^T}\right)\mathbf{A}$ and let $\mathbf{r}_0 = \left(\mathbf{I} - \mathbf{CC^T}\right)\mathbf{r}_{-1}$, where $\mathbf{x}_{-1}$ is an arbitrary initial guess. Let $\mathbf{r}_{-1}$ be a residual associated with the initial guess. We run BiCGStab (Algorithm 4) with $\mathbf{A_C}$ in place of $\mathbf{A}$ and with starting residual $\mathbf{r}_0$; we add special updates for the solution.

We note that based on properties of Krylov subspace method, we have

$$\mathbf{x}_{j+1} = \mathbf{x}_j + \mathbf{z}_j \quad \Rightarrow \quad \mathbf{r}_{j+1} = \mathbf{r}_j - \mathbf{A}\mathbf{z}_j,$$

which leads to

$$\mathbf{z}_j = \mathbf{A}^{-1}\left(\mathbf{r}_j - \mathbf{r}_{j+1}\right). \tag{5.1}$$

The residual is updated twice in Algorithm 4: first in line 17 and then, in line 28. In line 17, we have $\mathbf{s}_j = \mathbf{r}_{j-1} - \alpha_j\mathbf{v}_j$, which can be written as $\mathbf{s}_j = \mathbf{r}_{j-1} - \alpha_j\mathbf{A_C}\mathbf{p}_j$, hence $\mathbf{r}_{j-1} - \mathbf{s} = \alpha_j\mathbf{A_C}\mathbf{p}_j$. Next, we substitute $\mathbf{r}_j - \mathbf{s}$ in (5.1), which gives

$$\begin{aligned}\mathbf{z}_j &= \mathbf{A}^{-1}\alpha_j\mathbf{A_C}\mathbf{p}_j = \mathbf{A}^{-1}\left(\mathbf{I} - \mathbf{CC^T}\right)\mathbf{A}\alpha_j\mathbf{p}_j \\ &= \alpha_j\mathbf{p}_j - \alpha_j\mathbf{A}^{-1}\mathbf{C}\mathbf{C}^T\mathbf{A}\mathbf{p}_j \\ &= \alpha_j\mathbf{p}_j - \alpha_j\mathbf{U}\mathbf{R}^{-1}\eta_1,\end{aligned}$$

where $\eta_1 = \mathbf{C}^T\mathbf{A}\mathbf{p}_j$. An analogous formula can be derived for the residual update in line 28. In the Algorithm 12, the solution update resulting from recycling, $\alpha_j\mathbf{U}\mathbf{R}^{-1}\eta_1$, is postponed (last line). We do this to avoid multiplication by $\mathbf{U}$ and $\mathbf{R}^{-1}$ at every iteration. In Algorithm 12 the lines in which we accumulate the updates are highlighted in red.

Algorithm 12 rBiCGStab, adapted from [7]

$\mathbf{x}_{-1}$ given initial guess;
Compute $\mathbf{r}_{-1} \leftarrow \mathbf{b} - \mathbf{A}\mathbf{r}_{-1}$;
Compute $\|\mathbf{b}\|$, and set $i \leftarrow 0$;
$\mathbf{U}$ given; want to use $range(\mathbf{U})$ as recycle space. If no $\mathbf{C}$ and $\mathbf{R}$ available, compute $\mathbf{C}$ and $\mathbf{R}$;
Compute $\eta_1 \leftarrow \mathbf{C}^T\mathbf{r}_{-1}$;
$\mathbf{r}_0 \leftarrow \mathbf{r}_{-1} - \mathbf{C}\eta_1$ and $\xi \leftarrow -\eta_1$
Choose `maxit` and `tol`;
Choose $\tilde{\mathbf{r}}$;
while $\|\mathbf{r}_k\| > tol$ **and** $k \leq maxit$ **do**
  $\rho \leftarrow \mathbf{s}^T\mathbf{r}_k$
  if $\rho == 0$ **then**
    FAILED;
  end if
  if $k == 0$ **then**
    $\mathbf{p} \leftarrow \mathbf{r}_k$;
  else
    $\beta \leftarrow (\rho/\rho_{\text{old}})(\alpha/\omega)$;
    $\mathbf{p} \leftarrow \mathbf{r}_k + \beta(\mathbf{p} - \omega\mathbf{v})$;
  end if
  $\mathbf{v} \leftarrow \mathbf{A}\mathbf{p}$;
  $\eta_1 \leftarrow \mathbf{C}^T\mathbf{v}$;
  $\mathbf{v} \leftarrow \mathbf{v} - \mathbf{C}\eta_1$;
  $\alpha \leftarrow \rho/(\mathbf{s}^T\mathbf{v})$;
  if $\|\mathbf{s}\| \leq$ `tol` **then**
    $\mathbf{x}_{k+1} \leftarrow \mathbf{x}_k + \alpha\mathbf{p}$;
    $\mathbf{r}_{k+1} \leftarrow \mathbf{s}$;
    $\xi \leftarrow \xi + \alpha\eta_1$;
    CONVERGED;
  end if
  $\mathbf{t} \leftarrow \mathbf{A}\mathbf{s}$;
  $\eta_2 \leftarrow \mathbf{C}^T\mathbf{t}$;
  $\mathbf{t} \leftarrow \mathbf{t} - \mathbf{C}\eta_2$;
  $\omega \leftarrow (\mathbf{t}^T\mathbf{s})/(\mathbf{t}^T\mathbf{t})$;
  $\xi \leftarrow \xi + \alpha\eta_1 + \omega\eta_2$;
  $\mathbf{x}_{k+1} \leftarrow \mathbf{x}_k + \alpha\mathbf{p} + \omega\mathbf{s}$;
  $\mathbf{r}_{k+1} \leftarrow \mathbf{s} - \omega\mathbf{t}$;
  $\rho_{\text{old}} \leftarrow \rho$;
  $k \leftarrow k + 1$;
end while
$\mathbf{x}_k \leftarrow \mathbf{x}_k - \mathbf{U}(\mathbf{R}^{-1}\xi)$;

5.3 Results

We compare four solvers: rGCROT(m,k), rBiCGStab, GMRES(m), and the hybrid solver. The goal of this comparison is to asses the influence of recycling on runtime and computational cost. We also want to know how to use recycling (which method, with what parameters) to speed up the computations. To make the comparison fair, we first find the optimal parameters (such as the restart frequency or the size of the recycle space) for each solver. Once we obtain optimal parameters, we run the solvers and present the results. We perform this type of analysis for both problems, turbulent channel flow and flow through porous media (see Table 2).

5.3.1 Details of the testing setup

GMRES(m) requires one preconditioned matrix-vector product per iteration and (r)BiCGStab requires two, hence as a metric of comparison we use (1) the number of preconditioned matrix-vector products, and (2) time-to-solution. We do not check for early convergence of (r)BiCGStab (line 23 in Algorithm 12), thus the reported number of preconditioned matrix-vector products for (r)BiCGStab is always even. We only report the results for the pressure Poisson equation, because the systems arising from the momentum conservation equation require just a few solver iterations. Since both applications (turbulent channel flow and flow through porous media) have significant storage cost, we report the storage requirements for the solvers, too.

There exist several methods of comparing performance of two different linear solvers on a sequence of large linear systems. The more straightforward method is to run the simulation to completion with the first solver of interest while gathering the data on time and preconditioned matrix-vector products, and then to do the same for the second solver of interest. The drawback of this approach is that in each case, we solve slightly different linear systems. The advantages are easy to implement timings, and insight on the realistic timings that are observed for the actual simulation run with the solver. An alternative approach is to run one solver as a *master* solver, which determines the solution that is used in the subsequent time step, and run the other solver (*alternate solver*) on the side. Solving exactly the same linear systems with two different solvers gives us a very detailed insight on their performance, although we compute performance characteristics that would not actually be observed while using these solvers independently in GenIDLEST. We use the first approach; because it provides more realistic data. In our case, we start all solvers with the same initial condition (for GenIDLEST). We perform a large number of time steps, hence the effect of couple unfortunate right hand sides is averaged out. The right hand sides for two different solvers are not exactly the same. However, this should not affect the average number of the matrix-vector products and average runtimes per time step.

There are two preconditioners available in GenIDLEST: SSOR and Jacobi, [97, 7]. They have

been optimized for parallel execution. We precondition from the left for all the solvers except BiCGStab, where we apply preconditioning from the right.

5.3.2 Turbulent Channel Flow

In this subsection, we find optimal parameters for rGCROT and GMRES(m) for the turbulent channel flow problem. Our strategy is to find the optimal parameters for rGCROT, and then to compare the performance of the solvers using the optimal parameters. We compare rGCROT with GMRES and rBiCGSTab. For this problem the hybrid method was not faster than rGCROT. For completeness, we also report on memory requirements.

There are six total parameters associated with rGCROT [69]: the cycle length, m, the maximum size of the outer (recycle) space, k, the number of outer vectors after truncation of the outer space (we use $k-10$), the number of inner vectors to select outer vectors from (we use $m/2$), the number of inner vectors selected to extend the the outer space (set to 1), and the number of latest inner vectors kept to extend the outer space (set to 0).

Out of the six parameters listed above, the first two are the most important. We vary k from 20 to 170 in increments of 10. We vary m from 20 to 100, also in the increments of 10. We find that at the start of the simulations (when the systems are hard to solve), rGCROT(30,130) is the optimal choice with respect to the solution time, whereas rGCROT(20,130) is the optimal choice after the flow becomes quasi-stationary.

The take $\mathbf{x}_0 = \mathbf{0}$ as an initial guess at the start of the simulation. The guess is far from the true solution, and hence, the number of iterations is the highest at the start. The guess is improved through the time steps. As the simulation progresses, $\|\mathbf{r}_0\|$ decreases for consecutive systems.

The results in Tables 5.1 and 5.2 show that rGCROT performs the best. It is characterized by the lowest number of preconditioned matrix-vector products and the shortest runtimes. However, rGCROT has expensive iterations; in our case, rGCROT iterations take 1.18 times longer than the iterations of BiCGStab. All solvers perform better towards the end of the simulation (Table 5.2), once the flow is near stationary. It is worth to notice that in the beginning rGCROT(30,130) has not generated a good recycle space yet, and the convergence is slower.

First 30 time steps	**BiCGStab**	**GMRES(50)**	**rGCROT(30,130)**
Time-to-solution, total(s)	365	12680	323
Av. # of matvecs per time step	184	4350	69
Time per iteration /per cycle(s)	0.132 [s/it]	4.84 [s/cycle]	4.68 [s/cycle]
Av. # of iterations/cycles per time step	92 [it]	87 [cycle]	2.3 [cycle]

Table 5.1: Results for first 30 time steps of Turbulent Channel Flow simulation. Each time step corresponds to $5 \cdot 10^{-5}$ seconds. The relative tolerance for the solvers is 10^{-6}. The results were computed on a single machine with Dual Intel Xeon CPU X5650, 2.67GHz, 48GB RAM.

Last 30 time steps	**BiCGStab**	**GMRES(50)**	**rGCROT(30,130)**
Time-to-solution, total(s)	242.7	336.7	164.8
Av. # of matvecs per time step	112	2	1.5
Time per iteration (s)	0.144 [s/it]	5.52 [s/cycle]	3.66 [s/cycle]
Av. # of iterations/cycles per time step	56 [it]	2 [cycle]	1.5 [cycle]

Table 5.2: Results for the last 30 time steps of the turbulent channel flow simulation. Each time step corresponds to $5 \cdot 10^{-5}$ seconds. The initial guess for the first system is $\mathbf{x}_0 = \mathbf{0}$, and the relative tolerance for the solvers is 10^{-6}. The results were computed on a single machine with Dual Intel Xeon CPU X5650, 2.67GHz, 48GB RAM.

Let N be the size of the matrix arising in the pressure Poisson equation. The additional storage cost is $8N$ for BiCGStab, $55N$ for GMRES(50) and $293N$ for rGCROT(30, 130). It is clear from the data in the tables above that rGCROT works well, but its cycles are expensive in terms of time and storage. In contrast, BiCGStab has cheap iterations and low storage cost, however, on average it takes many more iterations to converge. This observation suggests the hybrid approach. Once a good recycle space is computed, we can use a solver with cheaper iterations. We tested the hybrid approach for this problem, but it was not faster than rGCROT(m,k).

