Nonlinear Preconditioning Methods for Optimization and Parallel-In-Time Methods for 1D Scalar Hyperbolic Partial Differential Equations

by

Alexander Howse

A thesis
presented to the University of Waterloo
in fulfillment of the
thesis requirement for the degree of
Doctor of Philosophy
in
Applied Mathematics

Waterloo, Ontario, Canada, 2017

© Alexander Howse 2017

Examining Committee Membership

The following served on the Examining Committee for this thesis. The decision of the Examining Committee is by majority vote.

External Examiner:	Xiao-Chuan Cai Professor, Dept. of Computer Science, University of Colorado Boulder
Supervisor:	Hans De Sterck Professor, Dept. of Applied Mathematics, University of Waterloo
Internal Members:	Sander Rhebergen Assistant Professor, Dept. of Applied Mathematics, University of Waterloo
	Stephen Vavasis Professor, Dept. of Applied Mathematics, University of Waterloo
Internal-External Member:	Henry Wolkowicz Professor, Dept. of Combinatorics and Optimization, University of Waterloo

I hereby declare that I am the sole author of this thesis. This is a true copy of the thesis, including any required final revisions, as accepted by my examiners.

I understand that my thesis may be made electronically available to the public.

Abstract

This thesis consists of two main parts, part one addressing problems from nonlinear optimization and part two based on solving systems of time dependent differential equations, with both parts describing strategies for accelerating the convergence of iterative methods.

In part one we present a nonlinear preconditioning framework for use with nonlinear solvers applied to nonlinear optimization problems, motivated by a generalization of linear left preconditioning and linear preconditioning via a change of variables for minimizing quadratic objective functions. In the optimization context nonlinear preconditioning is used to generate a preconditioner direction that either replaces or supplements the gradient vector throughout the optimization algorithm. This framework is used to discuss previously developed nonlinearly preconditioned nonlinear GMRES and nonlinear conjugate gradients (NCG) algorithms, as well as to develop two new nonlinearly preconditioned quasi-Newton methods based on the limited memory Broyden and limited memory BFGS (L-BFGS) updates. We show how all of the above methods can be implemented in a manifold optimization context, with a particular emphasis on Grassmann matrix manifolds.

These methods are compared by solving the optimization problems defining the canonical polyadic (CP) decomposition and Tucker higher order singular value decomposition (HOSVD) for tensors, which are formulated as minimizing approximation error in the Frobenius norm. Both of these decompositions have alternating least squares (ALS) type fixed point iterations derived from their optimization problem definitions. While these ALS type iterations may be slow to converge in practice, they can serve as efficient nonlinear preconditioners for the other optimization methods. As the Tucker HOSVD problem involves orthonormality constraints and lacks unique minimizers, the optimization algorithms are extended from Euclidean space to the manifold setting, where optimization on Grassmann manifolds can resolve both of the issues present in the HOSVD problem.

The nonlinearly preconditioned methods are compared to the ALS type preconditioners and non-preconditioned NCG, L-BFGS, and a trust region algorithm using both synthetic and real life tensor data with varying noise level, the real data arising from applications in computer vision and handwritten digit recognition. Numerical results show that the nonlinearly preconditioned methods offer substantial improvements in terms of time-to-solution and robustness over state-of-the-art methods for large tensors, in cases where there are significant amounts of noise in the data, and when high accuracy results are required.

In part two we apply a multigrid reduction-in-time (MGRIT) algorithm to scalar one-dimensional hyperbolic partial differential equations. This study is motivated by the obser-

vation that sequential time-stepping is an obvious computational bottleneck when attempting to implement highly concurrent algorithms, thus parallel-in-time methods are particularly desirable. Existing parallel-in-time methods have produced significant speedups for parabolic or sufficiently diffusive problems, but can have stability and convergence issues for hyperbolic or advection dominated problems. Being a multigrid method, MGRIT primarily uses temporal coarsening, but spatial coarsening can also be incorporated to produce cheaper multigrid cycles and to ensure stability conditions are satisfied on all levels for explicit time-stepping methods.

We compare convergence results for the linear advection and diffusion equations, which illustrate the increased difficulty associated with solving hyperbolic problems via parallel-in-time methods. A particular issue that we address is the fact that uniform factor-two spatial coarsening may negatively affect the convergence rate for MGRIT, resulting in extremely slow convergence when the wave speed is near zero, even if only locally. This is due to a sort of anisotropy in the nodal connections, with small wave speeds resulting in spatial connections being weaker than temporal connections. Through the use of semi-algebraic mode analysis applied to the combined advection-diffusion equation we illustrate how the norm of the iteration matrix, and hence an upper bound on the rate of convergence, varies for different choices of wave speed, diffusivity coefficient, space-time grid spacing, and the inclusion or exclusion of spatial coarsening.

The use of waveform relaxation multigrid on intermediate, temporally semi-coarsened grids is identified as a potential remedy for the issues introduced by spatial coarsening, with the downside of creating a more intrusive algorithm that cannot be easily combined with existing time-stepping routines for different problems. As a second, less intrusive, alternative we present an adaptive spatial coarsening strategy that prevents the slowdown observed for small local wave speeds, which is applicable for solving the variable coefficient linear advection equation and the inviscid Burgers equation using first-order explicit or implicit time-stepping methods. Serial numerical results show this method offers significant improvements over uniform coarsening and is convergent for inviscid Burgers' equation with and without shocks. Parallel scaling tests indicate that improvements over serial time-stepping strategies are possible when spatial parallelism alone saturates, and that scalability is robust for oscillatory solutions that change on the scale of the grid spacing.

Acknowledgements

First of all, I would like to thank my supervisor, Hans De Sterck, for all of his guidance and support throughout my PhD journey. While his move to Monash University certainly came with its own set of challenges, thanks to his continued dedication we were able to make it work, and it also provided the opportunity for me to make the normally once-in-a-lifetime trip to Australia, twice.

I offer my thanks to the members of my PhD examining committee; Xiao-Chuan Cai, Sander Rhebergen, Stephen Vavasis and Henry Wolkowicz; for taking the time to review the thesis and participating in the defense.

A big thank you to Robert Falgout and Jacob Schroder from Lawrence Livermore National Laboratory, who introduced me to the topic of multigrid reduction in time, and who were an invaluable source of advice on writing and debugging code using the XBraid and Hypre software packages.

I would also like to thank Ronald Haynes for serving as my Masters' supervisor while at Memorial University of Newfoundland, which helped set me on the path that I now follow; and Scott MacLachlan, also at Memorial University, for all of his help with multigrid reduction, and for providing a good excuse for a trip back home.

To my parents, Joan and Maxwell Howse, thank you for your love and encouragement every step along the way of this 24 year journey from Kindergarten to now.

Finally, thank you to my wife, Samantha, for being there for me during all the ups and downs.

Dedication

To Samantha, for all her dedication

Table of Contents

List of Tables

List of Figures

Introduction

This thesis is divided into two parts, with each part addressing a different problem in computational mathematics. Part I, consisting of Chapters 1–3, considers the use of nonlinear preconditioners for solving nonlinear optimization problems. Part II, consisting of Chapters 4–6, discusses the use of a multigrid based parallel-in-time method for the solution of hyperbolic partial differential equations.

Part I

The use of linear preconditioning techniques in the iterative solution of large linear systems and related matrix problems is ubiquitous in the literature [12], and for good reason: many problems are ill-conditioned and standard iterations either fail to converge entirely, or fail to converge in a reasonable amount of time. Similarly, nonlinear preconditioning strategies can be used to drastically improve the convergence of iterations solving nonlinear systems $\mathbf{g}(\mathbf{x}) = \mathbf{0}$, such as those arising from discretized partial differential equations [18]. In Part I of this thesis we discuss how nonlinear preconditioning can be applied to optimization algorithms and show how these preconditioned algorithms can be adapted for optimization over matrix manifolds. A nonlinear preconditioner in the optimization context is a fully nonlinear transformation applied to the gradient equations, as opposed to linear preconditioners that involve linear transformations encoded by matrix multiplications.

We consider two approaches for applying nonlinear preconditioning to optimization methods for solving the nonlinear optimization problem

$$\min \quad f(\mathbf{x}) \tag{1}$$

in manners that significantly speed up the convergence. Both of these approaches require supplementing or replacing gradient information with vectors generated by a preconditioner

function. The proposed approaches can be situated in the context of the recent renaissance in research on nonlinearly preconditioned nonlinear solvers [18]. Some of the ideas on nonlinear preconditioning date back as far as the 1960s and 1970s [6, 10, 27], but they remain underexplored in theory and in practice [18], especially in comparison to linear preconditioning.

The first approach is based on the general idea of left preconditioning, which is commonly used when solving linear systems, and has been generalized to the case of nonlinear preconditioners for nonlinear systems of equations, see, e.g., the review paper [18]. We refer to this approach as the LP approach (for Left Preconditioning). The second approach is inspired by a linear change of variables $\mathbf{x} = \mathbf{C}\mathbf{z}$ in optimization problem (1). It is derived by applying the optimization method to $\widehat{f}(\mathbf{z}) = f(\mathbf{C}\mathbf{z})$, and transforming back to the $\mathbf{x}$ variables. The resulting linearly preconditioned methods are well-known in the optimization community [54, 72], but we generalize this approach to the case of fully nonlinear preconditioners. We refer to this approach as the TP approach (for Transformation Preconditioning). We first illustrate the main ideas in the simplified context of linear preconditioning for convex quadratic optimization problems that correspond to solving symmetric positive definite linear systems. This allows us to explain and interpret the LP and TP approaches in relation to well-known ideas for linear preconditioning. We then extend the formalism to nonlinear preconditioning for general nonlinear objective functions $f(\mathbf{x})$, and provide extensive numerical tests illustrating and comparing the merits of the two approaches.

In part I of this thesis we apply nonlinear preconditioning strategies to three different optimization methods: (i) nonlinear GMRES (NGMRES) [33, 34, 105] (ii) nonlinear conjugate gradients (NCG) [77], and (iii) Broyden-Fletcher-Goldfarb-Shanno (L-BFGS) and limited memory Broyden (L-Broyden) quasi-Newton (QN) methods [20, 35]. NGMRES is an acceleration technique wherein a linear combination of past approximations is optimized to produce an improved search direction. It has been presented in different forms under different names, such as Anderson acceleration [6, 45, 104] and the direct inversion in the iterative subspace method [81], and is equivalent to an inverse projected Broyden secant method iteration [45]. The name NGMRES arose because it can be shown to be a generalization of GMRES for linear systems [86], see, e.g., [104, 105]. Ideas of introducing a nonlinear preconditioner into NCG have been considered as early as the 1970s [10, 27, 73], but have not been widely explored. Both the LP and TP approaches were explored implicitly in previous work on NCG [34], and we now present a framework that explicitly formalizes the LP and TP approaches for optimization, and apply this framework to nonlinearly precondition the L-BFGS and L-Broyden methods, the most successful QN methods in the optimization and nonlinear system contexts, respectively.

To demonstrate the efficacy of the proposed nonlinearly preconditioned algorithms we compare them in the context of solving approximate tensor decomposition problems: decomposition of a multidimensional array into a sum and/or product of multiple components (for example, vector outer products) for the purpose of reducing storage costs and carrying out data analysis. In the era of big data, tensors and tensor decompositions are commonly used when vast quantities of data are collected and need to be organized, stored, and analyzed, such as in chemometrics [92], data mining [65], food science [17], numerical linear algebra [9], numerical methods for elliptic partial differential equations (PDEs) [52], pattern and image recognition [88, 110], and signal processing [26]. Many more examples are given in [64]. Such problems are typically cast as minimizing the approximation error in a given norm. Specifically, we consider decompositions into canonical polyadic (CP) and Tucker tensor formats, the former representing a tensor as a sum of rank-one terms and the latter as a multilinear tensor-matrix product. Both of these tensor formats have standard iterative algorithms for computing decompositions, which can be slow to converge when used independently, but can be useful as nonlinear preconditioners, resulting in significant acceleration.

Nonlinearly preconditioned optimization algorithms have previously been developed for tensor decomposition problems. Nonlinearly preconditioned NGMRES [32] and NCG [34] methods have been successfully used to accelerate iterations for the CP decomposition problem, where they are among the fastest methods available for noisy problems and when high accuracy is required. To these methods we add the L-BFGS and L-Broyden nonlinearly preconditioned QN methods and show that these methods can offer even further improvements. Many other approaches exist for computing a CP decomposition [64, 97], including optimization approaches such as standard NCG [5].