5.3.3 Flow Through Porous Media

Here, too, we first tune the parameters for rGCROT(m, k) and for the GMRES(m). We vary both m and k; the remaining four parameters are kept the same as for turbulent channel flow. After extensive testing for rGCROT ($m = 10, 20, \ldots, 80$, $k = 10, 20, \ldots, 210$), the optimal values are found: $m = 10$ and $k = 40$. For GMRES with the Jacobi preconditioner the optimal restart frequency is $m = 30$, and for the SSOR preconditoner, it is $m = 50$.

For the hybrid solver, we use the Jacobi preconditioner, and we vary (1) the time step at which we switch the solver from rGCROT to rBiCGStab and (2) parameters m and k for

rGCROT(m,k). We find that the best choice for the hybrid approach is to switch the solver after the fifth time step. If we switch too early (after one or two time steps), the solver does not converge in the subsequent system solved with rBiCGStab. Switching later creates a computational overhead and does not accelerate the linear solvers in the time steps to follow, see Table 5.5.

Table 5.3 shows that the hybrid solver is the fastest for both preconditioner choices. The rGCROT solver with SSOR preconditioner reduces the number of matrix-vector products the most. This suggests that if the preconditioned matrix-vector product is the limiting factor, one wants to use rGCROT over other solvers. We point out that the results are for ten time steps only. For certain problems, we might want to recompute the recycle space after some number of time steps, based on the changes in the system matrix and right hand side(s). However, we did not investigate this idea. The storage costs are $8N$ (BiCGStab), $55N$ (GMRES(50)), $93N$ (rGCROT(10,40), $88N$ (Hybrid(5)).

	Av. # matvecs per time step		**Total time(s)**	
Solver	**Jacobi**	**SSOR**	**Jacobi**	**SSOR**
BiCGStab	1480 (max 2000)	1432 (max 2000)	3185	4202
Hybrid(5)	454	310	2682	2076
GMRES(30)	1800	-	6497	-
rGCROT(10,40)	420	280	3028	2943

Table 5.3: Total time for 10 time steps of flow through porous media for different preconditioners and solvers. The hyphens indicate that the method with the other preconditioner was worse. The convergence tolerance for all solvers is 10^{-6}. The results were computed on 16 CPU cores (Dual Intel Xeon CPU E5-2670, 2.60GHz, 64GB RAM) with MPI parallelism on BlueRidge HPC cluster.

	# matvecs		**Total time(s)**	
Solver	**Jacobi**	**SSOR**	**Jacobi**	**SSOR**
BiCGStab	(2000)	(2000)	4231	5821
rBiCGStab	436	298	215.9	178
GMRES(30)	1740	-	617.5	-
GMRES(50)	-	950	-	523.2
rGCROT(10)	420	280	298.5	293.4

Table 5.4: Flow through porous media. The table shows the comparison between different solver used in the 10th time step. The numbers in brackets denote that the method performed maximum number of matrix vector products and did not converge. The convergence tolerance for all solvers is 10^{-6}. The results were computed on 16 CPU cores (Dual Intel Xeon CPU E5-2670, 2.60GHz, 64GB RAM) with MPI parallelism on BlueRidge HPC cluster.

# of time steps with rGCROT(10,40)	**Sol. time in 10th time step(s)**	**Total time(s)**
1	unstable	
2	unstable	
3	239.5	3089
4	248.3	3003
5	215.9	2682
6	232.8	2792
7	233.9	2945
8	252.3	2928
9	234.4	2967

Table 5.5: Flow through porous media. The leftmost column shows the number of time steps, at which we use rGCROT(m,k) before we switch the solver to rBiCGStab. The middle column shows the solution time in the 10th time step, which is a good indicator of quality of the recycle space. The convergence tolerance for both solvers is 10^{-6}; the results were computed on 16 CPU cores (Dual Intel Xeon CPU E5-2670, 2.60GHz, 64GB RAM) with MPI parallelism on BlueRidge HPC cluster.

We conclude that the hybrid approach is promising, although it needs further investigation. Especially, the analysis of which preconditioner provides the best performance would be interesting. The hybrid approach is particularly appropriate for problems in which we solve a sequence of slowly changing systems where the (preconditioned) matrices that have high condition numbers. Such systems require many solver iterations (matvecs) to converge. The hybrid approach lowers the computational cost of recycling while providing necessary robustness.

Chapter 6

Using High Performance Computing with Krylov subspace methods

6.1 Motivation

The idea of improving the computational performance of Krylov subspace methods by using parallel machines is not new (see [82, 77, 31, 25, 32, 30, 78]); however, the last ten years have brought a new hardware platform with unique characteristics: General-Purpose Graphic Processing Unit (GPGPU or GPU). In this chapter, we analyze efficient implementation of Krylov subspace methods for the GPU.

In our analysis, we focus on two popular Krylov subspace solvers: GMRES (Chapter 3), proven to be optimal and robust, and BiCGStab (Chapter 3), commonly used due to its cheap iterations and low storage requirements. We look at the computational cost of these methods on the GPU. The computational cost depends on two factors: the number of iterations and the cost per iteration.

For example, in every iteration of GMRES we perform the modified Gram-Schmidt procedure. The computational cost of this procedure (number of flops) increases quadratically with the the number of iterations, and iterations quickly become expensive. The number of iterations needed for GMRES to converge can be reduced with an effective preconditioner. However, the preconditioner-vector product in some cases (for example for ILUT, see Section 6.8) is the most expensive procedure in GMRES. Preconditioner-vector product for weaker preconditioners is much cheaper, however, the number of iterations needed for convergence with a weaker preconditioner is higher, and then the modified Gram-Schmidt procedure is the dominant cost.

On the GPU, the computational cost of a solver iteration does not depend only on the number of flops. The cost depends also on how well can we implement the parts of the iteration (such

as Gram-Schmidt procedure or preconditioner-vector product) in parallel. In Section 6.8, we show that performing a lot of cheap iterations to obtain a solution with a desired accuracy can be a better strategy than performing fewer more expensive iterations, because in the first case, the runtime is shorter.

The purpose of our work is to show how to balance the costs on the GPU using an example problem (LDC, see Chapter 2). We compare different solver and preconditioner choices for the LDC problem. We test GMRES and BiCGStab solvers equipped with SAI (Sparse Approximate Inverse), ILUT (ILU with dual threshold), block ILUT, and line Jacobi preconditioners. We use runtime as our primary metric while comparing *performance* of two different Krylov subspace methods. We also report the number of matrix-vector products as a supplementary measure. We use number of sparse matrix-vector products (SpMVs) instead of the number of iterations, since the number of SpMVs corresponds to the dimension of the constructed Krylov subspace. In one iteration of BiCGStab we expand the Krylov subspace by adding two vectors, and in an iteration of GMRES, we add one vector, so using the number of SpMVs makes the comparison more meaningful.

We emphasize that we are interested only in balancing the costs on the GPU, and we do not make any CPU-GPU comparisons. The literature [14, 10, 59] provides enough evidence that the GPU almost always outperforms the CPU for iterative solvers. In addition, we propose a multi-step SAI (mSAI) preconditioner, which is a modified version of the SAI preconditioner. Multi-step SAI is particularly well-suited for the GPU.

The rest of the chapter is organized as follows. In Section 6.2, we briefly discuss fine grain parallelism. In Section 6.3, we provide an overview of published research on the GPU implementation of Krylov subspace methods. In Section 6.4, we talk about the preconditioners and we develop the theoretical background for multi-step SAI. In Section 6.5, we present an overview of the storage formats for sparse matrices and we discuss the influence of matrix format on the runtime of the sparse matrix-vector multiplication. In Section 6.6, we present the algorithms we developed and implemented for computing SAI and multistep SAI. The results are presented in Section 6.8.

6.2 GPU architecture and fine grain parallelism

The GPU is built of one or more streaming multiprocessors (SMs), and each streaming multiprocessor is equipped with many streaming processors (so called *CUDA cores*). These cores share L1 and L2 cache; GPUs support SIMD (Same Instructions, Multiple Data) parallelism. This type of parallelism is called *fine grain parallelism*, as the GPU can execute thousands of simple operations simultaneously.

Computations on the GPU are very efficient as long as we can formulate (or reformulate) the problem to suit the paradigm of fine grain parallelism. For example, vector addition is a problem that suits fine grain parallelism well. In a simple implementation, each GPU core

adds two numbers (two entries of the vectors) and places the result in the memory. Each core executes the same basic operation but on different data.

Many operations on vectors and matrices are well-suited for the GPU. This is the case for the dot product (dot), vector update (axpy) and scaling (scal), and for the sparse matrix-vector product (SpMV) [14, 59]. Since each Krylov subspace solver is a combination of the above-mentioned operations and preconditioner-vector multiplication (abbreviated as *precvec*), we can expect that the GPU improves runtimes of Krylov subspace methods. In the next subsection, we discuss the available literature on this topic.

6.3 Overview of the published literature

In this section we briefly discuss published literature regarding the implementation of Krylov subspace solvers and preconditioners on the GPUs.

A recurring theme in papers from the last 10 years is to compare performance of a method on the GPU versus the CPU. The authors usually focus a highly efficient GPU implementation. For example, Bahi et al. [10] implement a distributed GMRES algorithm on a GPU cluster, and compare its performance with GMRES on a CPU cluster. Coutourier and Domas [24] compare GMRES implemented on the GPU with GMRES implemented on the CPU and obtain an order of magnitude runtime speedup. In the testing, they use a computer with a single CPU and a single GPU.

Another common theme in the literature is to either optimize the runtime of the solver (assuming the choice of the preconditioner is fixed) or to find a a good preconditioner for the GPU (assuming the choice of the solver is fixed). In the paper by Coutourier and Domas, the authors use line Jacobi preconditioner and optimize the runtime of GMRES with line Jacobi. Dehnavi et al. [33] use BiCGStab with a Sparse Approximate Inverse (SAI) preconditioner, and demonstrate the efficiency of this combination for seven matrices from University of Florida Matrix Collection. The paper includes the preconditioner computation time in the experimental results. For the matrices, for which solver converges in less than 100 iterations, preconditioner computation takes more time than the solver. Because this cost can be so significant, we consider it in our analysis.

Lukash et al. [60] also use BiCGStab as a solver and apply a modified version of SAI as a preconditioner. Their approach, however, requires the matrix to be symmetric positive definite. Labutin and Surodina [56] investigate the performance of SAI using Conjugate Gradient (CG). Wang et al. [98] use GMRES and test block ILU preconditioner and a polynomial preconditioner.

Since ILU preconditioners are usually effective in reducing the number of the linear solver iterations, many researchers work on an efficient GPU implementations of both, computing the ILU decomposition of a matrix, and using this decomposition as a preconditioner in an

iterative solver. However, parallelizing triangular solves, required in precvec, is difficult [71, 6], because triangular solvers are *inherently sequential*. In [21], Edmond Chow proposes a new approach to incomplete LU decomposition on the GPU. This approach allows to evaluate each non-zero of **L** and **U** factors in parallel.

We can look at a Krylov subspace method as on a collection dot products, vector updates and scaling, SpMVs, and precvecs. In the literature from the past ten years, we can find a lot of results regarding efficient GPU implementation of these *building blocks*. For example, Bell and Garland [14] analyze the best strategies for the GPU-based matrix-vector products and test multiple matrix formats.

Several authors focus on developing GPU-based libraries for Krylov subspace methods. For example, Li and Saad [59] provide a general-purpose, well-documented library, which contains contains CG and GMRES solvers and a few preconditioners (ILUT, block ILUT, a polynomial preconditioner, SSOR) implemented and optimized for the GPU. The library is called CUDA_ITSOL and it is available online. This library is a starting point for our implementation.