We extend these nonlinearly preconditioned algorithms to decompose test tensors into a particular subtype of Tucker tensor called the higher order singular value decomposition (HOSVD), which introduces the challenges of large sets of matrix equality constraints and lack of uniqueness due to non-isolated minima. These challenges can most easily be handled by adapting these algorithms to use matrix manifold optimization techniques and then optimize over a product of Grassmann manifolds, which describe equivalence classes of matrices with orthonormal columns. Matrix manifold optimization for approximate Tucker decompositions has been considered before: past publications have adapted a number of popular optimization techniques to the manifold setting. Newton's method was considered by Eldén and Savas in [40], which was followed by adaptation of BFGS and L-BFGS by Savas and Lim [89]. Manifold NCG [60], a Riemannian trust-region method incorporating truncated conjugate gradient [61], and a differential-geometric Newton's method [62] in the Grassmann framework have been developed by Ishteva et al.

Part I of this thesis is organized as follows. Chapter 1 contains (i) necessary tensor definitions and operators, (ii) a description of tensor decomposition methods, (iii) a description of the optimization methods used, and (iv) an introduction to matrix manifold optimization. Chapter 2 introduces left preconditioning and transformation preconditioning in general and then describes the preconditioned GMRES, CG, and QN methods, first for linear problems, then for nonlinear problems, and finally for nonlinear problems defined over matrix manifolds. Chapter 3 provides implementation details and numerical results comparing nonlinearly preconditioned NGMRES, NCG, and QN methods over Euclidean space by using the CP tensor decomposition problem and their matrix manifold variants by using the Tucker HOSVD problem.

Part II

Due to stagnating processor speeds and increasing core counts, the current paradigm of high performance computing is to achieve shorter computing times by increasing the concurrency of computations. Time integration represents an obvious bottleneck for achieving greater speedup due to the sequential nature of many time integration schemes. While temporal parallelism may seem counter-intuitive, the development of parallel-in-time methods is an active area of research, with a history spanning several decades [48]. Variants include direct methods and iterative methods based on deferred corrections [41], domain decomposition [51], multigrid [58], multiple shooting [22], and waveform relaxation [101] approaches. For instance, one of the most prominent parallel-in-time methods, parareal [71], is equivalent to a two-level multigrid scheme [50]. These methods have resulted in significant speedup for parabolic equations, or equations with significant diffusivity, but have had markedly less success when the problem is hyperbolic or advection dominated [83].

In part II of this thesis we discuss the multigrid reduction-in-time (MGRIT) method [42] and use XBraid [2], an open-source implementation of MGRIT. A strength of the MGRIT framework is its non-intrusive nature, which allows existing time-stepping routines to be used within the MGRIT implementation. Thus far, MGRIT has been successfully implemented using time-stepping routines for linear [42] and nonlinear [44] parabolic PDEs in multiple dimensions, Navier-Stokes fluid dynamics problems [43], and power system models [66]. We now consider applying MGRIT to the conservative hyperbolic PDE

$$\partial_t u + \partial_x(f(u,x,t)) = 0,$$

for the cases of variable coefficient linear advection, $f(u,x,t) = a(x,t)u$, and the inviscid Burgers equation, $f(u,x,t) = \frac{1}{2}u^2$.

As a multigrid method, MGRIT primarily involves temporal coarsening, but spatial coarsening is a suitable approach for explicit time integration to ensure that stability conditions are satisfied on all levels of the grid hierarchy. Spatial coarsening may also be used with implicit time integration to produce smaller coarse-grid problems and, hence, cheaper multigrid cycles. However, it has been observed that spatial coarsening can result in significant deterioration in the rate of convergence. Furthermore, small local Courant numbers induce a sort of anisotropy in the discrete equations, meaning that the nodal connections in space are small compared to those in time. These so-called weak connections prevent pointwise relaxation from smoothing the error in space, thus inhibiting the effectiveness of spatial coarsening and leading to slow convergence. We investigate this phenomenon through the use of numerical tests and a technique known as semi-algebraic mode analysis (SAMA) [46], a generalized variant of Fourier analysis with improved predictive performance for methods applied to time evolution problems.

Two methods for improving the results of MGRIT with spatial coarsening have been identified. The first is the introduction of spatial relaxations, inspired by waveform relaxation multigrid (WRMG), on the intermediate grid (coarsening in time but not in space). While this almost entirely ameliorates the negative side-effects of using spatial coarsening, it results in a more intrusive algorithm that may not be easily combined with existing time-stepping methods. The second approach, which prevents the extremely poor convergence observed for small wave speeds, is an adaptive spatial coarsening strategy that locally prevents coarsening in regions with near zero Courant numbers. This adaptive coarsening strategy is compatible with pre-existing time-steppers (that are compatible with non-uniform spatial grids), and parallel scaling tests show that significant speedup is possible by using MGRIT with adaptive coarsening.

Part II of this thesis is organized as follows. Chapter 4 covers the derivation and implementation of the MGRIT parallel-in-time method, how it can be combined with spatial coarsening, and a description of the hyperbolic PDE of interest. Chapter 5 investigates the performance of MGRIT with and without spatial coarsening when applied to the linear advection equation, analysis of the approach via the SAMA technique, and a description of how to combine MGRIT with WRMG, along with corroborating numerical results. Chapter 6 motivates and describes the proposed adaptive spatial coarsening method for variable coefficient linear advection and the inviscid Burgers equation, illustrating the effectiveness via serial numerical results, as well as illustrating the potential for speedup when spatial parallelism alone saturates via parallel scaling tests.

Part I

Nonlinear Preconditioning for Optimization

Chapter 1

Preliminaries and Motivating Application

This chapter serves as an introduction to (i) required tensor definitions and operations, (ii) Canonical Polyadic (CP) and Tucker tensor decompositions, (iii) the optimization methods considered, and (iv) the concept of matrix manifold optimization, which will be used throughout the following two chapters.

1.1 Tensors

A *tensor* is a multidimensional array[1], and the number of *dimensions* (*modes*) of a tensor is called the tensor *order*. Order-1 tensors are vectors and order-2 tensors are matrices. For consistency, unless otherwise indicated vectors are indicated by bold lowercase letters ($\mathbf{x}$), matrices by bold uppercase letters ($\mathbf{X}$), and tensors by Euler script letters ($\mathcal{X}$). Tensor elements are indicated by subindices or bracketed arguments: $\mathcal{X}_{ijk} = \mathcal{X}(i, j, k)$. Tensors are useful when large quantities of data need to be organized and analyzed, because each dimension can represent a parameter and each element can represent an observation for a particular parameter combination. As a result tensors have seen widespread use in areas such as in chemometrics [92], data mining [65], food science [17], pattern and image recognition [88, 110], and signal processing [26].

[1] More correctly, a tensor is an element of a tensor product space [57, 70]. The coordinate representation of a tensor, which is dependent on the choice of basis, is a hypermatrix. A hypermatrix is a multidimensional array with well defined algebraic operations arising from the structure of the tensor product space. For simplicity we follow the convention of referring to representative hypermatrices as tensors.

1.1.1 Tensor Matricizations and Definitions of Rank

Tensor *fibers* are obtained by fixing all indices but one. Mode-n fibers are obtained by fixing all indices but the n^{th}. The *mode-n matricization* of $\mathcal{X}$, denoted $\mathbf{X}_{(n)}$, has the mode-n fibers of $\mathcal{X}$ as its columns. So long as the ordering of fibers is consistent throughout calculations, the specific ordering used is unimportant in many applications. A *rank-one tensor* $\mathcal{X} \in \mathbb{R}^{I_1 \times \cdots \times I_N}$ is the outer product

$$\mathcal{X} = \mathbf{a}^{(1)} \circ \mathbf{a}^{(2)} \circ \cdots \circ \mathbf{a}^{(N)}$$

where $\mathbf{a}^{(n)} \in \mathbb{R}^{I_n}$ for $n = 1, \ldots, N$ and

$$\mathcal{X}_{i_i i_2 \ldots i_N} = a^{(1)}_{i_1} a^{(2)}_{i_2} \cdots a^{(N)}_{i_N} \quad \text{for all } 1 \leq i_n \leq I_n.$$

The *rank* of $\mathcal{X}$, denoted $\text{rank}(\mathcal{X})$, is the minimum number of rank-one tensors required to express $\mathcal{X}$ as a linear combination [64]. The *n-rank* of $\mathcal{X}$ is the dimension of the space spanned by the mode-n fibers: $\text{rank}_n(\mathcal{X}) = \dim(\text{Col}(\mathbf{X}_{(n)}))$ [30]. The *multilinear rank* of $\mathcal{X}$ is the N-tuple $(\text{rank}_1(\mathcal{X}), \ldots, \text{rank}_N(\mathcal{X}))$.

1.1.2 Tensor and Matrix Products

The *mode-n contravariant product* of $\mathcal{X} \in \mathbb{R}^{I_1 \times \cdots \times I_N}$ and $\mathbf{A} \in \mathbb{R}^{J \times I_n}$ is $\mathcal{Y} = (\mathbf{A})_n \cdot \mathcal{X}$ [40]:

$$\mathcal{Y}(i_1, \ldots, i_{n-1}, j, i_{n+1}, \ldots, i_n) = \sum_{i_n=1}^{I_n} \mathbf{A}(j, i_n) \mathcal{X}(i_1, \ldots, i_N).$$

Each mode-n fiber of $\mathcal{X}$ is multiplied by each row of $\mathbf{A}$: $\mathbf{Y}_{(n)} = \mathbf{A}\mathbf{X}_{(n)}$. It follows that $(\mathbf{B})_n \cdot ((\mathbf{A})_n \cdot \mathcal{X}) = (\mathbf{B}\mathbf{A})_n \cdot \mathcal{X}$, and that multiplication in different modes is commutative. The *mode-n covariant product* of $\mathcal{X}$ and $\mathbf{A} \in \mathbb{R}^{I_n \times J}$ is $\mathcal{Y} = \mathcal{X} \cdot (\mathbf{A})_n$ [40]:

$$\mathcal{Y}(i_1, \ldots, i_{n-1}, j, i_{n+1}, \ldots, i_n) = \sum_{i_n=1}^{I_n} \mathcal{X}(i_1, \ldots, i_N) \mathbf{A}(i_n, j).$$

Clearly, $(\mathbf{A}^\intercal)_n \cdot \mathcal{X} = \mathcal{X} \cdot (\mathbf{A})_n$.

The *outer product* of an m-way tensor and an n-way tensor is an $(m+n)$-way tensor. To illustrate, the outer product of $\mathcal{X} \in \mathbb{R}^{J \times K \times L}$ and $\mathcal{Y} \in \mathbb{R}^{M \times N}$ is $\mathcal{Z} = \mathcal{X} \otimes \mathcal{Y} \in \mathbb{R}^{J \times K \times L \times M \times N}$, where $\mathcal{Z}_{jklmn} = \mathcal{X}_{jkl} \mathcal{Y}_{mn}$. The *inner product* of $\mathcal{X}$ and $\mathcal{Y} \in \mathbb{R}^{I_1 \times \cdots \times I_N}$ is

$$\langle \mathcal{X}, \mathcal{Y} \rangle = \sum_{i_1=1}^{I_1} \cdots \sum_{i_N=1}^{I_N} \mathcal{X}(i_1, , \ldots, i_N) \mathcal{Y}(i_1, \ldots, i_N).$$

The tensor Frobenius norm is $\|\mathcal{X}\|_F = \sqrt{\langle \mathcal{X}, \mathcal{X} \rangle}$, and $\|\mathcal{X}\|_F = \left\|\mathbf{X}_{(k)}\right\|_F$ for $k = 1, \ldots, N$. It is also invariant under orthogonal transformations $\mathbf{A}^{(n)}$:

$$\|\mathcal{X}\|_F = \left\|(\mathbf{A}^{(1)}, \ldots, \mathbf{A}^{(N)}) \cdot \mathcal{X}\right\|_F .$$

A *contracted product* is an outer product followed by contractions along specified pairs of modes, and requires input tensors to have at least one mode of common length. The contracted product of $\mathcal{X}$ and $\mathcal{Y}$ is denoted $\langle \mathcal{X}, \mathcal{Y} \rangle_{I;J}$, contracting over the $\mathcal{X}$ modes I and $\mathcal{Y}$ modes J. If $\mathcal{X}$ and $\mathcal{Y}$ are of equal size and corresponding modes are paired, only one subscript is used. Negative subscripts, $\langle \mathcal{X}, \mathcal{Y} \rangle_{-J}$, indicate a contracted product over all modes except those listed in J. To illustrate, given equal sized 3-way tensors $\mathcal{A}$ and $\mathcal{B}$, three possible contracted products are: $\mathcal{C} = \langle \mathcal{A}, \mathcal{B} \rangle_1$, $\mathbf{D} = \langle \mathcal{A}, \mathcal{B} \rangle_{1:2}$, and $e = \langle \mathcal{A}, \mathcal{B} \rangle_{1:3}$:

$$\mathcal{C}_{jklm} = \sum_{\lambda} \mathcal{A}_{\lambda jk} \mathcal{B}_{\lambda lm}, \quad \mathbf{D}_{jk} = \sum_{\lambda,\mu} \mathcal{A}_{\lambda \mu j} \mathcal{B}_{\lambda \mu k}, \quad \text{and} \quad e = \sum_{\lambda,\mu,\nu} \mathcal{A}_{\lambda \mu \nu} \mathcal{B}_{\lambda \mu \nu}.$$