6.4 Solvers and preconditioners on the GPU

The linear systems arising in the LDC problem are non-symmetric. As mentioned earlier, we test two Krylov subspace solvers, GMRES(m) and BiCGStab, for these matrices. GMRES uses modified Gram-Schmidt (mGS) to enforce orthogonality among the vectors that span the Krylov subspace. The mGS procedure consists of vector updates and dot products, which are fast on the GPU; however, the number of operations in mGS increases quadratically with the number of solver iterations, and the operations inside the mGS loop (lines 8–11 in Algorithm 1) have to be executed in a specific order. Moreover, in GMRES we need to store all the vectors forming the basis of the Krylov subspace. If the vectors are stored in the global GPU memory, then mGS increases memory overhead as well. Hence mGS not only makes the iterations expensive, but also is difficult to parallelize on the GPU.

In contrast, BiCGStab does not require orthogonality among the vectors that span the subspace. BiCGStab also has cheap iterations with constant number of flops per iteration. Moreover, in order to add a new vector to the space, we only need two previously computed Krylov subspace vectors. Thus, the method has very low storage requirements, i.e., if the matrix in the system (1.1) has dimensions $n \times n$, the total storage needed in an efficient implementation of BiCGStab is $7n$. The disadvantage of BiCGStab is its irregular convergence. Compared with GMRES, BiCGStab might take more SpMVs (and more precvecs) to converge (which is the case for the LDC problem).

6.4.1 Sparse Approximate Inverse (SAI) Preconditioner

Sparse Approximate Inverse (SAI) preconditioner has two advantages on the GPU. First, preconditioner computation is well suited for fine grain paralellism. Second, precvec for this preconditioner is an SpMV (sparse-matrix vector multiplication), so it does not require triangular solves. Next, we briefly explain the idea of preconditioner setup.

Let $\mathbf{A}$ be an $n \times n$ sparse matrix. Let $\mathbf{M}$ be a sparse matrix with a sparsity pattern chosen *a priori*. The idea is to find $\mathbf{M}$ that minimizes the Frobenius norm

$$\|\mathbf{AM} - \mathbf{I}\|_F, \tag{6.1}$$

where $\mathbf{I}$ is an $n \times n$ identity matrix. We usually choose the non-zero pattern of $\mathbf{M}$ to be a non-zero pattern of $\mathbf{A}$ or its powers.

For every column $\mathbf{M}_k$ of $\mathbf{M}$ with $k = 1, \ldots, n$,we solve an independent least squares problem

$$\mathbf{M}_k = \min_{\tilde{\mathbf{M}}_k} \|\mathbf{A}\tilde{\mathbf{M}}_k - \mathbf{I}_k\|_2, \tag{6.2}$$

where the minimum is taken over all column vectors $\tilde{\mathbf{M}}_k$ with the prescribed non-zero pattern.

For every column k, $k = 1, \ldots, n$, the problem (6.2) can be either solved adaptively (the non-zero pattern of $\mathbf{M}_k$ is adapted until the norm in (6.2) reaches a tolerance threshold), or directly, with a non-zero pattern for $\mathbf{M}_k$ chosen in advance and never changed. The second technique is computationally cheaper, and leads to similar results as the adaptive one [20].

If we choose a non-zero pattern of $\mathbf{A}^s$ with $s \geq 1$ for $\mathbf{M}$, then the non-zero pattern of $\mathbf{AM}$ is the same as non-zero pattern of $\mathbf{A}^{s+1}$. Hence, (6.2) is a least squares problem with more rows than columns. In our implementation of SAI, we do not use the fact that the non-zero pattern of $\mathbf{AM}$ is a product of non-zero patterns of $\mathbf{A}$ and $\mathbf{M}$. Instead, we use the non-zero pattern of $\mathbf{M}$, which results in a least squares problems with the same number of columns and rows.

Each column of the matrix $\mathbf{M}$ can be set up independently, and thus, the columns can be computed in parallel [46]. While computing $\mathbf{M}$, we solve a large number of independent, small least squares problems, which can all be solved in parallel on the GPU. In Section 6.6 we provide the details of the implementation.

6.4.2 Multi-step Sparse Approximate Inverse

The numerical results presented in Section 6.8 show that SAI is very efficient in terms of preconditioner computation and precvec times, however, linear solvers accelerated by SAI take a lot of iterations to converge. We can improve the effectiveness of SAI by applying several steps of fixed-point iteration with $\mathbf{AM}$ (Richardson iteration in our case).

Let $\mathbf{M}$ be a SAI preconditioner and $\mathbf{I}$ be an identity matrix. We define $\mathbf{G} = -\mathbf{R} = \mathbf{I} - \mathbf{AM}$. We write

$$\mathbf{AMx} = \mathbf{b} \Leftrightarrow \mathbf{x} = (\mathbf{I} - \mathbf{AM})\mathbf{x} + \mathbf{b},$$

which leads to an iteration

$$\mathbf{x}_{k+1} = (\mathbf{I} - \mathbf{AM})\mathbf{x}_k + \mathbf{b} = \mathbf{G}\mathbf{x}_k + \mathbf{b}. \tag{6.3}$$

Assuming $\mathbf{x}_0 = 0$, (6.3) becomes

$$\mathbf{x}_{k+1} = \left(\mathbf{G}^{m-1} + \mathbf{G}^{m-1} + \ldots + \mathbf{G} + \mathbf{I}\right)\mathbf{b}.$$

If $\mathbf{G}$ is invertible, or, equivalently, $\rho(\mathbf{G}) < 1$, we get

$$\left(\mathbf{G}^{m-1} + \mathbf{G}^{m-1} + \ldots + \mathbf{G} + \mathbf{I}\right) \approx (\mathbf{I} - \mathbf{G})^{-1}. \tag{6.4}$$

Since $\mathbf{G} = \mathbf{I} - \mathbf{AM}$, hence from (6.4) we get

$$\mathbf{AM}\left(\mathbf{G}^{m-1} + \mathbf{G}^{m-1} + \ldots + \mathbf{G} + \mathbf{I}\right) \approx \mathbf{I}. \tag{6.5}$$

We define a new preconditioner $\hat{\mathbf{M}}$ as

$$\hat{\mathbf{M}} := \mathbf{M}\left(\mathbf{G}^{m-1} + \mathbf{G}^{m-1} + \ldots + \mathbf{G} + \mathbf{I}\right). \tag{6.6}$$

We refer to $\hat{\mathbf{M}}$ as *multi-step SAI* or *m-step SAI* or *mSAI*.

Next, we use the definition of $\mathbf{G}$ to simplify (6.5)

$$\mathbf{A}\hat{\mathbf{M}} = (\mathbf{I} - \mathbf{G})\left(\mathbf{G}^{m-1} + \mathbf{G}^{m-1} + \ldots + \mathbf{G} + \mathbf{I}\right) = \mathbf{I} - \mathbf{G}^m. \tag{6.7}$$

Formula (6.7) combines multi-step SAI precvec and SpMV with $\mathbf{A}$ into one operation.

In our benchmarking application, we use $m = 3$, which leads to $\mathbf{A}\hat{\mathbf{M}}x = (\mathbf{I} - \mathbf{G}^3)\,\mathbf{x} = (\mathbf{I} + \mathbf{R}^3)\,\mathbf{x}$. Therefore, we replace separate SpMV and separate precvec with one vector update and three SpMVs with the matrix $\mathbf{R}$. The matrix $\mathbf{R} = \mathbf{AM} - \mathbf{I}$ has the non-zero pattern of $\mathbf{A}^2 - \mathbf{I}$, which makes it denser than $\mathbf{M}$. Hence, the cost of SpMV with $\mathbf{R}$ is slightly higher compared with SpMV with $\mathbf{A}$ or $\mathbf{M}$. At the end of the solver iteration we must multiply the solution by $\hat{\mathbf{M}}$. From formula (6.6), we get for $m = 3$:

$$\hat{\mathbf{M}}\mathbf{x} = \mathbf{M}\left(\mathbf{I} - \mathbf{R} + \mathbf{R}^2\right)\mathbf{x}, \tag{6.8}$$

which requires three matrix-vector products and two vector updates. However, this is typically done after many iterations (once per GMRES cycle and once per BiCGStab run).

The resulting preconditioner $\hat{\mathbf{M}}$ turns out to be very effective for the LDC problem, since the number of solver iterations is comparable to the number of solver iterations with ILUT, yet $\hat{\mathbf{M}}$ is much better suited for fine grain parallelism and has a better flop to data movement ratio, which leads to much shorter runtimes, too.

6.5 Matrix format

There are many well-established storage schemes for sparse matrices. One simple and popular format is the coordinate list (COO) format, in which the non-zeroes are stored as triplets consisting of a row number, a column number, and a value of corresponding matrix entry. Another common storage scheme is compressed sparse row (CSR) or compressed sparse column (CSC). In the CSR format, an $n \times n$ matrix with m non-zeros, is stored using three arrays: an m-element array of values, `A` (stored row after row, starting from the top row), an m element array `JA`, which contains column positions of elements from the array `A`, and a pointer, n-element array `IA`. In the last array, the jth entry points to the first entry of the row j in arrays `A` and `JA`. CSC storage is analogous to CSR, but the matrix is stored columnwise. If the matrix has a diagonal structure (all the non-zeros are placed on small number of diagonal), it can be stored using diagonal storage (DIA). In this format, the matrix is stored as an array of size $d \times n$, where d is the number of diagonals. Jagged Diagonal format (JAD), which is a generalization of DIA, is particularly well suited for matrices with *nearly diagonal* structure. We explain this format using an example.

Let

$$\mathbf{A} = \begin{bmatrix} 1 & 2 & 0 & 0 & 0 \\ 5 & 0 & 9 & 8 & 0 \\ 6 & 9 & 0 & 3 & 2 \\ 0 & 3 & 4 & 0 & 0 \\ 12 & 14 & 15 & 0 & 0 \end{bmatrix}$$

We permute the rows of $\mathbf{A}$; the rows with highest number of non-zeros are placed at the top. We also create a permutation array `Perm` that stores the permulation information.

$$\mathbf{PA} = \begin{bmatrix} 6 & 9 & 0 & 3 & 2 \\ 5 & 0 & 9 & 8 & 0 \\ 12 & 14 & 15 & 0 & 0 \\ 1 & 2 & 0 & 0 & 0 \\ 0 & 3 & 4 & 0 & 0 \end{bmatrix}, \quad \texttt{Perm} = \begin{bmatrix} 3 \\ 2 \\ 5 \\ 1 \\ 4 \end{bmatrix}$$

Next, we move all the non-zeros to the left.

$$\begin{bmatrix} 6 & 9 & 3 & 2 \\ 5 & 9 & 8 & - \\ 12 & 14 & 15 & - \\ 1 & 2 & - & - \\ 3 & 4 & - & - \end{bmatrix}$$

The maximum number of non-zeros in a row is 4. Hence, we save the matrix using 4 jagged

diagonals.

$$\begin{bmatrix} 6 & 9 & 3 & 2 \\ 5 & 9 & 8 & 0 \\ 12 & 14 & 15 & 0 \\ 1 & 2 & 0 & 0 \\ 3 & 4 & 0 & 0 \end{bmatrix} \tag{6.9}$$

The structure is stored using four arrays:

$$\begin{aligned} \texttt{A} &= \begin{bmatrix} 6 & 5 & 12 & 1 & 3 & 9 & 9 & 14 & 2 & 4 & 3 & 8 & 15 & 2 \end{bmatrix} \\ \texttt{IA} &= \begin{bmatrix} 1 & 6 & 11 & 14 & 15 \end{bmatrix} \\ \texttt{JA} &= \begin{bmatrix} 1 & 1 & 1 & 1 & 2 & 2 & 3 & 2 & 2 & 3 & 4 & 4 & 3 & 5 \end{bmatrix} \\ \texttt{Perm} &= \begin{bmatrix} 3 & 2 & 5 & 1 & 4 \end{bmatrix} \end{aligned}$$

The array `A` contains the values, stored diagonal after diagonal. Entry j of `IA` points to the start of the jth jagged diagonal in `A`. The array `JA` contains column indices of the entries in `A` and `Perm` stores permutation information. We often want the jagged diagonals to have the same length, or we want the length of every diagonal to be divisible by 32 (for efficient GPU SpMV), and hence we pad the diagonals with zeros, see (6.9). More detailed explanation of popular matrix formats can be found in [59].

Chapter 2, section 2.1 shows the non-zero structure of the matrices resulting from the LDC problem. Storing the LDC matrices in a diagonal format would require padding with zeroes in multiple diagonals due to the boundary conditions. JAD scheme requires a lot less padding.

Storage format is important for the performance of the matrix-vector product. Some formats lead to an ineffective memory access pattern. This is often the case for CSR [14]. Ineffective memory use can also occur for JAD if the matrix has many non-zeroes per row.

Figure 6.1 shows the cost of SpMV for the matrices obtained from the LDC simulation. The characteristics of these matrices are given in Table 6.1. Every matrix was multiplied 100 times by the same random vector. Time results shown in Figure 6.1 are per 100 SpMVs. The time is measured by a system wall timer (not with CUDA events). For a small matrix (2700×2700, which corresponds to 30×30 grid, the format, in which the matrix is stored, does not impact the performance as much as it does for the larger matrices (30603×30603 and larger, which corresponds to 101×101 and larger discretization grids). It is clear from Figure 6.1 that SpMV with the bigger LDC matrices stored in JAD format is the fastest. All the matrices can be stored using 7 jagged diagonals.

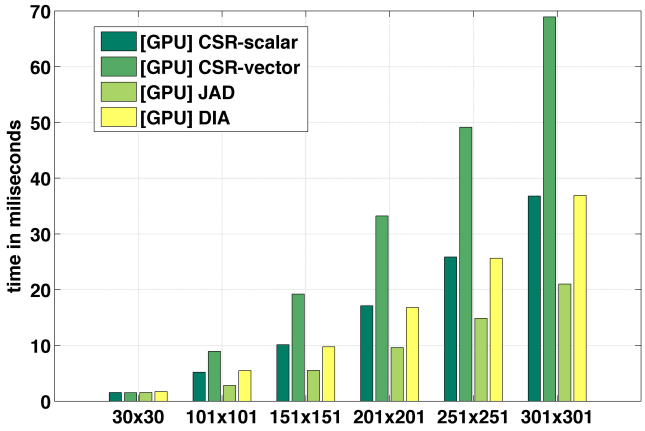


Figure 6.1: Runtime of the matrix-vector multiplication for LDC matrices using different storage formats. The bars show time in milliseconds per 100 SpMVs with the same random vector. The runtimes were computed on Tesla K20c GPU.

grid size	**matrix size**	**nnz**	**Average # of nnz per row**
30×30	2700×2700	15592	5.77
101×101	30603×30603	188619	6.16
151×151	68403×68403	425419	6.22
201×201	121203×121203	757219	6.25
251×251	189003×189003	1184019	6.26
301×301	271803×271803	1705819	6.28

Table 6.1: Basic characteristics of the matrices used in testing the influence of the matrix storage format on the performance of SpMV.

6.6 Algorithms

In this section, we discuss the algorithms we developed and optimized for SAI, multistep SAI and Jacobi preconditioner computation.

6.6.1 Sparse Approximate Inverse: preconditioner computation

We use the non-zero pattern of $\mathbf{A}$ for the SAI preconditioner. This approach carries some benefits. First, there is no additional time required to compute (set up) the pattern. Second, we need to store only the non-zero values of $\mathbf{M}$, since the remaining parts of the matrix structure are identical with $\mathbf{A}$. This means that if we store $\mathbf{A}$ in JAD or CSR format, we do not need to store the arrays `JA`, `IA` and `Perm` for $\mathbf{M}$ (see Section 6.5).

Let us assume that matrix `A` is available in CSR format (as `CSR-A`, `CSR-JA`, and `CSR-IA`) and in JAD format (as `JAD-A`, `JAD-JA`, `JAD-IA`, and `JAD-Perm`). Since $\mathbf{M}$ is computed column-wise, we first convert $\mathbf{A}$ from CSR to CSC, which can be efficiently done on the GPU using a cuSPARSE library function. Next, we use a CUDA kernel, which computes the columns of $\mathbf{M}$ in parallel. Inside the kernel, we form a least squares problem for every column and solve it using QR factorization with Householder transformation. We store the result in CSC. Then, we convert the result to CSR (using cuSPARSE) and to JAD (using our kernel). The last conversion is done, because for LDC matrices, SpMV with the matrix stored in JAD format is more efficient than SpMV with the matrix stored in CSR format.

Algorithm 13 shows the steps for preconditioner computation, and Listing 6.1 shows how we convert the matrix from CSR to JAD on the GPU. Note that we do not store the non-zero pattern of the matrix $\mathbf{A}$ (and $\mathbf{M}$) in a special data structure. We are able to form the least squares problem by manipulating the arrays `CSR-IA`, `CSR-JA`, `CSC-IA`, and `CSC-JA`, which are parts of available CSR and CSC data structures.

The conversion in line 20 of Algorithm 13 can be done very fast in parallel, since we already have the JAD structure of $\mathbf{A}$, and we can use it in the code.

6.6.2 Computing pattern of $\mathbf{A}^2$

For two-step SAI, we need the pattern of $\mathbf{A}^2$. The LDC problem is based on a five point stencil. The velocity components and the pressure for a point on the grid depend on its left, right, bottom and top neighbors (see Figure 2.1 and the left side of Figure 6.2). Thus, if we consider a matrix $\mathbf{A}^2$, it can be thought of as a matrix formed based on thirteen point stencil with special precautions applied to the points on the grid boundaries.

Consider the adjacency graph induced by five point stencil. The vertices of the graph correspond to the grid points. We place directed edges between the grid point (vertex) and its

Listing 6.1: CUDA code for the GPU conversion between CSR and JAD formats

```
__global__  void makeJADs(int n,  int nnz, int njad, int * perm, double
   *a, int * ia, int *jadia, double *jada){
        /*
           njad  - number of Jagged diagonals
           perm - JAD array for row permutation
           a - CSR array containing matrix values
           ia - CSR array containing row start indices (indices start at
               1)
           ja - CSR array containing column indices
           jadia - JAD array containing start indiceces of Jagged
              diagonals
           jada - JAD array for matrix values
        */
    int k = blockIdx.x * blockDim.x + threadIdx.x;
    int i;

    if (k<n){
        int indx = perm[k]-1;
        for (i=ia[indx]; i<ia[indx+1]; i++){
            jada[jadia[i-ia[indx]]-1+k] = a[i-1];
        }}
}
```

Algorithm 13 Computation of the SAI preconditioner

Data: Matrix $\mathbf{A}$ in CSC, CSR and JAD format
Result: SAI preconditioner $\mathbf{M}$ in JAD format
for $k = 1, \ldots, n$ *in parallel* **do**
  Find the indices of the non-zero elements in $\mathbf{A}_k$ and copy them to $\texttt{Ind}_k$
  for $j = 1, \ldots, \text{length}(\texttt{Ind}_k)$ **do**
    $l \leftarrow \texttt{Ind}_k(j)$
    if $l == k$ **then**
      $\texttt{Rhs}_k(j) \leftarrow 1$
    else
      $\texttt{Rhs}_k(j) \leftarrow 0$
    end if
    for $h = 1, \ldots, \text{length}(\texttt{Ind}_k)$ **do**
      $m \leftarrow \texttt{Ind}_k(h)$
      $\texttt{P}_k(j, h) \leftarrow \mathbf{A}(l, m)$
    end for
  end for
  Use QR factorization with Householder transformation to solve $\texttt{P}_k \mathbf{x}_k = \texttt{Rhs}_k$
  Save the solution $\texttt{x}_k$ using the CSC structure of the matrix $\mathbf{A}$.
end for
Convert $\mathbf{M}$ from CSC to CSR and then to JAD, reusing the JAD structure of $\mathbf{A}$

neighbors. We can find the non-zero pattern of $\mathbf{A}^2$ based on the principle *neighbors of my grid point neighbor become my neighbors*. Figure 6.2 illustrates the idea. On the left of Figure 6.2, we see a grid point (j, k) and its four neighbors. The gray circles are the neighbors of neighbors of the grid point (j, k). In $\mathbf{A}^2$, the gray grid points become neighbors of grid point (j, k), as shown on the right side of the Figure 6.2. This principle applies to other problems using similar type of discretization grid.

We developed a GPU kernel that identifies potential neighbors-of-neighbors for a given matrix row (one thread processes one matrix row). Using this approach, we are able to find the non-zero pattern of $\mathbf{A}^2$ in a very fast manner. Unlike for SAI, the non-zero pattern of $\mathbf{A}^2$ must be stored in a separate data structure. For convenience, we store the pattern in CSR, CSC and JAD formats. However, we only need to store arrays `IA` and `JA` (for CSR, CSC, and JAD), and `Perm` (for JAD) but there is no need to store the arrays of matrix values.

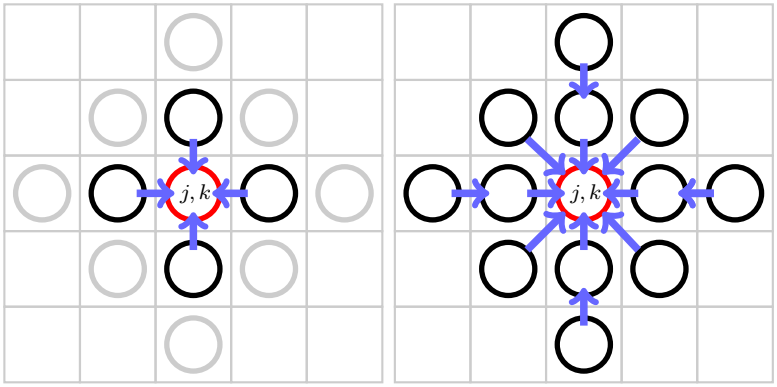


Figure 6.2: Illustration of neighbor-of-neighbor principle for 5 point stencil.

Algorithm 14 Computation of the pattern of $\mathbf{A}^2$

1: **Data:** Matrix $\mathbf{A}$ in CSC and CSR
2: **Result:** Non-zero pattern of $\mathbf{A}^2$ as CSR, CSC and JAD
3: Create bindless texture object for arrays `CSR-IA`, `CSR-JA`, `CSC-IA`, `CSC-JA` (for parts of CSR and CSC structure of $\mathbf{A}$)
4: **for** Every row, in parallel **do**
5: Identify the type of the grid point (interior, corner, boundary);
6: Generate the list of potential neighbors-of-neighbors
7: Check suitable rows and columns for matches
8: Save results as a list of column indices of non-zeroes
9: Save the length of the list
10: **end for**
11: Use prefix sum algorithm on the GPU to create array `S-CSR-IA` for CSR storage of the pattern;
12: Based on array created in the previous step, copy the column indices to array `S-CSR-JA` for CSR storage of the pattern;
13: Convert the pattern to JAD format.

In Algorithm 14, we perform the operation in line 11 in two steps. Prefix sum is needed to correctly set up the array `S-CSR-IA` with pointers to the row starts in CSR format (we need to know how many entries are in each row). In general, prefix sum is very efficient on the GPU if computed using one block of threads. The maximum size of a block of threads on a Kepler class GPUs is 1024. Since in our tests, we use matrices with dimensions 30603×30603 and larger, we need to compute a prefix sum of an array with at least 30603 elements. Therefore, we first split the array into segments of 1024 elements and compute prefix sum for each segment

separately using one thread block with 1024 threads per segment. As a result, we obtain an array of partial prefix sums. In every segment of length 1024 of this array, the entry j with $j = 1, \ldots, 1024$ contains a sum of the entries from $k = 1$ to $k = j$. Next, we run another GPU kernel, also with thread blocks of size 1024. In block k, we add the sum of elements from blocks $l = 1, \ldots, k-1$ to every element of the corresponding segment of the array of partial sums.

6.6.3 Computing R for two-step SAI

In Algorithm 15, we first compute $\mathbf{M}$ as in Algorithm 13, and then we compute $\mathbf{R}$ (in the implementation, this is done using two separate GPU kernels). The algorithm outputs both $\mathbf{M}$ and $\mathbf{R}$. Matrix $\mathbf{M}$ is needed in (6.8).