The *Hadamard (element-wise) product* of equal sized tensors $\mathcal{X}$ and $\mathcal{Y}$ is denoted $\mathcal{X} * \mathcal{Y}$. The *Kronecker* product of $\mathbf{A} \in \mathbb{R}^{I \times J}$ and $\mathbf{B} \in \mathbb{R}^{K \times L}$ is denoted by $\mathbf{A} \otimes \mathbf{B} \in \mathbb{R}^{(IK) \times (JL)}$, and the *Khatri-Rao product* of $\mathbf{C} \in \mathbb{R}^{I \times K}$ and $\mathbf{D} \in \mathbb{R}^{J \times K}$ is denoted by $\mathbf{C} \odot \mathbf{D} \in \mathbb{R}^{(IJ) \times K}$:

$$\mathbf{A} \otimes \mathbf{B} = \begin{bmatrix} a_{11}\mathbf{B} & a_{12}\mathbf{B} & \cdots & a_{1J}\mathbf{B} \\ a_{21}\mathbf{B} & a_{22}\mathbf{B} & \cdots & a_{2J}\mathbf{B} \\ \vdots & \vdots & \ddots & \vdots \\ a_{I1}\mathbf{B} & a_{I2}\mathbf{B} & \cdots & a_{IJ}\mathbf{B} \end{bmatrix}, \quad \mathbf{C} \odot \mathbf{D} = \begin{bmatrix} \mathbf{c}_1 \otimes \mathbf{d}_1 & \mathbf{c}_2 \otimes \mathbf{d}_2 & \cdots & \mathbf{c}_K \otimes \mathbf{d}_K \end{bmatrix}.$$

These products are useful when computing tensor decompositions and matricizing tensor-matrix products. For instance, using the (natural) fiber ordering of [40] we see that $\mathcal{Y} = (\mathbf{A}^{(1)}, \ldots, \mathbf{A}^{(N)}) \cdot \mathcal{X}$ if and only if

$$\mathbf{Y}_{(n)} = \mathbf{A}^{(n)} \mathbf{X}_{(n)} \left(\mathbf{A}^{(1)} \otimes \cdots \otimes \mathbf{A}^{(n-1)} \otimes \mathbf{A}^{(n+1)} \otimes \cdots \otimes \mathbf{A}^{(N)}\right)^{\mathsf{T}}.$$

The ordering of fibers in matricization is important here: different orderings produce different orderings in the Kronecker products.

1.2 Tensor Decompositions

Tensor decompositions express tensors as sums or products of several components with the goal of simplifying further work involving the tensor data. *Tensor approximation*

problems involve seeking the best approximation of a tensor $\mathcal{X}$ by a tensor $\widehat{\mathcal{X}}$, commonly having a specified decomposition, the components of which are determined by minimizing $\|\mathcal{X} - \widehat{\mathcal{X}}\|$. The remainder of this section provides an introduction to the CP and Tucker tensor decompositions, including all necessary operations and definitions, which motivate the nonlinearly preconditioned optimization methods developed and used in the remainder of part I.

1.2.1 Matrix Singular Value Decomposition (SVD)

A matrix $\mathbf{M} \in \mathbb{R}^{m\times n}$ has SVD $\mathbf{M} = \mathbf{U}\mathbf{\Sigma}\mathbf{V}^\intercal$, where $\mathbf{U} \in \mathbb{R}^{m\times m}$, $\mathbf{V} \in \mathbb{R}^{n\times n}$ are orthogonal and $\mathbf{\Sigma} \in \mathbb{R}^{m\times n}$ is diagonal with nonnegative real entries in decreasing order. The nonzero entries of $\mathbf{\Sigma}$ are the *singular values* of $\mathbf{M}$ and the columns of $\mathbf{U}$ ($\mathbf{V}$) are the *left-singular* (*right-singular*) vectors. The *rank* of $\mathbf{M}$ is equal to the number of singular values, and by the Eckhart-Young theorem, the best rank-r approximation of $\mathbf{M}$ in the Frobenius norm is obtained by keeping the largest r singular values, setting the rest to zero [38].

1.2.2 CP Decomposition

The CP decomposition, also known by the names CANDECOMP (canonical decomposition) and PARAFAC (parallel factors), decomposes a tensor into a sum of rank-one tensors [64]. The rank-R CP decomposition of $\mathcal{X} \in \mathbb{R}^{I_1\times\cdots\times I_N}$ is [64]

$$[\![\mathbf{A}^{(1)}, \mathbf{A}^{(2)}, \ldots, \mathbf{A}^{(N)}]\!] \equiv \sum_{r=1}^{R} \mathbf{a}_r^{(1)} \circ \mathbf{a}_r^{(2)} \circ \cdots \circ \mathbf{a}_r^{(N)}, \tag{1.1}$$

where $\mathbf{A}^{(n)} \in \mathbb{R}^{I_n\times R}$ for $n = 1, \ldots, N$. To compute a CP decomposition we solve

$$\min_{\{\mathbf{A}^{(n)}\}} \quad \tfrac{1}{2}\left\|\mathcal{X} - [\![\mathbf{A}^{(1)}, \mathbf{A}^{(2)}, \ldots, \mathbf{A}^{(N)}]\!]\right\|_F^2. \tag{1.2}$$

The standard approach for solving (1.2) is an alternating least squares (ALS) type iteration [21,55], reproduced here as Algorithm 1. The mode-n matricization of a CP tensor is given by

$$\mathbf{A}^{(n)}\left(\mathbf{A}^{(N)} \odot \cdots \odot \mathbf{A}^{(n+1)} \odot \mathbf{A}^{(n-1)} \odot \cdots \odot \mathbf{A}^{(1)}\right)^\intercal.$$

By fixing all matrices except $\mathbf{A}^{(n)}$ the problem becomes

$$\min_{\mathbf{A}^{(n)}} \quad \tfrac{1}{2}\left\|\mathbf{X}_{(n)} - \mathbf{A}^{(n)}(\mathbf{A}^{(N)} \odot \cdots \odot \mathbf{A}^{(n+1)} \odot \mathbf{A}^{(n-1)} \odot \cdots \odot \mathbf{A}^{(1)})^\intercal\right\|_F^2, \tag{1.3}$$

a linear least squares problem with exact solution

$$\mathbf{A}^{(n)} = \mathbf{X}_{(n)} \left(\left(\mathbf{A}^{(N)} \odot \cdots \odot \mathbf{A}^{(n+1)} \odot \mathbf{A}^{(n-1)} \odot \cdots \odot \mathbf{A}^{(1)} \right)^{\intercal} \right)^{\dagger},$$

where $\dagger$ denotes the Moore-Penrose pseudoinverse [74, 78]. Since the pseudoinverse of a Khatri-Rao product satisfies the identity [64]

$$(\mathbf{A} \odot \mathbf{B})^{\dagger} = \left((\mathbf{A}^{\intercal}\mathbf{A}) * (\mathbf{B}^{\intercal}\mathbf{B}) \right) (\mathbf{A} \odot \mathbf{B})^{\intercal},$$

this exact solution is typically implemented as

$$\mathbf{A}^{(n)} = \mathbf{X}_{(n)} \left(\mathbf{A}^{(N)} \odot \cdots \odot \mathbf{A}^{(n+1)} \odot \mathbf{A}^{(n-1)} \odot \cdots \odot \mathbf{A}^{(1)} \right)^{\intercal} \left(\mathbf{\Gamma}^{(n)} \right)^{\dagger},$$

where

$$\mathbf{\Gamma}^{(n)} = \left(\mathbf{A}^{(1)\intercal}\mathbf{A}^{(1)} \right) * \cdots * \left(\mathbf{A}^{(n-1)\intercal}\mathbf{A}^{(n-1)} \right) * \left(\mathbf{A}^{(n+1)\intercal}\mathbf{A}^{(n+1)} \right) * \cdots * \left(\mathbf{A}^{(N)\intercal}\mathbf{A}^{(N)} \right)$$

for $n = 1, \ldots, N$.

Algorithm 1 CP-ALS

procedure CP-ALS($\mathcal{X}, \mathbf{A}^{(1)}, \ldots, \mathbf{A}^{(N)}$)
  for $n = 1, \ldots, N$ **do**
    $\mathbf{\Gamma}^{(n)} = \left(\mathbf{A}^{(1)\intercal}\mathbf{A}^{(1)} \right) * \cdots * \left(\mathbf{A}^{(n-1)\intercal}\mathbf{A}^{(n-1)} \right) * \left(\mathbf{A}^{(n+1)\intercal}\mathbf{A}^{(n+1)} \right) * \cdots * \left(\mathbf{A}^{(N)\intercal}\mathbf{A}^{(N)} \right)$
    $\mathbf{A}^{(n)} = \mathbf{X}_{(n)} \left(\mathbf{A}^{(N)} \odot \cdots \odot \mathbf{A}^{(n+1)} \odot \mathbf{A}^{(n-1)} \odot \cdots \odot \mathbf{A}^{(1)} \right)^{\intercal} \left(\mathbf{\Gamma}^{(n)} \right)^{\dagger}$
  end for
  return $\mathbf{A}^{(1)}, \ldots, \mathbf{A}^{(N)}$
end procedure

The CP-ALS iteration can be slow to converge in practice, thus alternative optimization algorithms are desirable. Most optimization algorithms require the gradient of

$$f(\mathbf{x}) = \tfrac{1}{2} \left\| \mathcal{X} - [\![\mathbf{A}^{(1)}, \mathbf{A}^{(2)}, \ldots, \mathbf{A}^{(N)}]\!] \right\|_F^2,$$

where $\mathbf{x} = (\mathbf{A}^{(1)}, \mathbf{A}^{(2)}, \ldots, \mathbf{A}^{(N)})$ is the N-tuple of factor matrices. The partial derivative of f with respect to $\mathbf{A}^{(n)}$ is [5]

$$\frac{\partial f}{\partial \mathbf{A}^{(n)}} = -\mathbf{X}_{(n)} \left(\mathbf{A}^{(N)} \odot \cdots \odot \mathbf{A}^{(n+1)} \odot \mathbf{A}^{(n-1)} \odot \cdots \odot \mathbf{A}^{(1)} \right) + \mathbf{A}^{(n)}\mathbf{\Gamma}^{(n)}.$$

Note that setting the gradient of f equal to zero gives

$$\mathbf{A}^{(n)}\mathbf{\Gamma}^{(n)} = \mathbf{X}_{(n)} \left(\mathbf{A}^{(N)} \odot \cdots \odot \mathbf{A}^{(n+1)} \odot \mathbf{A}^{(n-1)} \odot \cdots \odot \mathbf{A}^{(1)} \right),$$

from which the CP-ALS iteration immediately follows.

1.2.3 The Tucker Format and the Higher Order SVD (HOSVD)

The Tucker format was introduced by Tucker in 1963 for 3-mode tensors [99], and has since been extended to N-mode tensors; see, for example, [30,100]. A tensor $\mathcal{X} \in \mathbb{R}^{I_1 \times \cdots \times I_N}$ is expressed in Tucker format as $(\mathbf{A}^{(1)}, \ldots, \mathbf{A}^{(N)}) \cdot \mathcal{S}$, where $\mathcal{S} \in \mathbb{R}^{R_1 \times \cdots \times R_N}$ and $\mathbf{A}^{(n)} \in \mathbb{R}^{I_n \times R_n}$. We must have $R_n \leq I_n$, and in practice often have $R_n \ll I_n$, resulting in a significant reduction in storage. If $R_n \geq \text{rank}_n(\mathcal{X})$ for all n, the decomposition is exact. If $R_n < \text{rank}_n(\mathcal{X})$ for some n, this is an approximate Tucker decomposition (ATD). Tucker decompositions are not unique: replacing $\mathcal{S}$ by $(\mathbf{B})_n \cdot \mathcal{S}$ and $\mathbf{A}^{(n)}$ by $\mathbf{A}^{(n)}\mathbf{B}^{-1}$ produces an equivalent tensor.

In [30] the authors introduce a Tucker decomposition called the HOSVD and prove all tensors have such a decomposition. The HOSVD of $\mathcal{X} \in \mathbb{R}^{I_1 \times \cdots \times I_N}$ is

$$\mathcal{X} = (\mathbf{A}^{(1)}, \ldots, \mathbf{A}^{(N)}) \cdot \mathcal{S},$$

where $\mathcal{S} \in \mathbb{R}^{I_1 \times \cdots \times I_N}$, each $\mathbf{A}^{(n)} \in \mathbb{R}^{I_n \times I_n}$ is orthogonal, and $\mathcal{S}$ satisfies

(i) all-orthogonality: for all possible n, α, and β, $\alpha \neq \beta$: $\langle \mathcal{S}_{i_n=\alpha}, \mathcal{S}_{i_n=\beta} \rangle = 0$;

(ii) the ordering: $\|\mathcal{S}_{i_n=1}\|_F \geq \|\mathcal{S}_{i_n=2}\|_F \geq \cdots \geq \|\mathcal{S}_{i_n=I_n}\|_F \geq 0$ for all n.