6.7 Implementation

The starting point for the implementation of the Lid Driven Cavity with GPU-accelerated Krylov subspace solvers was the CUDA_ITSOL package [59]. We expanded the package with a GPU-based implementation of BiCGStab. We also added GPU-based implementations of the SAI, the multi-step SAI and the line Jacobi preconditioners. Then, the expanded CUDA_ITSOL package was added to the Lid Driven Cavity code.

6.7.1 ILUT and Block ILUT preconditioners

ILUT (Incomplete LU factorization with dual threshold) [79] is one of the preconditioners provided by the authors of the CUDA_ITSOL package. In order to setup ILUT, we follow [59]. The $\mathbf{L}$ and the $\mathbf{U}$ factors are computed on the CPU. We use 10^{-1} as a fill-in threshold, and we limit the maximum number of non-zeroes per row to a 100. Next, the ILUT data structure is copied to the GPU. During the setup of ILUT, we also split the coefficients in $\mathbf{L}$ and $\mathbf{U}$ into so-called levels. All the coefficients within the same level can be computed in parallel in forward/backward substitution.

In the triangular solve, the levels are processed one after the other and the parallelism is limited to computations within a single level .The forward solve must be completed before the backward solve starts. This leads to a large number of CUDA kernels, each yielding a small workload. For the LDC matrices, we obtain from a 600 to over a 1,000 levels for both $\mathbf{L}$ and $\mathbf{U}$, which leads to from 600 to a $1,000$ kernels for the forward substitution and the same number of kernels for the backward substitution. This implementation results in a very inefficient use of the GPU, since the GPU idle time between kernel calls is often longer then the kernel execution time. We experimented with combining the forward and backward solve

Algorithm 15 Computation of the matrix $\mathbf{R}$ used in two-step SAI

Data: Matrix $\mathbf{A}$ in CSR and CSC format; $\mathbf{S}$, the pattern of $\mathbf{A}^2$ in CSC format (arrays `JA` and `IA`).
Result: Matrices $\mathbf{M}$ and $\mathbf{R}$ in CSC, CSR and JAD format
for $k = 1, \ldots, n$ *in parallel* **do**
  Find the indices of the non-zero elements in $\mathbf{A}_k$ and copy them to $\texttt{Ind}_k$
  for $j = 1, \ldots,$ length($\texttt{Ind}_k$) **do**
    $l \leftarrow \texttt{Ind}_k(j)$
    if $l == k$ **then**
      $\texttt{Rhs}_k(j) \leftarrow 1$
    else
      $\texttt{Rhs}_k(j) \leftarrow 0$
    end if
    for $h = 1, \ldots,$ length($\texttt{Ind}_k$) **do**
      $m \leftarrow \texttt{Ind}_k(h)$
      $\texttt{P}_k(j,h) \leftarrow \mathbf{A}(l,m)$
    end for
  end for
  Use QR factorization with Householder transformation to solve $\texttt{P}_k\mathbf{x}_k = \texttt{Rhs}_k$
  Save the solution $\texttt{x}_k$ using the CSC structure of the matrix $\mathbf{A}$.
  Copy the indices of non-zeros of the kth column of the pattern, $\mathbf{S}_k$ to $\texttt{IndS}_k$
  Initialize array $\texttt{r}_k$ with size length($\texttt{IndS}_k$);
  for j=1..., length($\texttt{Ind}_k$) **do**
    $l \leftarrow \texttt{IndS}_k(j)$
    $\texttt{r}_k \leftarrow \texttt{r}_k + \mathbf{A}(:, \texttt{IndS}_k) * \mathbf{M}_k(l)$
    if k==l **then**
      $\texttt{r}_k(l) \leftarrow \texttt{r}_k(l) - 1$
    end if
  end for
end for
Convert $\mathbf{M}$ to CSR, and then to JAD
Convert $\mathbf{R}$ to CSR, and then to JAD

into one kernel. This strategy improves the performance of forward/backward substitution for smaller LDC matrices (for the 101×101 grid and 151×151 grid), but significantly diminishes performance for larger grids (such as 351×351). Thus, we decided to continue to use the original version (the version implemented in CUDA_ITSOL).

Block ILUT (abbreviated as BILUT) is a block version of ILUT. The matrix is split into n_b blocks using a domain decomposition algorithm (we use $n_b = 32$, since our extensive testing demonstrated that this is the optimal number for the LDC matrices, regardless of the grid size). Then, the ILUT factorization is computed on the CPU for each block, and the $\mathbf{L}$ and

U factors are copied to the GPU. Preconditioner-vector multiplication requires n_b smaller triangular solves. The approach to parallelism is, however, different than for ILUT, because in BILUT distinct matrix blocks are processed simultaneously by different thread blocks. While executing the forward or backward solve for one block, we use levels as for ILUT, and the parallel reduction is applied in order to compute a partial result.

6.7.2 Two-step SAI preconditioner

Using this preconditioner, we can combine SpMV and precvec into one operation. At the start of the solver execution, we pin the arrays `A`, `JA` and `Perm` (parts of JAD structure of $\mathbf{R} = \mathbf{AM} - \mathbf{I}$) to the texture memory. The multiplication is performed as three separate SpMVs with **R**. Kernel calls serve as host side synchronization. We use one auxiliary vector to store the intermediate result.

6.7.3 Jacobi preconditioners

As shown in Chapter 2, Section 1, the structure of matrices coming from the LDC problem is such that the 3×3 matrices located along the diagonal are, in fact, diagonal matrices. Hence, we can only use line Jacobi. Even though using the Jacobi preconditioner causes an increase in the number of iterations compared with ILUT, BILUT, and SAI, it leads to surprisingly good runtimes, as is shown in the next section.

6.8 Results

In this section, we give a detailed overview of the runtimes for several solvers and preconditioners used to solve the linear systems arising in the LDC simulation. We analyze how various choices (combinations of solver and preconditioner) influence the runtime. We search for a balance between effectiveness (low number of solver iterations) and efficiency (short runtime); this balance is very different on the GPU than on the CPU. Based on the numerical results, it is clear that weaker preconditioners (SAI and Jacobi) can be more efficient than ILUT/BILIT on the GPU. We also show the results for the two-step SAI preconditioner. This preconditioner is more effective than SAI but more efficient than ILUT.

All the results, unless explicitly stated otherwise, were computed on a server running Linux Debian (release: jessie). The server was equipped with a Tesla K20c GPU and an Intel Xeon 2.0 Ghz E5405 CPU. All the experiments were performed using GCC 4.8.2 and NVCC 6.0. We used CUDA 6.0. and suitable CUDA libraries distributed with CUDA 6.0.

6.8.1 GMRES(m)

We first solve the LDC problem using GMRES(40) with ILUT and BILUT as preconditioners.

Grid size	# MVs	GMRES(40) ms	Prec-V ms	Prec-V %	GS ms	GS %	MV ms	MV %	Remaining ms	Remaining %	BiCGStab w. SAI – ms
101 × 101	21	261.4	233.5	89.3%	14.9	5.7%	2.4	0.9%	10.6	4.1%	46.6
151 × 151	24	420.7	379.3	90.2%	21.0	5.0%	3.6	0.9%	16.8	3.9%	46.5
201 × 201	23	486.2	435.9	89.7%	25.2	5.2%	5.0	1.0%	20.0	4.1%	56.6
251 × 251	23	578.4	527.7	91.2%	29.7	5.2%	7.0	1.2%	14.0	2.4%	71.3
301 × 301	23	689.4	631.3	91.6%	35.3	5.1%	8.8	1.3%	14.0	2.0%	86.1
351 × 351	23	815.8	742.0	91.0%	42.0	5.1%	11.7	1.4%	20.1	2.5%	107.6

Table 6.2: Average runtimes (per linear system) for GMRES(40) with ILUT as a preconditioner. We give time in milli1seconds on the left and the percentage of the total solver time on the right. We also provide runtimes for main parts of the solver: SpMV time (MV), precvec time (PrecV), modified Gram-Schmidt time (GS). The last column shows the runtimes for BiCGStab with SAI applied to the same problems. This shows how much the runtimes improve by change of preconditioner and solver.

Table 6.2 shows average runtimes for GMRES(40) with ILUT as a preconditioner. The convergence tolerance is 10^{-8}. We run the LDC code for multiple time steps (until the LDC residuals for velocity components are smaller than 10^{-1}, see Section 2.1). We use CFL numbers 18, 24, 30, 36, 42 and 48 for increasing grid sizes.

Table 6.2 shows that the relative cost of the ILUT preconditioner multiplication (forward and backward solve) is high – for example, for a 151×151 grid it takes about $379.3/24 \approx 15.8$ms per single forward/backward solve). Especially compared with the very low cost of SpMV (for the same 151×151 grid, the cost of a single SpMV is 0.15ms), this cost is high (90% of time is taken by precvec and only approximately 1% is taken by SpMV). The sequential precvec outweighs even the modified Gram-Schmidt procedure, which typically attributes to the highest cost in GMRES(m). We expect better performance with preconditioners for which precvec works as an SpMV (SAI and its variations [56, 98, 60], AINV [22], Jacobi [80]), because for these preconditioners, precvec is much better suited for fine grain parallelism.

Using BILUT as a parallel preconditioning technique is strongly recommended in the literature, see [98, 80]. Table 6.3 shows that BILUT leads to an improvement in the runtime for larger grids. We observe that for all but the 101×101 grid, the time for the preconditioner-vector multiplication decreases compared with ILUT. However, since the number of iterations increases by a factor 1.5, the modified Gram-Schmidt procedure takes roughly three times longer, and hence the overall improvement is modest.

Next, we evaluate the SAI preconditioner, for which the preconditioner-vector multiplication is an SpMV. SAI is not as effective as ILUT in terms of decreasing the number of iterations, but, as shown in Table 6.4, it makes the preconditioner-vector multiplication very cheap. Although

Grid size	# MVs	GMRES(40) ms	Prec-V ms	Prec-V %	GS ms	GS %	MV ms	MV %	Remaining ms	Remaining %
101×101	48	365.2	283.8	77.7%	69.3	19.0%	5.5	1.5%	6.6	1.8%
151×151	41	388.3	304.3	78.3%	72.5	18.7%	6.1	1.6%	5.3	1.4%
201×201	37	470.0	373.2	79.4%	83.6	17.8%	8.1	1.7%	5.1	1.1%
251×251	35	550.3	445.1	80.9 %	89.0	16.2%	10.6	1.9%	5.6	1.0%
301×301	34	652.4	530.7	81.4%	102.2	15.7%	13.1	2.0%	6.4	0.9%
351×351	33	759.9	616.5	81.1%	119.1	15.6%	16.9	2.2%	7.4	0.9%

Table 6.3: Average runtimes (per linear system) for GMRES(40) with BILUT as a preconditioner. We give time in milli1seconds on the left and the percentage of the total solver time on the right. We also provide runtimes for main parts of the solver: SpMV time (MV), precvec time (PrecV), modified Gram-Schmidt time (GS).

Grid size	# MVs	GMRES(40) ms	Prec-V ms	Prec-V %	GS ms	GS %	MV ms	MV %	Remaining ms	Remaining %
101×101	93	171.6	10.7	6.1%	121.6	71.0%	2.4	6.2%	28.6	16.7%
151×151	73	160.8	10.7	6.7%	102.4	63.7%	11.1	6.9%	36.6	22.7%
201×201	71	182.8	14.4	7.9%	128.1	70.0%	15.7	8.6%	24.5	13.4%
251×251	67	203.1	18.5	9.1%	137.3	67.6%	20.6	10.1%	26.7	13.1%
301×301	66	233.2	23.0	9.9%	159.5	68.4%	25.7	11.0%	25.0	10.7%
351×351	65	274.2	30.4	11.1%	185.4	67.6%	33.7	12.3 %	24.6	9.0%

Table 6.4: Average runtimes (per linear system) for GMRES(40) with SAI as a preconditioner. We give time in milli1seconds on the left and the percentage of the total solver time on the right. We also provide runtimes for main parts of the solver: SpMV time (MV), precvec time (PrecV), modified Gram-Schmidt time (GS).

the time of precvec is drastically reduced as expected, now mGS dominates the runtime and accounts for approximately 70% of the total cost. To decrease the runtime further, we need either a solver that does not require mGS or a preconditioner, that works like SpMV, but is either much faster than SAI or more effective than SAI.