Given a target multilinear rank $(R_1, \ldots, R_N)$, a truncated HOSVD may be computed in which $\mathbf{A}^{(n)}$ contains only R_n orthonormal columns. The procedure is described in Algorithm 2 [64]. In both exact and approximate cases, we call $\mathbf{A}^{(n)}$ a mode-n HOSVD basis if it leads to a tensor $\mathcal{S}$ that satisfies points (i) and (ii) above.

Algorithm 2 HOSVD

```
procedure TRUNCATED HOSVD(X, R_1, ..., R_N)
    for n = 1, ..., N do
        A^(n) ← R_n leading left singular vectors of X_(n)
    end for
    S ← X · (A^(1), ..., A^(N))
    return S, A^(1), ..., A^(N)
end procedure
```

1.2.4 The Best Tucker HOSVD Approximation Problem

An important difference between the matrix SVD and the HOSVD is that there is no higher dimensional equivalent of the Eckhart-Young theorem: truncating a HOSVD does not result in an optimal approximation [30]. To determine the best orthonormal ATD of a given $\mathcal{X}$, we minimize the approximation error in the Frobenius norm. The minimization problem is

$$\begin{aligned} \min_{\mathcal{S},\{\mathbf{A}^{(n)}\}} \quad & \frac{1}{2}\left\|\mathcal{X} - (\mathbf{A}^{(1)},\ldots,\mathbf{A}^{(N)})\cdot\mathcal{S}\right\|_F^2 \\ \text{subject to} \quad & \mathcal{S}\in\mathbb{R}^{R_1\times\cdots\times R_N},\ \mathbf{A}^{(n)}\in\mathbb{R}^{I_n\times R_n} \text{ and } \mathbf{A}^{(n)\intercal}\mathbf{A}^{(n)} = \mathbf{I}_{R_n}., \end{aligned}$$

which is equivalent to [31]

$$\begin{aligned} \max_{\{\mathbf{A}^{(n)}\}} \quad & \frac{1}{2}\left\|\mathcal{X}\cdot(\mathbf{A}^{(1)},\ldots,\mathbf{A}^{(N)})\right\|_F^2 \\ \text{subject to} \quad & \mathbf{A}^{(n)}\in\mathbb{R}^{I_n\times R_n} \text{ and } \mathbf{A}^{(n)\intercal}\mathbf{A}^{(n)} = \mathbf{I}_{R_n}, \end{aligned} \tag{1.4}$$

where $\mathcal{S} = \mathcal{X}\cdot\left(\mathbf{A}^{(1)},\ldots,\mathbf{A}^{(N)}\right)$.

Algorithm 3 HOOI

```
procedure HOOI(X, A^(1), ..., A^(N))
    for n = 1, ..., N do
        Y ← X · (A^(1), ..., A^(n-1), I, A^(n+1), ..., A^(N))
        A^(n) ← R_n leading left singular vectors of Y_(n)
    end for
    S ← (A^(1)ᵀ, ..., A^(N)ᵀ) · X
    return S, A^(1), ..., A^(N)
end procedure
```

The most popular method for solving (1.4) is the higher-order orthogonal iteration (HOOI), first proposed in [31] and reproduced here as Algorithm 3 [64]. If we use the natural fiber ordering of [40] we see that the mode-n matricization of $\left(\mathbf{A}^{(1)\intercal},\ldots,\mathbf{A}^{(N)\intercal}\right)\cdot\mathcal{X}$ is

$$\mathbf{Y}_{(n)} := \mathbf{A}^{(n)\intercal}\mathbf{X}_{(n)}\left(\mathbf{A}^{(1)}\otimes\cdots\otimes\mathbf{A}^{(n-1)}\otimes\mathbf{A}^{(n+1)}\otimes\cdots\otimes\mathbf{A}^{(N)}\right).$$

Fixing all factor matrices but $\mathbf{A}^{(n)}$ gives $\frac{1}{2}\left\|\mathbf{A}^{(n)\intercal}\mathbf{Y}_{(n)}\right\|_F^2$, which is maximized by taking the R_n leading left singular vectors of $\mathbf{Y}_{(n)}$ as the columns of $\mathbf{A}^{(n)}$. The HOOI Algorithm implements this iteration. While simple to implement, HOOI may be slow to converge in practice, hence alternative optimization methods are desired.

1.3 Optimization Method Building Blocks

In this section we provide introductions to the generalized minimal residual method, the conjugate gradients method, and the limited memory quasi-Newton method. Linearly and nonlinearly preconditioned variants of these methods on both Euclidean space and matrix manifolds are developed in chapter 2.

1.3.1 Generalized Minimal Residual Methods

The Generalized Minimal Residual (GMRES) method [86] is an iteration for solving $\mathbf{Ax} = \mathbf{b}$ for general invertible matrices $\mathbf{A}$ that minimizes $\|\mathbf{r}_k\| = \|\mathbf{b} - \mathbf{Ax}_k\|$ over the Krylov subspace $\operatorname{span}\left\{\mathbf{r}_0, \mathbf{Ar}_0, \ldots, \mathbf{A}^{k-1}\mathbf{r}_0\right\}$ at every step. One iteration of GMRES with a window size of m is described in Algorithm 4, where $\mathbf{e}_1$ is the first column of the $(m+1)\times(m+1)$ identity matrix, and $\overline{\mathbf{H}}_m$ is the $(m+1)\times m$ matrix with nonzero elements $h_{i,j}$ computed in the algorithm.

Algorithm 4 GMRES(m) Iteration

procedure GMRES($\mathbf{A}, \mathbf{b}, \mathbf{x}_0, m$)
  $\mathbf{r}_0 = \mathbf{b} - \mathbf{Ax}_0$, $\beta = \|\mathbf{r}_0\|$, $\mathbf{v}_1 = \mathbf{r}_0/\beta$
  for $j = 1, 2, \ldots, m$ **do**
    $h_{i,j} = \mathbf{v}_i^{\mathsf{T}}(\mathbf{Av}_j)$, $i = 1, 2, \ldots, j$
    $\widehat{\mathbf{v}}_{j+1} = \mathbf{Av}_j - \sum_{i=1}^{j} h_{i,j}\mathbf{v}_i$
    $h_{j+1,j} = \|\widehat{\mathbf{v}}_{j+1}\|$
    $\mathbf{v}_{j+1} = \widehat{\mathbf{v}}_{j+1}/h_{j+1,j}$
  end for
  $\mathbf{x}_m = \mathbf{x}_0 + \mathbf{V}_m\mathbf{y}_m$, where $\mathbf{y}_m = \operatorname{argmin} \left\|\beta\mathbf{e}_1 - \overline{\mathbf{H}}_m\mathbf{y}\right\|$
  return $\mathbf{x}_m$
end procedure

Nonlinear GMRES (NGMRES) [32, 33, 105] is, in its simplest form, an algorithm that accelerates the convergence of the gradient descent method for $\mathbf{g}(\mathbf{x}) = \mathbf{0}$: $\mathbf{x}_{k+1} = \mathbf{x}_k - \gamma_k\mathbf{g}_k$. NGMRES is closely related to Anderson acceleration (Anderson mixing) [6, 18, 45, 104] and the direct inversion in the iterative subspace (DIIS) method [81], and is equivalent to an inverse projected Broyden secant method [45]. The name NGMRES is used because it can be formulated as a generalization of GMRES for linear systems [86], see, e.g., [104, 105].

NGMRES begins by computing a tentative iterate $\overline{\mathbf{x}}_{k+1} = \mathbf{x}_k - \gamma_k\mathbf{g}_k$; see [33] for strategies to choose γ_k (e.g. line-search). Given past iterates $\{\mathbf{x}_j\}_{j=k-m+1}^{k}$, we seek an

accelerated iterate in the form of a linear combination of $\overline{\mathbf{x}}_{k+1}$ and the past values:

$$\widehat{\mathbf{x}}_{k+1} = \overline{\mathbf{x}}_{k+1} + \sum_{j=k-m+1}^{k} \beta_j (\overline{\mathbf{x}}_{k+1} - \mathbf{x}_j).$$

The ideal coefficients β_j are those that minimize $\|\widehat{\mathbf{g}}(\mathbf{x}_{k+1})\|_2$. However, this norm is a nonlinear function of $\boldsymbol{\beta}$, hence it is only approximately minimized in practice. By linearizing $\mathbf{g}(\widehat{\mathbf{x}}_{k+1})$ about $\overline{\mathbf{x}}_{k+1}$ we obtain

$$\begin{aligned}
\mathbf{g}(\widehat{\mathbf{x}}_{k+1}) &\approx \mathbf{g}(\overline{\mathbf{x}}_{k+1}) + \sum_{j=k-m+1}^{k} \left.\frac{\partial \mathbf{g}}{\partial \mathbf{x}}\right|_{\overline{\mathbf{x}}_{i+1}} \beta_j (\overline{\mathbf{x}}_{k+1} - \mathbf{x}_j) \\
&\approx \mathbf{g}(\overline{\mathbf{x}}_{k+1}) + \sum_{j=k-m+1}^{k} \beta_j (\mathbf{g}(\overline{\mathbf{x}}_{k+1}) - \mathbf{g}(\mathbf{x}_j)),
\end{aligned}$$

with the second approximation eliminating the need for Hessian-vector products. This simplifies the norm minimization to a least squares problem for $\boldsymbol{\beta}$:

$$\left\| \mathbf{g}(\overline{\mathbf{x}}_{k+1}) + \sum_{j=k-m+1}^{k} \beta_j (\mathbf{g}(\overline{\mathbf{x}}_{k+1}) - \mathbf{g}(\mathbf{x}_j)) \right\|_2 .$$

If we let $(\mathbf{g}(\overline{\mathbf{x}}_{k+1}) - \mathbf{g}(\mathbf{x}_j))$ be the j^{th} column of $\mathbf{A}$, $\mathbf{b} = \mathbf{g}(\overline{\mathbf{x}}_{k+1})$, and $\boldsymbol{\beta} = (\beta_{k-m+1}, \ldots, \beta_k)^\intercal$, we may describe $\boldsymbol{\beta}$ as the solution to the normal equations, $\mathbf{A}^\intercal \mathbf{A} \boldsymbol{\beta} = -\mathbf{A}^\intercal \mathbf{b}$, and can solve for $\boldsymbol{\beta}$ via any method for the linear least-squares problem (e.g. QR decomposition, the Moore–Penrose pseudoinverse). One iteration of NGMRES is described in Algorithm 5.

Algorithm 5 Nonlinear GMRES Iteration

procedure NGMRES($\mathbf{g}, \mathbf{x}_{k-m+1}, \ldots, \mathbf{x}_k$)
  $\overline{\mathbf{x}}_{k+1} = \mathbf{x}_k - \gamma_k \mathbf{g}(\mathbf{x}_k)$
  Solve $\left\| \mathbf{g}(\overline{\mathbf{x}}_{k+1}) + \sum_{j=k-m+1}^{k} \beta_j (\mathbf{g}(\overline{\mathbf{x}}_{k+1}) - \mathbf{g}(\mathbf{x}_j)) \right\|_2$ for $\boldsymbol{\beta}$
  $\mathbf{p}_k = \sum_{j=k-m+1}^{k} \beta_j (\overline{\mathbf{x}}_{k+1} - \mathbf{x}_j)$
  $\mathbf{x}_{k+1} = \overline{\mathbf{x}}_{k+1} + \alpha_k \mathbf{p}_k$ ▷ α_k determined by line-search
  return $\mathbf{x}_{k+1}$
end procedure

1.3.2 Conjugate Gradients Method

The conjugate gradients (CG) method [77, 91] is an iterative solver for linear systems $\mathbf{Ax} = \mathbf{b}$ with symmetric positive definite (SPD) matrices $\mathbf{A}$; or equivalently, a solver that minimizes convex quadratic objective functions

$$f(\mathbf{x}) = \tfrac{1}{2}\mathbf{x}^\intercal \mathbf{A}\mathbf{x} - \mathbf{b}^\intercal \mathbf{x}. \tag{1.5}$$

Starting from (possibly arbitrary) $\mathbf{x}_0$ with $\mathbf{p}_0 = \mathbf{r}_0 = \mathbf{b} - \mathbf{A}\mathbf{x}_0$, one step of the CG iteration is described in Algorithm 6. This iteration is based on the conjugacy of the sequence of search directions $\mathbf{p}_k$ with respect to $\mathbf{A}$, and generates an orthogonal sequence of residual vectors $\mathbf{r}_k$ [77]. In addition to the low storage requirements, we only require the means to compute the matrix-vector product $\mathbf{A}\mathbf{p}_k$; storage of $\mathbf{A}$ is unnecessary.

Algorithm 6 Conjugate Gradients Iteration

procedure CG($\mathbf{A}, \mathbf{x}_k, \mathbf{p}_k, \mathbf{r}_k$)
  $\alpha_k = (\mathbf{r}_k^\intercal \mathbf{r}_k)/(\mathbf{p}_k^\intercal \mathbf{A}\mathbf{p}_k)$ ▷ Exact minima for quadratic objective
  $\mathbf{x}_{k+1} = \mathbf{x}_k + \alpha_k \mathbf{p}_k$
  $\mathbf{r}_{k+1} = \mathbf{r}_k + \alpha_k \mathbf{A}\mathbf{p}_k$
  $\beta_{k+1} = (\mathbf{r}_{k+1}^\intercal \mathbf{r}_{k+1})/(\mathbf{r}_k^\intercal \mathbf{r}_k)$
  $\mathbf{p}_{k+1} = -\mathbf{r}_{k+1} + \beta_{k+1}\mathbf{p}_k$
  return $\mathbf{x}_{k+1}$, $\mathbf{p}_{k+1}$, $\mathbf{r}_{k+1}$
end procedure

The nonlinear conjugate gradients (NCG) iteration arose as an adaptation of CG for minimizing general nonlinear objective functions $f(\mathbf{x})$ [77]. Three main changes are required: (i) we replace the residual $\mathbf{r}_k$ with the gradient of the objective function, $\mathbf{g}(\mathbf{x}) = \nabla f(\mathbf{x})$, (ii) the step length α_k must be determined by a line-search, and (iii) the search direction update parameter β_{k+1} can be specified by a number of different formulas. Three of the most successful are

$$\text{Polak-Ribière [79]:} \quad \beta_{k+1}^{\text{PR}} = \frac{\mathbf{g}_{k+1}^\intercal \mathbf{y}_k}{\mathbf{g}_k^\intercal \mathbf{g}_k}, \tag{1.6}$$

$$\text{Hestenes-Stiefel [56]:} \quad \beta_{k+1}^{\text{HS}} = \frac{\mathbf{g}_{k+1}^\intercal \mathbf{y}_k}{\mathbf{p}_k^\intercal \mathbf{y}_k}, \tag{1.7}$$

$$\text{Hager-Zhang [53]:} \quad \beta_{k+1}^{\text{HZ}} = \left(\mathbf{y}_k - 2\mathbf{p}_k \frac{\|\mathbf{y}_k\|^2}{\mathbf{p}_k^\intercal \mathbf{y}_k}\right)^\intercal \frac{\mathbf{g}_{k+1}}{\mathbf{p}_k^\intercal \mathbf{y}_k}, \tag{1.8}$$

where $\mathbf{g}_{k+1} = \mathbf{g}(\mathbf{x}_{k+1})$ and $\mathbf{y}_k = \mathbf{g}_{k+1} - \mathbf{g}_k$. One iteration of NCG is described in Algorithm 7. Like the linear version, NCG enjoys the benefits of very low storage requirements.

Algorithm 7 Nonlinear Conjugate Gradients Iteration

1: **procedure** NCG($\mathbf{g}(\cdot), \mathbf{x}_k, \mathbf{p}_k, \mathbf{g}_k$)
2: $\quad \mathbf{x}_{k+1} = \mathbf{x}_k + \alpha_k \mathbf{p}_k$ ▷ α_k determined by line-search
3: $\quad \mathbf{g}_{k+1} = \mathbf{g}(\mathbf{x}_{k+1})$
4: $\quad$ Compute β_{k+1} by one of (1.6–1.8)
5: $\quad \mathbf{p}_{k+1} = -\mathbf{g}_{k+1} + \beta_{k+1}\mathbf{p}_k$
6: $\quad$ **return** $\mathbf{x}_{k+1}$, $\mathbf{p}_{k+1}$, $\mathbf{g}_{k+1}$
7: **end procedure**

1.3.3 Limited Memory Quasi-Newton (QN) Methods

QN methods [35] are iterations based on the standard Newton-Raphson iteration for solving nonlinear systems $\mathbf{g}(\mathbf{x})$ (or minimizing nonlinear functions $f(\mathbf{x})$, depending on the context). The expensive evaluation of the Jacobian (or Hessian) matrix at each iteration is replaced with a low-rank update of a matrix approximation based on a secant condition. The quadratic rate of convergence of Newton's method is traded for super-linear convergence, with the hope that the approximation results in a significantly lower per iteration cost.

Limited memory QN iterations aim for further savings in storage requirements and work per iteration by expressing the matrix approximation in terms of an initial matrix (often diagonal) and at most m vector pairs. In this way the full matrix approximation does not need to be formed or stored to compute matrix-vector products. We describe two limited memory QN methods in the remainder of this subsection: one based on the good Broyden update for solving nonlinear systems (L-Broyden), and one based on the Broyden-Fletcher-Goldfarb-Shanno (BFGS) update for minimizing nonlinear objective functions (L-BFGS).

The L-Broyden Update

We first describe the general good Broyden update for standard QN methods, then give the limited memory variant. When considering a nonlinear system $\mathbf{g}(\mathbf{x}) = \mathbf{0}$, we denote an approximation to the Jacobian matrix by $\mathbf{A}_k$, and define the vectors $\mathbf{s}_k = \mathbf{x}_{k+1} - \mathbf{x}_k$ and $\mathbf{y}_k = \mathbf{g}_{k+1} - \mathbf{g}_k$ [35]. Broyden's good update minimizes the change in the affine model

$$\mathbf{M}_{k+1}(\mathbf{x}) = \mathbf{g}_{k+1} + \mathbf{A}_{k+1}(\mathbf{x} - \mathbf{x}_{k+1})$$

between iterations, subject to the secant equation

$$\mathbf{A}_{k+1}\mathbf{s}_k = \mathbf{y}_k.$$

The resulting rank-one update is

$$\mathbf{A}_{k+1} = \mathbf{A}_k + \frac{(\mathbf{y}_k - \mathbf{A}_k\mathbf{s}_k)\mathbf{s}_k^\intercal}{\mathbf{s}_k^\intercal\mathbf{s}_k}, \tag{1.9}$$

which, by applying the Sherman-Morrison-Woodbury formula [11,90,106], gives the inverse matrix update

$$\mathbf{A}_{k+1}^{-1} = \mathbf{A}_k^{-1} + \frac{(\mathbf{s}_k - \mathbf{A}_k^{-1}\mathbf{y}_k)\mathbf{s}_k^\intercal\mathbf{A}_k^{-1}}{\mathbf{s}_k^\intercal\mathbf{A}_k^{-1}\mathbf{y}_k}. \tag{1.10}$$

An example of one QN iteration in this context is given in Algorithm 8.

Algorithm 8 Quasi-Newton Iteration for Nonlinear Systems

procedure QN($\mathbf{g}, \mathbf{x}_k, \mathbf{A}_k^{-1}$)
  $\mathbf{p}_k = -\mathbf{A}_k^{-1}\mathbf{g}(\mathbf{x}_k)$
  $\mathbf{x}_{k+1} = \mathbf{x}_k + \alpha_k\mathbf{p}_k$ ▷ α_k determined by line-search
  $\mathbf{s}_k = \mathbf{x}_{k+1} - \mathbf{x}_k$
  $\mathbf{y}_k = \mathbf{g}(\mathbf{x}_{k+1}) - \mathbf{g}(\mathbf{x}_k)$
  $\mathbf{A}_{k+1}^{-1} = \mathbf{U}(\mathbf{A}_k^{-1}, \mathbf{s}_k, \mathbf{y}_k)$ ▷ $\mathbf{U}$ update formula (1.10).
  return $\mathbf{x}_{k+1}$, $\mathbf{A}_{k+1}^{-1}$
end procedure

In the limited memory context, where only a window of m previous vector pairs are retained, a compact, non-recursive representation of update (1.10) derived in [20] is

$$\mathbf{A}_k^{-1} = \left[\mathbf{A}_0^{(k)}\right]^{-1} - \left(\left[\mathbf{A}_0^{(k)}\right]^{-1}\mathbf{Y}_k - \mathbf{S}_k\right)\left(\mathbf{M}_k + \mathbf{S}_k^\intercal\left[\mathbf{A}_0^{(k)}\right]^{-1}\mathbf{Y}_k\right)^{-1}\mathbf{S}_k^\intercal\left[\mathbf{A}_0^{(k)}\right]^{-1} \tag{1.11}$$

where

$$\mathbf{S}_k = [\mathbf{s}_{k-m} \,|\, \mathbf{s}_{k-m+1} \,|\, \cdots \,|\, \mathbf{s}_{k-1}], \tag{1.12}$$
$$\mathbf{Y}_k = [\mathbf{y}_{k-m} \,|\, \mathbf{y}_{k-m+1} \,|\, \cdots \,|\, \mathbf{y}_{k-1}], \tag{1.13}$$

and

$$(\mathbf{M}_k)_{i,j} = \begin{cases} -\mathbf{s}_{i-1}^\intercal\mathbf{s}_{j-1} & \text{if } i > j \\ 0 & \text{otherwise} \end{cases}. \tag{1.14}$$

For $\mathbf{A}_0^{(k)}$ we will typically use a scaled identity matrix. It is this representation that we will use in the L-Broyden iteration.

The L-BFGS Update

In the optimization context we use $\mathbf{B}_k$ to denote an approximation to the Hessian and $\mathbf{H}_k$ for an approximation to the inverse of the Hessian. Arguably the most successful QN update for nonlinear optimization is the BFGS update, that, in addition to enforcing the secant equation, ensures that $\mathbf{B}_k$ is SPD provided $\mathbf{s}_{k-1}^\intercal \mathbf{y}_{k-1} > 0$ and $\mathbf{B}_{k-1}$ is SPD [35]. This is a rank-two update given by

$$\mathbf{B}_{k+1} = \mathbf{B}_k + \frac{\mathbf{y}_k \mathbf{y}_k^\intercal}{\mathbf{y}_k^\intercal \mathbf{s}_k} - \frac{\mathbf{B}_k \mathbf{s}_k (\mathbf{B}_k \mathbf{s}_k)^\intercal}{\mathbf{s}_k^\intercal \mathbf{B}_k \mathbf{s}_k}, \tag{1.15}$$

with inverse update

$$\mathbf{H}_{k+1} = \mathbf{H}_k + \frac{(\mathbf{s}_k - \mathbf{H}_k \mathbf{y}_k)\mathbf{s}_k^\intercal + \mathbf{s}_k (\mathbf{s}_k - \mathbf{H}_k \mathbf{y}_k)^\intercal}{\mathbf{y}_k^\intercal \mathbf{s}_k} - \frac{\langle \mathbf{s}_k - \mathbf{H}_k \mathbf{y}_k, \mathbf{y}_k \rangle \mathbf{s}_k \mathbf{s}_k^\intercal}{(\mathbf{y}_k^\intercal \mathbf{s}_k)^2}. \tag{1.16}$$

In the limited memory case, given initial Hessian approximation $\mathbf{B}_0^{(k)}$ $\left(\text{or } \mathbf{H}_0^{(k)}\right)$ and at most m vector pairs $\mathbf{y}_k$, $\mathbf{s}_k$, compact versions of (1.15) and (1.16) are [20]:

$$\mathbf{B}_k = \mathbf{B}_0^{(k)} - \begin{bmatrix} \mathbf{B}_0^{(k)} \mathbf{S}_k & \mathbf{Y}_k \end{bmatrix} \begin{bmatrix} \mathbf{S}_k^\intercal \mathbf{B}_0^{(k)} \mathbf{S}_k & \mathbf{L}_k \\ \mathbf{L}_k^\intercal & -\mathbf{D}_k \end{bmatrix} \begin{bmatrix} \mathbf{S}_k^\intercal \mathbf{B}_0^{(k)} \\ \mathbf{Y}_k^\intercal \end{bmatrix} \tag{1.17}$$

and

$$\mathbf{H}_k = \mathbf{H}_0^{(k)} + \begin{bmatrix} \mathbf{S}_k & \mathbf{H}_0^{(k)} \mathbf{Y}_k \end{bmatrix} \begin{bmatrix} \mathbf{R}_k^{-\intercal} (\mathbf{D}_k + \mathbf{Y}_k^\intercal \mathbf{H}_0^{(k)} \mathbf{Y}_k) \mathbf{R}_k^{-1} & -\mathbf{R}_k^{-\intercal} \\ -\mathbf{R}_k^{-1} & \mathbf{0} \end{bmatrix} \begin{bmatrix} \mathbf{S}_k^\intercal \\ \mathbf{Y}_k^\intercal \mathbf{H}_0^{(k)} \end{bmatrix}, \tag{1.18}$$

where $\mathbf{S}_k$ and $\mathbf{Y}_k$ are as in (1.12) and (1.13), and

$$\mathbf{D}_k = \operatorname{diag}[\mathbf{s}_{k-m}^\intercal \mathbf{y}_{k-m}, \ldots, \mathbf{s}_{k-1}^\intercal \mathbf{y}_{k-1}], \tag{1.19}$$

$$(\mathbf{L}_k)_{i,j} = \begin{cases} (\mathbf{s}_{k-m-1+i})^\intercal (\mathbf{y}_{k-m-1+j}) & \text{if } i > j \\ 0 & \text{otherwise} \end{cases}, \tag{1.20}$$

$$(\mathbf{R}_k)_{i,j} = \begin{cases} (\mathbf{s}_{k-m-1+i})^\intercal (\mathbf{y}_{k-m-1+j}) & \text{if } i \leq j \\ 0 & \text{otherwise} \end{cases}. \tag{1.21}$$

It is common to set $\mathbf{H}_0^{(k)} = \gamma_k \mathbf{I}$, where

$$\gamma_k = \frac{\mathbf{s}_{k-1}^\intercal \mathbf{y}_{k-1}}{\mathbf{y}_{k-1}^\intercal \mathbf{y}_{k-1}}. \tag{1.22}$$

γ_k is a scaling factor which attempts to make the size of $\mathbf{H}_0^{(k)}$ similar to that of the true Hessian inverse, $\nabla^2 f(\mathbf{x}_{k-1})^{-1}$, along the most recent search direction. This helps ensure the search direction $\mathbf{p}_k$ is scaled so that a unit step length α_k is acceptable in more iterations [77]. When working with the inverse L-BFGS update, the product $\mathbf{H}_k\mathbf{g}_k$ defining the QN direction $\mathbf{p}_k$ (similar to Algorithm 8) can be efficiently computed by a two-loop recursion, described in Algorithm 9.