Jacobi is another cheap SpMV-like preconditioner. The runtimes shown in Table 6.5 indicate that using the Jacobi preconditioner makes preconditioner-vector multiplication faster than for SAI. However, the number of iterations increases twice compared with GMRES(40) with SAI. Thus, the gain from using a cheap preconditioner is outweighted by the increased cost of mGS. We conclude that we need a solver without mGS.

In our last numerical experiment, we use two-step SAI as a preconditioner. It turns out to be the best preconditioner for GMRES. The number of iterations is comparable to ILUT and BILUT, but precvec with two-step SAI is cheap. For all the grids the matrix-vector multiplication together with preconditioner-vector multiplication is barely more expensive compared with SAI (at most 20% for smallest grid) but the number of iterations is much lower. One might observe that for the grids of size 251×251 and larger, the number of iteration

Grid size	# MVs	GMRES(40) ms	Prec-V ms	Prec-V %	GS ms	GS %	MV ms	MV %	Remaining ms	Remaining %
101×101	180	268.7	17.6	6.5%	213.0	79.3%	20.7	7.7%	17.5	6.5%
151×151	154	245.6	16.5	6.7%	189.8	77.3%	23.5	9.6%	15.7	6.4%
201×201	143	291.7	18.3	6.3%	223.9	76.8%	31.9	10.9%	17.5	6.0%
251×251	137	342.1	20.4	6.0%	260.3	76.1%	42.4	12.4%	19.1	5.6%
301×301	134	410.8	24.0	5.8%	312.6	76.2%	52.7	12.8%	21.5	5.2%
351×351	133	499.1	29.5	5.9%	375.7	75.3%	69.5	13.9%	24.5	4.9%

Table 6.5: Average runtimes (per linear system) for GMRES(40) with Jacobi as a preconditioner. We give time in milli1seconds on the left and the percentage of the total solver time on the right. We also provide runtimes for main parts of the solver: SpMV time (MV), precvec time (PrecV), modified Gram-Schmidt time (GS).

becomes so low that modified Gram-Schmidt is less of a bottleneck, and SpMV plus precvec takes over in terms of being the most costly procedure. Note: the results shown in Table 6.6 were generated on a server equipped with Tesla K20C and Intel i5-2400 CPU with 3.10GHz clock. The results for SpMV and precvec runtimes were measured using CUDA events, and thus, they are not affected by the faster processor; the mGS runtimes were measured using system wall timer, however, the mGS procedure is performed as a sequence of GPU functions (dot products and vector updates), thus we do not expect significant speedup due to change of processor. The change of processor affects the runtimes labeled as "Remaining" but they contribute to only 5% of the total runtime.

Grid size	# MVs	GMRES(40) ms	MV-Prec-V ms	MV-Prec-V %	GS ms	GS %	Remaining ms	Remaining %
101×101	56	61.0	16.4	27.0%	40.8	66.9%	3.8	6.2%
151×151	37	53.5	18.7	35.0%	31.1	58.1%	3.7	6.9%
201×201	31	65.5	26.4	40.3%	34.2	52.2%	4.9	7.5%
251×251	29	80.4	37.2	56.3%	37.0	46.0%	6.2	7.7%
301×301	28	102.6	51.5	50.2%	43.6	42.5%	7.5	7.3%
351×351	28	132.6	69.2	52.2%	52.5	39.6%	10.9	8.2%

Table 6.6: Average runtimes (per linear system) for GMRES(40) with two-step SAI as a preconditioner. We give time in milli1seconds on the left and the percentage of the total solver time on the right. We also provide runtimes for main parts of the solver: combined SpMV-precvec time (MV-PrecV) and modified Gram-Schmidt time (GS).

6.8.2 BiCGStab

In the previous subsection, we showed that preconditioners that are effective on the GPU (SAI, Jacobi, and two-step SAI) are usually not very efficient in reducing the number of iterations. Thus, modified Gram-Schmidt, used in GMRES, becomes a computational bottleneck.

We can eliminate the high cost of modified Gram-Schmidt if we choose a Krylov subspace solver that does not require explicitly making the vectors that span the Krylov subspace orthogonal to each other. Hence, we test BiCGStab. A drawback of using BiCGStab is that it might requires more iterations compared with GMRES. Thus, if the runtime is dominated by precvec, using a solver without mGS is unlikely to improve the runtimes.

Grid size	**# MVs**	**BiCGStab**	**Prec-V**		**MV**		**Other**	
		ms	**ms**	**%**	**ms**	**%**	**ms**	**%**
101×101	23	285.5	256.9	90.0%	2.6	0.9%	26.1	9.1%
151×151	27	447.0	428.8	96.0%	4.0	0.9%	14.2	3.1%
201×201	25	512.2	475.5	92.8%	7.6	1.5%	29.2	5.7%
251×251	25	617.7	575.7	93.2%	7.6	1.2%	34.4	5.6%
301×301	25	739.5	688.7	93.1%	9.6	1.3%	41.2	5.6%
351×351	25	871.3	809.5	92.9%	12.8	1.5%	49.1	5.6%

Table 6.7: Average runtimes (per linear system) for BiCGStab with ILUT as a preconditioner. We give time in milli1seconds on the left and the percentage of the total solver time on the right. We also provide runtimes for main parts of the solver: SpMV time (MV), Preconditioner-vector multiplication time (PrecV).

Table 6.7 shows that using BiCGStab with ILUT is effective (low number of iterations) but not particularly fast (long runtime). As shown in the Table 6.7, more than 90% of time is taken by the preconditioner-vector multiplication. The number of preconditioner-vector multiplications is one less than the number of SpMVs. BiCGStab needs more precvecs than GMRES, and as a result, the runtime has increased. For example, for the 151×151 grid it takes 16.5ms for every preconditioner-vector product. BiCGStab performs three more precvecs than GMRES(40). As a consequence, BiCGStab is not faster than GMRES(40) when using ILUT preconditioner. Therefore, if we decide to use BiCGStab as a solver and we want to reduce the runtime, we need to decrease the number of iterations or use a cheaper preconditioner.

In Table 6.8, we present results combining SAI with BiCGStab. Although BiCGStab with SAI converges in considerably more iterations than GMRES(40), the iterations are cheap. Hence, we see 4× runtimes speedups compared with BiCGStab with ILUT and 6× to 12× speedups (depending on grid size) compared with GMRES(40) with ILUT.

For BiCGStab/SAI, half of the runtime is taken by SpMV and precvec, and the other half of the runtime arises in *Other*. These *other* procedures consist of vector updates, dot products, and arithmetic operations such as multiplication, addition, and square roots. If we want to reduce the runtime even further, we need to decrease the number of iterations or make precvec cheaper (either by using different preconditioner or though more efficient implementation).

Next, we give results for our cheapest preconditioner, the Jacobi preconditioner. The Jacobi preconditioner leads to more solver iterations than SAI, i.e., it results in performing two to three times more SpMVs, precvecs, and *other* operations compared with BiCGStab with SAI. The gain from using a preconditioner with a cheap precvec is thus diminished. Even though

Grid size	# MVs	BiCGStab ms	Prec-V ms	Prec-V %	MV ms	MV %	Other ms	Other %
101×101	107	46.6	12.2	26.3%	12.1	25.9%	22.3	47.8%
151×151	87	46.5	13.0	27.9%	12.9	27.8%	20.6	44.3%
201×201	79	56.6	17.0	30.1%	16.3	28.7%	23.3	41.2%
251×251	79	71.3	23.3	32.7%	21.6	30.3%	26.4	37.0%
301×301	75	86.1	29.3	34.0%	26.5	30.8%	30.3	35.2%
351×351	73	107.6	38.5	35.7%	34.7	32.3%	34.4	32.0%

Table 6.8: Average runtimes (per liner system) for BiCGStab with SAI as a preconditioner. We give time in milli1seconds on the left and the percentage of the total solver time on the right. We also provide runtimes for main parts of the solver: SpMV time (MV), Preconditioner-vector multiplication time (PrecV).

the runtimes are better than for GMRES(40) with any preconditioner, and better than for BiCGStab with ILUT, they are worse than for BiCGStab with SAI.

Grid size	# MVs	BiCGStab ms	Prec-V ms	Prec-V %	MV ms	MV %	Other ms	Other %
101×101	327	135.3	32.1	23.7%	36.7	27.2%	66.5	49.1%
151×151	239	114.1	26.3	23.0%	35.0	30.7%	52.8	46.3%
201×201	207	125.8	28.0	22.2%	42.6	33.9%	55.3	43.9%
251×251	191	148.1	33.3	22.5%	50.4	34.0%	64.4	43.5%
301×301	181	172.9	39.9	23.1%	63.2	36.6%	30.3	40.3%
351×351	175	208.5	47.9	23.0%	80.8	38.8%	79.7	38.2%

Table 6.9: Average runtimes (per linear system) for BiCGStab with Jacobi as a preconditioner. We give time in milli1seconds on the left and the percentage of the total solver time on the right. We also provide runtimes for main parts of the solver: SpMV time (MV), Preconditioner-vector multiplication time (PrecV).

Our last set of results comes from BiCGStab with two-step SAI. Since the cost of matrix-vector product combined with Prec-V is barely higher than for BiCGStab with SAI, and the number of iterations is lower by about one half, this approach leads to the highest reduction in the computation time.

Note that the results shown in Table 6.10 were generated on a server equipped with Tesla K20C and Intel i5-2400 CPU with 3.10GHz clock. The change of processor does not affect the SpMV-precvec time, because this time is measured using CUDA events. However, the change does affect a small part of the *other* procedures (mostly the arithmetic operations).

Grid size	# MVs	BiCGStab ms	MV-Prec-V ms	MV-Prec-V %	Other ms	Other %
101×101	67	29.8	19.5	66.1%	10.1	33.9%
151×151	41	29.7	20.8	70.0%	8.9	30.0%
201×201	35	40.0	30.0	75.0%	10.0	25.0%
251×251	33	54.3	42.3	77.9%	12	22.1%
301×301	31	71.6	57.4	80.2%	14.4	19.8%
351×351	31	94.2	77.4	82.2%	16.8	17.8%

Table 6.10: Average runtimes (per linear system) for BiCGStab with two-step SAI as a preconditioner. We give times as milliseconds on the left and percentage of the total solver time on the right. SpMV performed together with precvec is labeled as MV-Prec-V.

6.8.3 Preconditioner Computation

In the two previous subsections, we focus on the time taken by a Krylov subspace solver accelerated by certain preconditioners but we have not considered the time needed to compute the preconditioner.

The computation of ILUT is a sequential operation. In the CUDA_ITSOL library [59], the incomplete LU factorization is performed entirely on the CPU, and then the result is copied to the GPU. The computation of the BILUT preconditioner requires domain decomposition, which is also computed on the CPU, and then the incomplete LU factorization is computed for every block in the domain in the same manner as for ILUT.

The computation of the SAI preconditioner requires solving an independent small LS problem for every column of the matrix, which is done in parallel on the GPU in our implementation.

The computation of the Jacobi preconditioner is very simple. It requires computing the reciprocals of the diagonal values. All the computations are independent and can be executed concurrently.

For matrices arising in the LDC code, the computation of two-step SAI requires solving n least squares problems with size up to 19×19 (which is done in parallel). This procedure is expected to be slower than the setup of (regular) SAI. The gain resulting from high efficiency and effectiveness of two-step SAI is thus reduced by the relatively expensive setup. However, if the linear systems require a lot of iterations, or if other costs (such as mGS in GMRES) are high, we might be able to make up for the time lost on the expensive setup. Note: the results regarding two-step SAI were generated on a server equipped with Tesla K20C and Intel i5-2400 CPU with 3.10GHz clock.

Table 6.11 and Table 6.12 show the time needed for preconditioner computation paired with solver runtime for different solver/preconditioner choices. When we consider the preconditioner computation time in addition to the solver time, choosing SAI/BiCGStab shows a large advantage.