Algorithm 9 L-BFGS Two-Loop Recursion

procedure 2LOOP($\mathbf{H}_0^{(k)}, \mathbf{g}_k, \mathbf{S}_k, \mathbf{Y}_k$)
  $\mathbf{q} = \mathbf{g_k}$
  for $i = k-1, k-2, \ldots, k-m$ **do**
    $\rho_i = (\mathbf{y}_i^\intercal \mathbf{s}_i)^{-1}$
    $\alpha_i = \rho_i \mathbf{s}_i^\intercal \mathbf{q}$
    $\mathbf{q} = \mathbf{q} - \alpha_i \mathbf{y}_i$
  end for
  $\mathbf{r} = \mathbf{H}_0^{(k)} \mathbf{q}$
  for $i = k-m, k-m+1, \ldots, k-1$ **do**
    $\beta = \rho_i \mathbf{y}_i^\intercal \mathbf{r}$
    $\mathbf{r} = \mathbf{r} + (\alpha_i - \beta)\mathbf{s}_i$
  end for
  return r ▷ Contains $\mathbf{H_k g}_k$
end procedure

In situations where $\mathbf{s}_{k-1}^\intercal \mathbf{y}_{k-1} \leq 0$, there is a damped BFGS variant that ensures the updated Hessian is SPD [77] by defining

$$\theta_k = \begin{cases} 1 & \text{if } \mathbf{s}_k^\intercal \mathbf{y}_k \geq 0.1 \mathbf{s}_k^\intercal \mathbf{B}_k \mathbf{s}_k \\ (0.9\mathbf{s}_k^\intercal \mathbf{B}_k \mathbf{s}_k)/(\mathbf{s}_k^\intercal \mathbf{B}_k \mathbf{s}_k - \mathbf{s}_k^\intercal \mathbf{y}_k) & \text{if } \mathbf{s}_k^\intercal \mathbf{y}_k < 0.1 \mathbf{s}_k^\intercal \mathbf{B}_k \mathbf{s}_k \end{cases}$$

and setting

$$\mathbf{y}_k = \theta_k \mathbf{y}_k + (1 - \theta_k)\mathbf{B}_k \mathbf{s}_k,$$

which reduces to the standard update for $\mathbf{s}_k^\intercal \mathbf{y}_k \geq 0.1 \mathbf{s}_k^\intercal \mathbf{B}_k \mathbf{s}_k$. We use this damping step in our L-BFGS implementations.

Relationship of BFGS to CG

There are some noteworthy similarities between the CG and BFGS methods being considered, both for the convex quadratic objective function (1.5) and more general nonlinear

objective functions. It has been shown for convex quadratic objective functions that the CG and BFGS iterations are identical when exact line-searches are used [19, 76]. Furthermore, the "memory-less" BFGS method (L-BFGS with $m = 1$), in conjunction with an exact line-search, applied to a general nonlinear objective function is equivalent to using NCG with the Hestenes-Stiefel (HS) or Polak-Ribière (PR) β formulas (which are equivalent since $\mathbf{g}_{k+1}^{\intercal}\mathbf{p}_k = 0$ by the exact line-search) [72, 77].

1.4 Matrix Manifold Optimization

In this section we introduce the concepts from matrix manifold optimization that we use to adapt nonlinearly preconditioned optimization methods to the manifold setting. As a general reference for this section, see [4].

1.4.1 Motivation for Manifold Optimization

As previously noted, the tensor Frobenius norm is invariant under orthogonal transformations, meaning (1.4) does not have isolated maxima. Furthermore, the orthonormality imposed on $\{\mathbf{A}^{(n)}\}_{n=1}^{N}$ introduces a large number of equality constraints. However, if we define the N-tuple of factor matrices to be a point on a Cartesian product of matrix manifolds, we are able to eliminate both of these issues. We restrict our discussion to *Riemannian manifolds*, which are those manifolds that have smoothly varying inner products.

The *Stiefel manifold*, $\mathrm{St}(n,p) = \{\mathbf{X} \in \mathbb{R}^{n\times p} | \mathbf{X}^{\intercal}\mathbf{X} = \mathbf{I}_p\}$, is the set of all $n \times p$ orthonormal matrices. The *Grassmann manifold* (*Grassmannian*), $\mathrm{Gr}(n,p)$, is the set of p-dimensional linear subspaces of $\mathbb{R}^n$ [39]. In both contexts $p \leq n$. We can represent $\mathcal{Y} \in \mathrm{Gr}(n,p)$ as the column space of some $\mathbf{Y} \in \mathrm{St}(n,p)$. This $\mathbf{Y}$ is not unique: the subset of $\mathrm{St}(n,p)$ with the same column space as $\mathbf{Y}$ is $\mathbf{Y}O_p := \{\mathbf{YM} | \mathbf{M} \in O_p\}$, where O_p is the set of $p \times p$ orthogonal matrices. $\mathrm{Gr}(n,p)$ is thus identified with the set of matrix equivalence classes $\mathrm{St}(n,p)/O_p := \{\mathbf{Y}O_p | \mathbf{Y}^{\intercal}\mathbf{Y} = \mathbf{I}_p\}$ induced by $\mathbf{X} \sim \mathbf{Y}$ if and only if $\mathrm{Col}(\mathbf{X}) = \mathrm{Col}(\mathbf{Y})$. The inner product on these manifolds is $\langle \mathbf{X}, \mathbf{Y} \rangle = \mathrm{tr}(\mathbf{X}^{\intercal}\mathbf{Y})$.

If (1.4) is solved over a Cartesian product of Grassmannians, the representative N-tuples of factor matrices satisfy the orthonormality constraints by definition. Furthermore, because these matrices represent equivalence classes, the result is an unconstrained problem with isolated extrema:

$$\max_{\{\mathbf{A}^{(n)}\}} \quad \frac{1}{2}\left\|\mathcal{X} \cdot (\mathbf{A}^{(1)}, \ldots, \mathbf{A}^{(N)})\right\|_F^2, \tag{1.23}$$

where $\mathbf{A}^{(n)} \in \mathrm{St}(I_n, R_n)$ represents $\mathcal{A}^{(n)} \in \mathrm{Gr}(I_n, R_n)$. Expressions for the Riemannian gradient and Hessian of this objective function can be obtained from their Euclidean equivalents. We refer readers to [40, 61], where these expressions can be found. Once solved, orthogonal transformation matrices that map representative matrices $\mathbf{A}^{(n)}$ to HOSVD bases are obtained by applying Algorithm 2 to $\mathcal{S}$.

1.4.2 Directions and Movement on Manifolds

Let $\mathcal{M}$ denote an arbitrary manifold. A *tangent vector* at $x \in \mathcal{M}$, denoted ξ_x, describes a possible direction of travel tangent to $\mathcal{M}$ at x. The *tangent space*, $T_x\mathcal{M}$, is the vector space of all tangent vectors at x. A tangent vector at $\mathbf{Y} \in \mathrm{St}(n, p)$ is itself an $n \times p$ matrix, and just as $\mathbf{Y}$ can represent $\mathcal{Y} \in \mathrm{Gr}(n, p)$, we can use elements of $T_{\mathbf{Y}}\mathrm{St}(n, p)$ to represent elements of $T_{\mathcal{Y}}\mathrm{Gr}(n, p)$. $T_{\mathbf{Y}}\mathrm{St}(n, p)$ may be expressed as $\mathcal{V}_{\mathbf{Y}} \oplus \mathcal{H}_{\mathbf{Y}}$, where the *vertical space* $\mathcal{V}_{\mathbf{Y}}$ contains directions for movement within the equivalence class $\mathcal{Y}$ and the *horizontal space* $\mathcal{H}_{\mathbf{Y}}$ contains directions for movement into new equivalence classes. Elements of $\mathcal{H}_{\mathbf{Y}}$ are used as unique representative tangent vectors for points on $\mathrm{Gr}(n, p)$: $\mathcal{V}_{\mathbf{Y}} = \{\mathbf{YM} | \mathbf{M} = -\mathbf{M}^\intercal, \mathbf{M} \in \mathbb{R}^{p \times p}\}$ and $\mathcal{H}_{\mathbf{Y}} = \{\mathbf{Z} \in \mathbb{R}^{n \times p} | \mathbf{Y}^\intercal\mathbf{Z} = \mathbf{0}_p\}$, where the orthogonal projection onto $\mathcal{H}_{\mathbf{Y}}$ is [4, 39]

$$\Pi_{\mathbf{Y}} = \mathbf{I} - \mathbf{YY}^\intercal. \tag{1.24}$$

Movement along $\mathcal{M}$ in the direction of ξ_x is described by a *retraction* mapping. On Riemannian manifolds the *exponential map* describes motion along a geodesic, the curve connecting two points with minimal length. The exponential map on $\mathrm{St}(n, p)$ starting at $\mathbf{Y}$ in the direction $\xi_{\mathbf{Y}}$ is

$$\mathrm{Exp}_{\mathbf{Y}}(t\xi_{\mathbf{Y}}) = \mathbf{YV}\cos(\mathbf{\Sigma}t)\mathbf{V}^\intercal + \mathbf{U}\sin(\mathbf{\Sigma}t)\mathbf{V}^\intercal, \tag{1.25}$$

where $\xi_{\mathbf{Y}}$ has compact SVD $\mathbf{U\Sigma V}^\intercal$. This mapping is expensive to compute, requiring a SVD and several matrix products. A cheaper, but less accurate retraction is

$$R_{\mathbf{Y}}(t\xi_{\mathbf{Y}}) = \mathrm{qf}(\mathbf{Y} + t\xi_{\mathbf{Y}}), \tag{1.26}$$

where $\mathrm{qf}(\mathbf{Z}) = \mathbf{Q}$ factor of the thin $\mathbf{QR}$ decomposition of $\mathbf{Z}$ [4].

Tangent spaces $T_x\mathcal{M}$ and $T_y\mathcal{M}$ for $x \neq y$ are generally different vector spaces, hence linear combinations of $\xi_x \in T_x\mathcal{M}$ and $\eta_y \in T_y\mathcal{M}$ are not well defined. By using a *vector transport* mapping, we instead find a $\xi'_y \in T_y\mathcal{M}$ to use in place of ξ_x. Given $\mathbf{X} \in \mathrm{St}(n, p)$

and $\xi_{\mathbf{X}}, \eta_{\mathbf{X}} \in T_{\mathbf{X}}\mathrm{St}(n,p)$, if $\xi_{\mathbf{X}}$ has compact SVD $\mathbf{U}\mathbf{\Sigma}\mathbf{V}^\intercal$, the parallel transport of $\eta_{\mathbf{X}}$ along the geodesic of length t starting at $\mathbf{X}$ in the direction of $\xi_{\mathbf{X}}$ is [4]:

$$\mathcal{T}_{\mathbf{X},t\xi_{\mathbf{X}}}(\eta_{\mathbf{X}}) = \left(\begin{bmatrix} \mathbf{X}\mathbf{V} & \mathbf{U} \end{bmatrix} \begin{bmatrix} -\sin(\mathbf{\Sigma}t) \\ \cos(\mathbf{\Sigma}t) \end{bmatrix} \mathbf{U}^\intercal + (\mathbf{I} - \mathbf{U}\mathbf{U}^\intercal) \right) \eta_{\mathbf{X}}. \tag{1.27}$$

If $\xi_{\mathbf{X}} = \eta_{\mathbf{X}}$, this simplifies to

$$\mathcal{T}_{\mathbf{X},t\xi_{\mathbf{X}}}(\xi_{\mathbf{X}}) = \begin{bmatrix} \mathbf{X}\mathbf{V} & \mathbf{U} \end{bmatrix} \begin{bmatrix} -\sin(\mathbf{\Sigma}t) \\ \cos(\mathbf{\Sigma}t) \end{bmatrix} \mathbf{\Sigma}\mathbf{V}^\intercal.$$

As in the case of retractions, we may also use a cheaper, non-exact alternative. If $\mathbf{X}, \mathbf{Y} \in \mathrm{St}(n,p)$ and $\xi_{\mathbf{X}} \in T_{\mathbf{X}}\mathrm{St}(n,p)$ are given, a substitute vector in $T_{\mathbf{Y}}\mathrm{St}(n,p)$ is determined using (1.24) [4]:

$$\widetilde{\mathcal{T}}_{\mathbf{Y}}(\xi_{\mathbf{X}}) = \Pi_{\mathbf{Y}}\xi_{\mathbf{X}}. \tag{1.28}$$

The direction of travel from x to y cannot be described by a vector $y - x$: this operation is not defined. Note, however, that $\mathrm{Exp}_x(\cdot)$ defines a diffeomorphism from a neighbourhood $\widehat{\mathcal{U}}$ about the origin of $T_x\mathcal{M}$ onto a neighbourhood $\mathcal{U}$ of x. If $\xi_x \in \widehat{\mathcal{U}}$ implies $t\xi_x \in \widehat{\mathcal{U}}$ for $t \in [0,1]$, $\mathcal{U}$ is a normal neighbourhood of x [4, 80]. $\mathrm{Exp}_x(\cdot)$ is invertible within a normal neighbourhood of x, and when $\mathrm{Exp}_x(\cdot)$ is invertible, a tangent vector defining a geodesic from x to y can be found via *logarithmic map*. Given $\mathbf{X}$ and $\mathbf{Y}$, the tangent vector in $T_{\mathbf{X}}\mathrm{St}(n,p)$ for the geodesic from $\mathbf{X}$ to $\mathbf{Y}$ is

$$\mathrm{Log}_{\mathbf{X}}(\mathbf{Y}) = \mathbf{U}\arctan(\mathbf{\Sigma})\mathbf{V}^\intercal, \tag{1.29}$$

where $\mathbf{U}\mathbf{\Sigma}\mathbf{V}^\intercal$ is the compact SVD of $\Pi_{\mathbf{X}}\mathbf{Y}(\mathbf{X}^\intercal\mathbf{Y})^{-1}$ [93].

For a manifold $\mathcal{M} = \prod_{k=1}^{N} \mathrm{Gr}(n_k, p_k)$, a Cartesian product of N Grassmannians, elements are N-tuples of linear subspaces $y = (\mathcal{Y}_1, \ldots, \mathcal{Y}_N)^\intercal$, in turn represented by N-tuples of matrices $\mathbf{y} = (\mathbf{Y}_1, \ldots, \mathbf{Y}_N)^\intercal$. The tangent space at $y \in \mathcal{M}$ is the Cartesian product of tangent spaces $T_{\mathcal{Y}_k}\mathrm{Gr}(n_k, p_k)$. The inner product on $\mathcal{M}$ is $\langle \mathbf{x}, \mathbf{y} \rangle = \sum_{k=1}^{N} \langle \mathbf{X}_k, \mathbf{Y}_k \rangle$. All other required operations are performed component-wise using the operations defined for $\mathrm{Gr}(n_k, p_k)$.

Chapter 2

Linear and Nonlinear Preconditioning

We begin this chapter by considering the special case of linearly preconditioned optimization methods applied to convex quadratic objective functions. Given the success of these methods for the quadratic case we then generalize linear preconditioning strategies to incorporate nonlinear preconditioners for use in solving nonlinear optimization problems

$$\min \quad f(\mathbf{x}). \tag{2.1}$$

Finally, we adapt these methods to work for optimization problems defined over matrix manifolds.

2.1 Linearly Preconditioned Optimization Methods for Convex Quadratics

In this section we discuss the use of linearly preconditioned iterations for the solution of the convex quadratic minimization problem

$$\min_{\mathbf{x}} \quad \tfrac{1}{2}\mathbf{x}^\intercal \mathbf{A}\mathbf{x} - \mathbf{b}^\intercal \mathbf{x}. \tag{2.2}$$

The optimality equations of this problem are given by

$$\mathbf{g}(\mathbf{x}) = \mathbf{A}\mathbf{x} - \mathbf{b} = \mathbf{0}.$$

We consider optimization methods that solve $\mathbf{g}(\mathbf{x}) = \mathbf{0}$ by some form of fixed-point iteration. For example, one of the simplest choices for solving $\mathbf{g}(\mathbf{x}) = \mathbf{A}\mathbf{x} - \mathbf{b} = \mathbf{0}$ is Richardson

iteration [63]

$$\mathbf{x}_{k+1} = \mathbf{x}_k - \mathbf{g}(\mathbf{x}_k) = \mathbf{x}_k - (\mathbf{A}\mathbf{x}_k - \mathbf{b}),$$

which is equivalent to steepest descent with unit step length, and converges if $\|\mathbf{I} - \mathbf{A}\| < 1$.

Two different preconditioning strategies are described in § 2.1.1 and 2.1.2: we may either (i) apply a left-preconditioning matrix $\mathbf{P}$ to the optimality equations and solve the left-preconditioned system $\mathbf{PAx} = \mathbf{Pb}$; or (ii) introduce a change of variables $\mathbf{x} = \mathbf{Cz}$ and solve the transformed optimization problem $\widehat{f}(\mathbf{z})$. In § 2.1.3–2.1.5 we discuss how these strategies define preconditioned GMRES, CG, L-BFGS, and L-Broyden iterations.

2.1.1 Linear Left Preconditioning (LP)

Instead of solving the optimality equations $\mathbf{g}(\mathbf{x}) = \mathbf{0}$, we can apply Richardson iteration to the left-preconditioned optimality equations

$$\mathbf{P}\mathbf{g}(\mathbf{x}) = \mathbf{P}(\mathbf{A}\mathbf{x} - \mathbf{b}) = \mathbf{0},$$

to obtain

$$\mathbf{x}_{k+1} = \mathbf{x}_k - \mathbf{P}(\mathbf{A}\mathbf{x}_k - \mathbf{b}). \tag{2.3}$$

We take a step in direction $\mathbf{Pg}(\mathbf{x})$ instead of the gradient direction $\mathbf{g}(\mathbf{x})$, and we can interpret $\mathbf{Pg}(\mathbf{x})$ as the preconditioned gradient direction. In the optimization context, this form of left preconditioning works in general by replacing any occurrence of the gradient $\mathbf{g}(\mathbf{x})$ in the fixed-point iteration (such as Richardson) by the preconditioned gradient $\mathbf{Pg}(\mathbf{x})$ (i.e. we are applying the fixed-point iteration to $\mathbf{Pg}(\mathbf{x}) = \mathbf{0}$ instead of $\mathbf{g}(\mathbf{x}) = \mathbf{0}$).

Here $\mathbf{P}$ could be chosen to be the matrix from any of the stationary linear iterations commonly used as preconditioners, such as Gauss-Seidel (GS); successive over-relaxation (SOR); or, since we assume $\mathbf{A}$ to be SPD, symmetric GS (SGS) and symmetric SOR (SSOR). Using the matrix splitting $\mathbf{A} = \mathbf{D} + \mathbf{L} + \mathbf{U}$, SOR is equivalent to (2.3) with preconditioner

$$\mathbf{P} = \omega \left(\mathbf{D} + \omega \mathbf{L}\right)^{-1}$$

where $\omega \in (0, 2)$, with GS corresponding to the particular choice of $\omega = 1$. Similarly, the preconditioner matrix for SSOR is

$$\mathbf{P} = \left[\left(\mathbf{D} + \omega \mathbf{U}^\intercal\right) \frac{1}{\omega(2-\omega)} \mathbf{D}^{-1} \left(\mathbf{D} + \omega \mathbf{U}\right)\right]^{-1}, \tag{2.4}$$

with $\omega = 1$ corresponding to SGS.

2.1.2 Linear Transformation Preconditioning (TP)

By defining a linear change of variables $\mathbf{x} = \mathbf{C}\mathbf{z}$ for some nonsingular matrix $\mathbf{C}$, we may rewrite the optimization problem (2.1) as

$$\widehat{f}(\mathbf{z}) = f(\mathbf{C}\mathbf{z}). \tag{2.5}$$

We then apply our original optimization method (e.g., Richardson, CG or L-BFGS) to (2.5), and transform back to the $\mathbf{x}$ variables. In doing so, we employ

$$\nabla_{\mathbf{z}}\widehat{f}(\mathbf{z}) = \nabla_{\mathbf{z}} f(\mathbf{C}\mathbf{z}) = \mathbf{C}^{\intercal}\nabla_{\mathbf{x}} f(\mathbf{x}),$$

and observe that in the resulting iteration formula for $\mathbf{x}$, matrices $\mathbf{C}$ and $\mathbf{C}^{\intercal}$ will appear along with $\nabla_{\mathbf{x}} f(\mathbf{x}) = \mathbf{g}(\mathbf{x})$. In particular, any products $\mathbf{g}^{\intercal}(\mathbf{x})\mathbf{g}(\mathbf{x})$ in the iteration formula will be transformed to $\mathbf{g}^{\intercal}(\mathbf{x})\mathbf{C}\mathbf{C}^{\intercal}\mathbf{g}(\mathbf{x})$.

For the specific example of the convex quadratic minimization problem (2.2) and Richardson iteration, the transformed objective and gradient functions are

$$\widehat{f}(\mathbf{z}) = \frac{1}{2}\mathbf{z}^{\intercal}\mathbf{C}^{\intercal}\mathbf{A}\mathbf{C}\mathbf{z} - (\mathbf{C}^{\intercal}\mathbf{b})^{\intercal}\,\mathbf{z} \quad \text{and} \quad \widehat{\mathbf{g}}(\mathbf{z}) = \mathbf{C}^{\intercal}\mathbf{A}\mathbf{C}\mathbf{z} - \mathbf{C}^{\intercal}\mathbf{b},$$

with corresponding iteration

$$\mathbf{z}_{k+1} = \mathbf{z}_k - \left(\mathbf{C}^{\intercal}\mathbf{A}\mathbf{C}\mathbf{z}_k - \mathbf{C}^{\intercal}\mathbf{b}\right),$$

which gives, upon transforming back to $\mathbf{x}$,

$$\mathbf{x}_{k+1} = \mathbf{x}_k - \mathbf{C}\mathbf{C}^{\intercal}\left(\mathbf{A}\mathbf{x}_k - \mathbf{b}\right). \tag{2.6}$$

If we call the SPD matrix $\mathbf{C}\mathbf{C}^{\intercal}$ the preconditioner matrix $\mathbf{P}$ and take it to be the SGS or SSOR matrix, we get, for Richardson, the same result as the LP formula (2.3). Given an SPD preconditioner matrix $\mathbf{P}$, such as for SSOR, we can similarly compute the factorization $\mathbf{P} = \mathbf{C}\mathbf{C}^{\intercal}$; in the case of SSOR:

$$\mathbf{C} = \sqrt{\omega(2-\omega)}\,(\mathbf{D} + \omega\mathbf{U})^{-1}\,\mathbf{D}^{1/2}.$$

An important observation is that for more elaborate optimization methods such as L-BFGS, the LP and TP approaches may give different results. For example, any scalar product $\mathbf{g}^{\intercal}(\mathbf{x})\mathbf{g}(\mathbf{x})$ in the iteration formula will be transformed to $\mathbf{g}^{\intercal}(\mathbf{x})\mathbf{P}\mathbf{g}(\mathbf{x})$ in the TP approach, whereas it will become $\mathbf{g}^{\intercal}(\mathbf{x})\mathbf{P}^{\intercal}\mathbf{P}\mathbf{g}(\mathbf{x})$ in the LP approach. This difference may appear subtle, and intuitively the TP approach may appear preferable since it is more closely aligned with the original optimization problem, but we will see in the preliminary numerical results for the convex quadratic case, and in the general results after extending the approaches to nonlinear preconditioning, that both approaches may have their merits (corresponding also to the findings for nonlinearly preconditioned NCG in [34]).