Grid	GMRES(40)/ILUT		GMRES(40)/SAI		BiCGStab/ILUT		BiCGStab/SAI	
	Setup	Total	Setup	Total	Setup	Total	Setup	Total
	ms	ms	ms	ms	ms	ms	ms	ms
101×101	31.2	292.6	10.5	182.1	31.2	316.8	10.5	57.1
151×151	44.5	465.2	15.9	176.7	44.6	491.6	15.9	62.4
201×201	70.6	556.8	24.0	206.8	70.6	582.8	24.0	80.6
251×251	106.8	685.2	34.3	237.4	106.8	724.4	34.3	105.5
301×301	152.1	841.5	46.5	279.7	152.1	891.6	46.5	132.6
351×351	212.1	1027.9	61.0	335.2	212.1	1083.4	61.0	168.6

Table 6.11: Average runtimes (per linear system) for preconditioner computation (setup) and preconditioner computation plus solver execution (total).

Grid	GMRES(40)/Jacobi		BiCGStab/Jacobi		GMRES(40)/SAI2		BiCGStab/SAI2	
	Setup	Total	Setup	Total	Setup	Total	Setup	Total
	ms	ms	ms	ms	ms	ms	ms	ms
101×101	0.5	269.2	0.5	135.8	25.7	86.7	25.7	55.5
151×151	0.2	245.7	0.2	114.2	41.5	95.0	41.5	71.2
201×201	0.2	291.9	0.2	126.1	66.7	147.1	66.7	106.7
251×251	0.3	342.4	0.3	148.4	100.8	181.2	100.8	155.1
301×301	0.4	411.2	0.4	173.3	144.9	247.5	144.9	216.5
351×351	0.5	499.7	0.5	209.0	194.1	326.7	194.1	288.3

Table 6.12: Average runtimes (per linear system) for preconditioner computation (setup) and preconditioner computation plus solver execution (total). SAI2 = two-step SAI.

6.8.4 Summary

Our results show that the ILUT preconditioner, while usually effective in reducing the number of iterations, is not efficient on the GPU, even if implemented block-wise (BILUT). In contrast, the weaker preconditioners (SAI, Jacobi) suit fine grain parallelism very well. The less effective preconditioners lead to more iterations and thus, to get good performance, this must be balanced by making the preconditioner-vector multiplication very cheap and avoiding additional cost arising from the solver.

Let us recall that the cost of performing Krylov subspace method equals the cost per iteration times the number of iterations. To reduce the runtime, we might decrease the number of iterations or make the cost per iteration cheaper. Finding the optimal choice, which leads to small number of cheap iterations, is different for different problems. What is more, the cost of preconditioner (computation and precvec) and solver on the GPU cannot be determined by simply counting flops, because some preconditioners (Jacobi, SAI) and solvers (BiCGStab) parallelize much better than other (ILUT and BILUT preconditioners, GMRES solver).

The choice of a Krylov solver is arbitrary – often various solvers can be applied to the same linear system. If the linear system requires a lot of iterations to obtain a solution with desired accuracy, it is better to choose a solver with cheap iterations. If the cost is determined by the cost of precvec, and using multiplicative preconditioner (SAI, Jacobi, mSAI) is impractical or impossible, we should use a solver that results in the smallest possible number of iterations (GMRES(m)).

The time needed for preconditioner computation is an important factor in choosing the preconditioner. If computation of a particular preconditioner is very cheap on the GPU, this preconditioner might be a reasonable choice, even if it leads to an increase in the number of iterations (for the LDC problem, this is the case for BiCGStab with Jacobi). We can choose a preconditioner that is expensive to compute in case the preconditioner computation time is balanced by either its high effectiveness (which leads to fast convergence) or its high efficiency (iterations are very cheap), or if we reuse the same preconditioner for multiple linear systems.

For the LDC problem, we find the middle ground between making the iterations cheap and keeping the number of iterations low by choosing BiCGStab as a solver and two-step SAI as a preconditioner. For a different problem, to find a balance between the cost per iteration and the number of iterations, we can perform a similar analysis as for the LDC problem. If we use the GPU to accelerate the computations, we needs to take into account the ease and efficiency of fine grain parallel implementation of the preconditioner (preconditioner computation and precvec) and the solver.

Bibliography

[1] A. M. Abdel-Rehim, R. B. Morgan, D. Nicely, and W. Wilcox. Deflated Hermitian Lanczos methods for multiple right-hand sides. *ArXiv e-prints*, Jan. 2009. Available at https://arxiv.org/abs/0901.3163.

[2] K. Ahuja. Recycling Bi-Lanczos algorithms: BiCG, CGS, and BiCGSTAB. Master's thesis, Virginia Tech, 2009. (Adviser: Eric de Sturler).

[3] K. Ahuja. *Recycling Krylov Subspaces and Preconditioners*. PhD thesis, Virginia Tech, 2011. (Adviser: Eric de Sturler).

[4] K. Ahuja, P. Benner, E. de Sturler, and L. Feng. Recycling BiCGSTAB with an application to parametric model order reduction. *SIAM Journal on Scientific Computing*, 37(5):S429–S446, 2015.

[5] K. Ahuja, E. de Sturler, S. Gugercin, and E. R. Chang. Recycling BiCG with an application to model reduction. *SIAM Journal on Scientific Computing*, 34(4):A1925–A1949, 2012.

[6] F. L. Alvarado and R. Schreiber. Optimal parallel solution of sparse triangular systems. *SIAM Journal on Scientific Computing*, 14(2):446–460, 1993.

[7] A. Amritkar, E. de Sturler, K. Świrydowicz, D. Tafti, and K. Ahuja. Recycling Krylov subspaces for CFD applications and a new hybrid recycling solver. *Journal of Computational Physics*, 303:222–237, 2015.

[8] S. R. Arridge. Topical review: optical tomography in medical imaging. *Inverse Problems*, 15:R41–R93, Apr. 1999.

[9] S. S. Aslan, E. de Sturler, and M. E. Kilmer. Randomized approach to nonlinear inversion combining simultaneous random and optimized sources and detectors, 2017. Available at https://arxiv.org/abs/1706.05586.

[10] J. M. Bahi, R. Couturier, and L. Z. Khodja. Parallel GMRES implementation for solving sparse linear systems on GPU clusters. In *Proceedings of the 19th High Performance Computing Symposia*, pages 12–19, San Diego, CA, USA, 2011. Society for Computer Simulation International.

[11] Z. Bai. Error analysis of the Lanczos algorithm for the nonsymmetric eigenvalue problem. *Mathematics of Computation*, 62(205):209–226, 1994.

[12] Z. Bai, M. Fahey, and G. Golub. Some large-scale matrix computation problems. *Journal of Computational and Applied Mathematics*, 74(1-2):71–89, 1996. TICAM Symposium (Austin, TX, 1995).

[13] A. Basermann. Parallel block ILUT/ILDLT preconditioning for sparse eigenproblems and sparse linear systems. *Numerical Linear Algebra with Applications*, 7(7-8):635–648, 2000.

[14] N. Bell and M. Garland. Implementing sparse matrix-vector multiplication on throughput-oriented processors. In *Proc. Conf. High Performance Computing Networking Storage and Analysis*, pages 1–11, Nov. 2009.

[15] M. P. Bendsøe and O. Sigmund. Topology optimization: theory, methods and applications. 2003.

[16] M. W. Benson and P. O. Frederickson. Iterative solution of large sparse linear systems arising in certain multidimensional approximation problems. *Utilitas Mathematica. A Canadian Journal of Applied Mathematics, Computer Science, and Statistics*, 22:127–140, 1982.

[17] M. Benzi. Preconditioning techniques for large linear systems: a survey. *Journal of Computational Physics*, 182(2):418–477, 2002.

[18] M. Benzi and M. Tuma. A sparse approximate inverse preconditioner for nonsymmetric linear systems. *SIAM Journal on Scientific Computing*, 19(3):968–994, 1998.

[19] M. H. Carpenter, C. Vuik, P. Lucas, M. van Gijzen, and H. Bijl. A general algorithm for reusing Krylov subspace information. *Unsteady Navier-Stokes. NASA/TM*, 2010216190, 2010.

[20] E. Chow. A priori sparsity patterns for parallel sparse approximate inverse preconditioners. *SIAM Journal on Scientific Computing*, 21(5):1804–1822, 2000.

[21] E. Chow and A. Patel. Fine-grained parallel incomplete LU factorization. *SIAM Journal on Scientific Computing*, 37(2):C169–C193, 2015.

[22] E. Chow and Y. Saad. Approximate inverse techniques for block-partitioned matrices. *SIAM Journal on Scientific Computing*, 18(6):1657–1675, 1997.

[23] E. Chow and Y. Saad. Approximate inverse preconditioners via sparse-sparse iterations. *SIAM Journal on Scientific Computing*, 19(3):995–1023, 1998.

[24] R. Couturier and S. Domas. Sparse systems solving on GPUs with GMRES. *The Journal of Supercomputing*, 59(3):1504–1516, 2012.

[25] E. de Sturler. Incomplete block LU preconditioners on slightly overlapping subdomains for a massively parallel computer. *Applied Numerical Mathematics*, 19(1-2):129–146, 1995.

[26] E. de Sturler. Nested Krylov methods based on GCR. *Journal of Computational and Applied Mathematics*, 67(1):15–41, 1996.

[27] E. de Sturler. Truncation strategies for optimal Krylov subspace methods. *SIAM Journal on Numerical Analysis*, 36(3):864–889, 1999.

[28] E. de Sturler, S. Gugercin, M. E. Kilmer, S. Chaturantabut, C. Beattie, and M. O'Connell. Nonlinear parametric inversion using interpolatory model reduction. *SIAM Journal on Scientific Computing*, 37(3):B495–B517, 2015.

[29] E. de Sturler and M. E. Kilmer. A regularized Gauss-Newton trust region approach to imaging in diffuse optical tomography. *SIAM Journal on Scientific Computing*, 33(5):3057–3086, 2011.

[30] E. de Sturler and D. Loher. Parallel iterative solvers for irregular sparse matrices in High Performance Fortran. *Future Generation Computer Systems*, 13(4-5):315–325, 1998.

[31] E. de Sturler and H. A. van der Vorst. Communication cost reduction for Krylov methods on parallel computers. In W. Gentzsch and U. Harms, editors, *High-Performance Computing and Networking, International Conference and Exhibition, HPCN Europe 1994, Munich, Germany, April 18-20, 1994, Proceedings, Volume II: Networking and Tools*, volume 797 of *Lecture Notes in Computer Science*, pages 190–195. Springer, 1994.

[32] E. de Sturler and H. A. van der Vorst. Reducing the effect of global communication in GMRES (m) and CG on parallel distributed memory computers. *Applied Numerical Mathematics*, 18(4):441–459, 1995.

[33] M. M. Dehnavi, D. M. Fernández, J. L. Gaudiot, and D. D. Giannacopoulos. Parallel sparse approximate inverse preconditioning on Graphic Processing Units. *IEEE Transactions on Parallel and Distributed Systems*, 24(9):1852–1862, Sept. 2013.

[34] H. C. Elman. *Iterative methods for large, sparse, nonsymmetric systems of linear equations*. PhD thesis, Yale University New Haven, Conn, 1982. (Advisers: Martin Schultz and Stanley Eisenstat).

[35] M. Embree. How descriptive are GMRES convergence bounds? Technical report, Oxford University Computing Laboratory, 1999.

[36] R. Fletcher. Conjugate gradient methods for indefinite systems. In *Numerical Analysis*, pages 73–89. Springer, 1976.

[37] M. B. Giles and N. A. Pierce. An introduction to the adjoint approach to design. *Flow, Turbulence and Combustion*, 65:393–415, 2000.

[38] G. Golub. *Matrix Computation and the Theory of Moments*, pages 1440–1448. Birkhäuser Basel, Basel, 1995.

[39] G. H. Golub and G. Meurant. Matrices, moments and quadratures. *Pitman Research Notes in Mathematics Series*, 1994.

[40] G. H. Golub and G. Meurant. *Matrices, moments and quadrature with applications*. Princeton University Press, 2009.

[41] G. H. Golub, M. Stoll, and A. Wathen. Approximation of the scattering amplitude and linear systems. *Electronic Transactions on Numerical Analysis*, 31:178–203, 2008.

[42] G. H. Golub and Z. Strakoš. Estimates in quadratic formulas. *Numerical Algorithms*, 8(2):241–268, 1994.

[43] A. Greenbaum. *Iterative Methods for Solving Linear Systems*. Society for Industrial and Applied Mathematics, Philadelphia, PA, 1997.

[44] A. Greenbaum, V. Pták, and Z. Strakoš. Any nonincreasing convergence curve is possible for GMRES. *SIAM Journal on Matrix Analysis and Applications*, 17(3):465–469, 1996.

[45] A. A. Groenwold and L. Etman. On the equivalence of optimality criterion and sequential approximate optimization methods in the classical topology layout problem. *International Journal for Numerical Methods in Engineering*, 73(3):297–316, 2008.

[46] M. J. Grote and T. Huckle. Parallel preconditioning with sparse approximate inverses. *SIAM Journal on Scientific Computing*, 18(3):838–853, 1997.

[47] E. Haber, M. Chung, and F. Herrmann. An effective method for parameter estimation with PDE constraints with multiple right-hand sides. *SIAM Journal on Optimization*, 22(3):739–757, 2012.

[48] F. H. Harlow and J. E. Welch. Numerical calculation of time-dependent viscous incompressible flow of fluid with free surface. *The Physics of Fluids*, 8(12):2182–2189, 1965.

[49] M. R. Hestenes and E. Stiefel. *Methods of conjugate gradients for solving linear systems*, volume 49. NBS, 1952.

[50] J. E. Hicken, M. Osusky, and D. W. Zingg. Comparison of parallel preconditioners for a Newton-Krylov flow solver. In *Computational Fluid Dynamics 2010*, pages 457–463. Springer, 2011.

[51] J. E. Hicken and D. W. Zingg. A simplified and flexible variant of GCROT for solving nonsymmetric linear systems. *SIAM Journal on Scientific Computing*, 32(3):1672–1694, 2010.

[52] R. M. Holland, A. J. Wathen, and G. J. Shaw. Sparse approximate inverses and target matrices. *SIAM Journal on Scientific Computing*, 26(3):1000–1011, 2005.

[53] M. F. Hutchinson. A stochastic estimator of the trace of the influence matrix for Laplacian smoothing splines. *Communications in Statistics. Simulation and Computation*, 19(2):433–450, 1990.

[54] C. W. Jackson and C. J. Roy. A multi-mesh CFD technique for adaptive mesh solutions. In *Proceedings of 53rd AIAA Aerospace Sciences Meeting*, page 1958, 2015.

[55] M. E. Kilmer and E. de Sturler. Recycling subspace information for diffuse optical tomography. *SIAM Journal on Scientific Computing*, 27(6):2140–2166, 2006.

[56] I. B. Labutin and I. V. Surodina. Algorithm for sparse approximate inverse preconditioners in the conjugate gradient method. *Reliable Computing*, 19:121, 2013.

[57] C. Lanczos. Solution of systems of linear equations by minimized-iterations. *Journal of Research of the National Bureau of Standards*, 49:33–53, 1952.

[58] T. M. Leung and D. W. Zingg. Aerodynamic shape optimization of wings using a parallel Newton-Krylov approach. *AIAA journal*, 50(3):540–550, 2012.

[59] R. Li and Y. Saad. GPU-accelerated preconditioned iterative linear solvers. *The Journal of Supercomputing*, 63(2):443–466, Feb 2013.

[60] M. Lukash, K. Rupp, and S. Selberherr. Sparse approximate inverse preconditioners for iterative solvers on GPUs. In *Proceedings of the 2012 Symposium on High Performance Computing*, HPC '12, pages 13:1–13:8, San Diego, CA, USA, 2012. Society for Computer Simulation International.

[61] J. Meijerink and H. A. van der Vorst. An iterative solution method for linear systems of which the coefficient matrix is a symmetric M-matrix. *Mathematics of Computation*, 31(137):148–162, 1977.

[62] J. Meijerink and H. A. van der Vorst. Guidelines for the usage of incomplete decompositions in solving sets of linear equations as they occur in practical problems. *Journal of Computational Physics*, 44(1):134–155, 1981.

[63] G. Meurant and J. D. Tebbens. The role eigenvalues play in forming GMRES residual norms with non-normal matrices. *Numerical Algorithms*, 68(1):143–165, 2015.

[64] K. Mohamed, S. Nadarajah, and M. Paraschivoiu. Krylov recycling techniques for unsteady simulation of turbulent aerodynamic flows. In *Proceedings of 26th International Congress of the Aeronautical Sciences, Anchorage, Alaska, International Council of The Aeronautical Sciences*, 2008.

[65] R. B. Morgan. GMRES with deflated restarting. *SIAM Journal on Scientific Computing*, 24(1):20–37, 2002.

[66] K. Nagendra, D. K. Tafti, and K. Viswanath. A new approach for conjugate heat transfer problems using immersed boundary method for curvilinear grid based solvers. *Journal of Computational Physics*, 267:225–246, 2014.

[67] R. A. Nicolaides. Deflation of conjugate gradients with applications to boundary value problems. *SIAM Journal on Numerical Analysis*, 24:355–365, Apr. 1987.

[68] C. C. Paige and M. A. Saunders. Solution of sparse indefinite systems of linear equations. *SIAM Journal on Numerical Analysis*, 12(4):617–629, 1975.

[69] M. L. Parks, E. de Sturler, G. Mackey, D. D. Johnson, and S. Maiti. Recycling Krylov subspaces for sequences of linear systems. *SIAM Journal on Scientific Computing*, 28(5):1651–1674, 2006.

[70] M. L. Parks, R. Sampath, and P. K. V. V. Nukala. Efficient simulation of large-scale 3D fracture networks via Krylov subspace recycling. Unpublished, 2013.

[71] A. Pothen and F. L. Alvarado. A fast reordering algorithm for parallel sparse triangular solution. *SIAM journal on scientific and statistical computing*, 13(2):645–653, 1992.

[72] A. Ramm. Symmetry properties of scattering amplitudes and applications to inverse problems. *Journal of Mathematical Analysis and Applications*, 156(2):333 – 340, 1991.

[73] J. K. Reid. *Large Sparse Sets of Linear Equations*, chapter On the Method of Conjugate Gradients for the Solution of Large Sparse Linear Equations, pages 231–254. Academic Press, New York, NY, USA, 1971.

[74] C. Roy. Strategies for driving mesh adaptation in CFD. In *Proceedings of 47th AIAA Aerospace Sciences Meeting including The New Horizons Forum and Aerospace Exposition*, page 1302, 2009.

[75] C. Roy. Review of discretization error estimators in scientific computing. In *Proceedings of 48th AIAA Aerospace Sciences Meeting Including the New Horizons Forum and Aerospace Exposition*, page 126, 2010.

[76] Y. Saad. The Lanczos biorthogonalization algorithm and other oblique projection methods for solving large unsymmetric systems. *SIAM Journal on Numerical Analysis*, 19(3):485–506, 1982.

[77] Y. Saad. Krylov subspace methods on supercomputers. *SIAM. Journal on Scientific and Statistical Computing*, 10(6):1200–1232, 1989.

[78] Y. Saad. Highly parallel preconditioners for general sparse matrices. In *Recent Advances in Iterative Methods*, volume 60 of *IMA Vol. Math. Appl.*, pages 165–199. Springer, New York, 1994.

[79] Y. Saad. ILUT: a dual threshold incomplete *LU* factorization. *Numerical Linear Algebra with Applications*, 1(4):387–402, 1994.

[80] Y. Saad. *Iterative methods for sparse linear systems*. SIAM, Philadelphia, Pa., 2nd edition, 2003.

[81] Y. Saad and M. H. Schultz. GMRES: A generalized minimal residual algorithm for solving nonsymmetric linear systems. *SIAM Journal on Scientific and Statistical Computing*, 7(3):856–869, 1986.

[82] Y. Saad and M. H. Schultz. Data communication in parallel architectures. *Parallel Computing. Systems & Applications*, 11(2):131–150, 1989.

[83] Y. Saad, M. Yeung, J. Erhel, and F. Guyomarc'h. A deflated version of the conjugate gradient algorithm. *SIAM Journal on Scientific Computing*, 21(5):1909–1926, 2000.

[84] Y. Saad and J. Zhang. BILUTM: a domain-based multilevel block ILUT preconditioner for general sparse matrices. *SIAM Journal on Matrix Analysis and Applications*, 21(1):279–299, 1999.

[85] P. Sonneveld. CGS, a fast Lanczos-type solver for nonsymmetric linear systems. *SIAM Journal on Scientific and Statistical Computing*, 10(1):36–52, 1989.

[86] A. Stathopoulos and K. Orginos. Computing and deflating eigenvalues while solving multiple right-hand side linear systems with an application to quantum chromodynamics. *SIAM Journal on Scientific Computing*, 32(1):439–462, 2010.

[87] Z. Strakoš. On the real convergence rate of the conjugate gradient method. *Linear Algebra and its Applications*, 154:535–549, 1991.

[88] Z. Strakoš. Model reduction using the vorobyev moment problem. *Numerical Algorithms*, 51(3):363–379, 2009.

[89] Z. Strakoš and P. Tichỳ. On error estimation in the conjugate gradient method and why it works in finite precision computations. *Electronic Transactions on Numerical Analysis*, 13(56-80):8, 2002.

[90] Z. Strakoš and P. Tichý. On efficient numerical approximation of the bilinear form $c^* A^{-1} b$. *SIAM Journal on Scientific Computing*, 33(2):565–587, 2011.

[91] D. Tafti. GenIDLEST: a scalable parallel computational tool for simulating complex turbulent flows. In *Proceedings of the ASME Fluids Engineering Division*, volume 256 of *ASME Publications-FED*, pages 347–356, 2001.

[92] L. N. Trefethen and M. Embree. *Spectra and pseudospectra: the behavior of nonnormal matrices and operators*. Princeton University Press, 2005.

[93] W. C. Tyson, K. Świrydowicz, J. M. Derlaga, C. J. Roy, and E. de Sturler. Improved functional-based error estimation and adaptation without adjoints. In *Proceedings of 46th AIAA Fluid Dynamics Conference*, 2016.

[94] H. A. van der Vorst. Bi-CGSTAB: A fast and smoothly converging variant of Bi-CG for the solution of nonsymmetric linear systems. *SIAM Journal on Scientific and Statistical Computing*, 13(2):631–644, 1992.

[95] H. A. van der Vorst. *Iterative Krylov Methods for Large Linear Systems*. Cambridge University Press, 2003.

[96] J. von Neumann and R. D. Richtmyer. A method for the numerical calculation of hydrodynamic shocks. *Journal of applied physics*, 21(3):232–237, 1950.

[97] G. Wang and D. K. Tafti. Performance enhancement on microprocessors with hierarchical memory systems for solving large sparse linear systems. *International Journal of High Performance Computing Applications*, 13(1):63–79, 1999.

[98] M. Wang, H. Klie, M. Parashar, and H. Sudan. Solving sparse linear systems on NVIDIA Tesla GPUs. In G. Allen, J. Nabrzyski, E. Seidel, G. van Albada, J. Dongarra, and P. Sloot, editors, *Computational Science – ICCS 2009*, volume 5544 of *Lecture Notes in Computer Science*, pages 864–873. Springer Berlin Heidelberg, 2009.

[99] S. Wang, E. de Sturler, and G. H. Paulino. Large-scale topology optimization using preconditioned Krylov subspace methods with recycling. *International Journal for Numerical Methods in Engineering*, 69(12):2441–2468, 2007.

[100] X. Zhang, E. de Sturler, and G. H. Paulino. Stochastic sampling for structural topology optimization with many load cases: Density-based and ground structure approaches. Available at http://arxiv.org/pdf/1609.03099v1, to appear im Computer Methods in Applied Mechanics and Engineering.