2.1.3 Linearly Preconditioned GMRES

As described in [85], GMRES can be effectively combined with left, right or split preconditioning, in each case by applying GMRES (Algorithm 4) to the preconditioned linear system. Right preconditioning involves defining a change of variables $\mathbf{x} = \mathbf{P}\mathbf{z}$ to obtain the transformed system $\mathbf{APz} = \mathbf{b}$. When $\mathbf{P} = \mathbf{LU}$ (linear TP being a specific example) the split-preconditioned system is $\mathbf{UALz} = \mathbf{Ub}$, where $\mathbf{x} = \mathbf{Lz}$. There is generally little difference between the three preconditioning methods, though the different residual formulations may affect the stopping criteria, and when $\mathbf{A}$ is symmetric or nearly symmetric the split preconditioner may produce better results.

2.1.4 Linearly Preconditioned CG

Given that CG would require the matrix $\mathbf{PA}$ to be SPD for defining the weighted norm in which the error is minimized, in general the LP strategy is inappropriate, and thus we only consider the use of TP for CG. This derivation is well documented in the literature; see, for instance, [54]. Writing the CG iteration in terms of $\mathbf{z}$,

$$\begin{aligned} \mathbf{z}_{k+1} &= \mathbf{z}_k + \widehat{\alpha}_k \widehat{\mathbf{p}}_k, \\ \widehat{\mathbf{p}}_{k+1} &= -\widehat{\mathbf{g}}_{k+1} + \widehat{\beta}_{k+1} \widehat{\mathbf{p}}_k, \quad \widehat{\mathbf{p}}_0 = -\widehat{\mathbf{g}}_0, \end{aligned}$$

then converting back to $\mathbf{x}_k$, we obtain

$$\begin{aligned} \mathbf{x}_{k+1} &= \mathbf{x}_k + \alpha_k \mathbf{p}_k, \\ \mathbf{p}_{k+1} &= -\mathbf{P}\mathbf{g}_{k+1} + \widehat{\beta}_{k+1} \mathbf{p}_k, \quad \mathbf{p}_0 = -\mathbf{P}\mathbf{g}_0, \end{aligned} \tag{2.7}$$

as

$$\widehat{\alpha}_k = -\frac{\widehat{\mathbf{p}}_k^\intercal \widehat{\mathbf{g}}_k}{\widehat{\mathbf{p}}_k^\intercal \mathbf{C}^\intercal \mathbf{A} \mathbf{C} \widehat{\mathbf{p}}_k} = -\frac{\mathbf{p}_k^\intercal \mathbf{C}^{-\intercal} \mathbf{C}^\intercal \mathbf{g}_k}{\mathbf{p}_k^\intercal \mathbf{C}^{-\intercal} \mathbf{C}^\intercal \mathbf{A} \mathbf{C} \mathbf{C}^{-1} \mathbf{p}_k} = -\frac{\mathbf{p}_k^\intercal \mathbf{g}_k}{\mathbf{p}_k^\intercal \mathbf{A} \mathbf{p}_k} = \alpha_k$$

and

$$\widehat{\beta}_{k+1} = \frac{\widehat{\mathbf{g}}_{k+1}^\intercal \widehat{\mathbf{g}}_{k+1}}{\widehat{\mathbf{g}}_k^\intercal \widehat{\mathbf{g}}_k} = \frac{\mathbf{g}_{k+1}^\intercal \mathbf{P} \mathbf{g}_{k+1}}{\mathbf{g}_k^\intercal \mathbf{P} \mathbf{g}_k}.$$

2.1.5 Linearly Preconditioned QN

L-BFGS Update

LP L-BFGS requires the direct replacement of each gradient $\mathbf{g}_k$ with the left-preconditioned gradient $\mathbf{P}\mathbf{g}_k$ in the components $\mathbf{Y}_k$, $\mathbf{D}_k$, $\mathbf{R}_k$, and γ_k of (1.18), and in computing the QN direction $\mathbf{p}_k$.

To derive TP L-BFGS, we write (1.18) for $\widehat{f}(\mathbf{z})$ as

$$\widehat{\mathbf{H}}_k = \widehat{\mathbf{H}}_0^{(k)} + \begin{bmatrix} \widehat{\mathbf{S}}_k & \widehat{\mathbf{H}}_0^{(k)}\widehat{\mathbf{Y}}_k \end{bmatrix} \begin{bmatrix} \widehat{\mathbf{R}}_k^{-\intercal}(\widehat{\mathbf{D}}_k + \widehat{\mathbf{Y}}_k^{\intercal}\widehat{\mathbf{H}}_0^{(k)}\widehat{\mathbf{Y}}_k)\widehat{\mathbf{R}}_k^{-1} & -\widehat{\mathbf{R}}_k^{-\intercal} \\ -\widehat{\mathbf{R}}_k^{-1} & \mathbf{0} \end{bmatrix} \begin{bmatrix} \widehat{\mathbf{S}}_k^{\intercal} \\ \widehat{\mathbf{Y}}_k^{\intercal}\widehat{\mathbf{H}}_0^{(k)} \end{bmatrix},$$

where $\widehat{\mathbf{H}}_0^{(k)} = \widehat{\gamma}_k\mathbf{I}$. By examining definitions (1.12), (1.13), (1.19), (1.21), and (1.22), we obtain the following relationships between original and transformed quantities:

$$\widehat{\mathbf{S}}_k = \mathbf{C}^{-1}\mathbf{S}_k, \quad \widehat{\mathbf{Y}}_k = \mathbf{C}^{\intercal}\mathbf{Y}_k, \quad \widehat{\mathbf{D}}_k = \mathbf{D}_k, \quad \widehat{\mathbf{R}}_k = \mathbf{R}_k,$$

and

$$\widehat{\gamma}_k = \frac{\mathbf{s}_{k-1}^{\intercal}\mathbf{y}_{k-1}}{\mathbf{y}_{k-1}^{\intercal}\mathbf{P}\mathbf{y}_{k-1}}.$$

The transformed QN update equation

$$\mathbf{z}_k = \mathbf{z}_{k-1} + \alpha_k\widehat{\mathbf{H}}_k\widehat{\mathbf{g}}(\mathbf{z}_{k-1})$$

corresponds to

$$\mathbf{x}_k = \mathbf{x}_{k-1} + \alpha_k\mathbf{C}\widehat{\mathbf{H}}_k\mathbf{C}^{\intercal}\mathbf{g}(\mathbf{x}_{k-1}).$$

Computing $\mathbf{H}_k := \mathbf{C}\widehat{\mathbf{H}}_k\mathbf{C}^{\intercal}$, we have

$$\mathbf{H}_k = \widehat{\gamma}_k\mathbf{P} + \begin{bmatrix} \mathbf{S}_k & \widehat{\gamma}_k\mathbf{P}\mathbf{Y}_k \end{bmatrix} \begin{bmatrix} \mathbf{R}_k^{-\intercal}(\mathbf{D}_k + \widehat{\gamma}_k\mathbf{Y}_k^{\intercal}\mathbf{P}\mathbf{Y}_k)\mathbf{R}_k^{-1} & -\mathbf{R}_k^{-\intercal} \\ -\mathbf{R}_k^{-1} & \mathbf{0} \end{bmatrix} \begin{bmatrix} \mathbf{S}_k^{\intercal} \\ \widehat{\gamma}_k\mathbf{Y}_k^{\intercal}\mathbf{P} \end{bmatrix}. \tag{2.8}$$

Solving the preconditioned problem $\widehat{f}(\mathbf{z})$ using L-BFGS is equivalent to solving $f(\mathbf{x})$ with L-BFGS where $\mathbf{H}_0^{(k)} = \widehat{\gamma}_k\mathbf{P}$, which is essentially the same preconditioning strategy described for BFGS in [72, § 10.7], except they omit the scaling factor $\widehat{\gamma}_k$.

L-Broyden Update

Similar to L-BFGS, the LP L-Broyden update simply requires the replacement of each gradient $\mathbf{g}_k$ with the left-preconditioned gradient $\mathbf{P}\mathbf{g}_k$ in the component $\mathbf{Y}_k$ of (1.11) and in computing the QN direction $\mathbf{p}_k$.

To derive the TP L-Broyden update we write (1.10) in terms of $\mathbf{z}$ to obtain:

$$\widehat{\mathbf{A}}_k^{-1} = \left[\widehat{\mathbf{A}}_0^{(k)}\right]^{-1} - \left(\left[\widehat{\mathbf{A}}_0^{(k)}\right]^{-1}\widehat{\mathbf{Y}}_k - \widehat{\mathbf{S}}_k\right)\left(\widehat{\mathbf{M}}_k + \widehat{\mathbf{S}}_k^{\intercal}\left[\widehat{\mathbf{A}}_0^{(k)}\right]^{-1}\widehat{\mathbf{Y}}_k\right)^{-1}\widehat{\mathbf{S}}_k^{\intercal}\left[\widehat{\mathbf{A}}_0^{(k)}\right]^{-1},$$

where $\left[\widehat{\mathbf{A}}_0^{(k)}\right]^{-1} = \widehat{\gamma}_k \mathbf{I}$. Recalling the definition (1.14) for $i > j$

$$(\widehat{\mathbf{M}}_k)_{i,j} = -\widehat{\mathbf{s}}_{i-1}^\intercal \widehat{\mathbf{s}}_{j-1} = -\mathbf{s}_{i-1}^\intercal \left(\mathbf{C}\mathbf{C}^\intercal\right)^{-1} \mathbf{s}_{j-1} = \mathbf{g}_{i-1}^\intercal \mathbf{s}_{j-1},$$

where the last equality follows from (2.6). As before,

$$\mathbf{x}_k = \mathbf{x}_{k-1} + \alpha_k \mathbf{C}\widehat{\mathbf{A}}_k^{-1}\mathbf{C}^\intercal \mathbf{g}(\mathbf{x}_{k-1}),$$

thus the inverse matrix update is

$$\mathbf{A}_k^{-1} := \mathbf{C}\widehat{\mathbf{A}}_k^{-1}\mathbf{C}^\intercal = \widehat{\gamma}_k \mathbf{P} - \left(\widehat{\gamma}_k \mathbf{P}\mathbf{Y}_k - \mathbf{S}_k\right)\left(\widehat{\mathbf{M}}_k + \widehat{\gamma}_k \mathbf{S}_k^\intercal \mathbf{Y}_k\right)^{-1} \mathbf{S}_k^\intercal \widehat{\gamma}_k. \tag{2.9}$$

Compared to the L-BFGS case, this is not a full replacement of $\left[\widehat{\mathbf{A}}_0^{(k)}\right]^{-1}$ by $\widehat{\gamma}_k \mathbf{P}$: only two of the instances involve $\mathbf{P}$, the remaining two only require $\widehat{\gamma}_k$.

2.1.6 Illustration of Linear LP and TP Methods

To illustrate the different preconditioning possibilities, we solve (2.2) corresponding to a finite difference discretization of the 2D Poisson equation

$$u_{xx} + u_{yy} = 2[(1-6x^2)y^2(1-y^2) + (1-6y^2)x^2(1-x^2)], \quad (x,y) \in [0,1] \times [0,1],$$

with homogeneous Dirichlet boundary conditions and mesh spacing $dx = dy = 10^{-2}$, resulting in a problem with 9801 unknowns. We solve this problem using GMRES, CG, L-BFGS, L-Broyden, and their preconditioned variants using SGS or SSOR, the latter with $\omega = 1.9$. The QN methods use a window size of $m = 5$. For all methods but GMRES we use the exact step length for quadratic problems from Algorithm 6. To precondition GMRES, L-BFGS and L-Broyden we consider both LP and TP strategies. Residuals are scaled by the number of unknowns, and preconditioned GMRES results are scaled to have the same initial residual as the rest of the methods. These results are presented in Figure 2.1, the left panel containing results for SGS preconditioning and the right for SSOR preconditioning.

Non-preconditioned CG and L-BFGS have essentially identical convergence histories, as expected from the discussion of § 1.3.3, whereas L-Broyden in fact does not converge for this problem, illustrated by the irregular oscillations of the scaled residual norm value. The GMRES plots are slightly better than the CG and L-BFGS plots for both preconditioned and non preconditioned variants, with very little difference between the TP and LP GMRES

results. The TP-L-BFGS overlap the PCG plots for both SGS and SSOR. Larger differences are observed for the LP-L-BFGS methods, more-so for SGS than SSOR, indicating that preconditioning based on variable transformation is the more effective approach. For L-Broyden it is interesting to observe that preconditioning enables the iterations to converge, although the residual curve is still very oscillatory for SGS based preconditioning. When using SGS the TP-BROY method is somewhat more effective than the LP version, whereas for SSOR this is reversed, with LP-LBROY coming close to the PCG results. Finally, echoing the fact that the SSOR convergence rate is provably better than the convergence rate of SGS [108, 109], we see that SSOR is clearly a better choice of preconditioner.

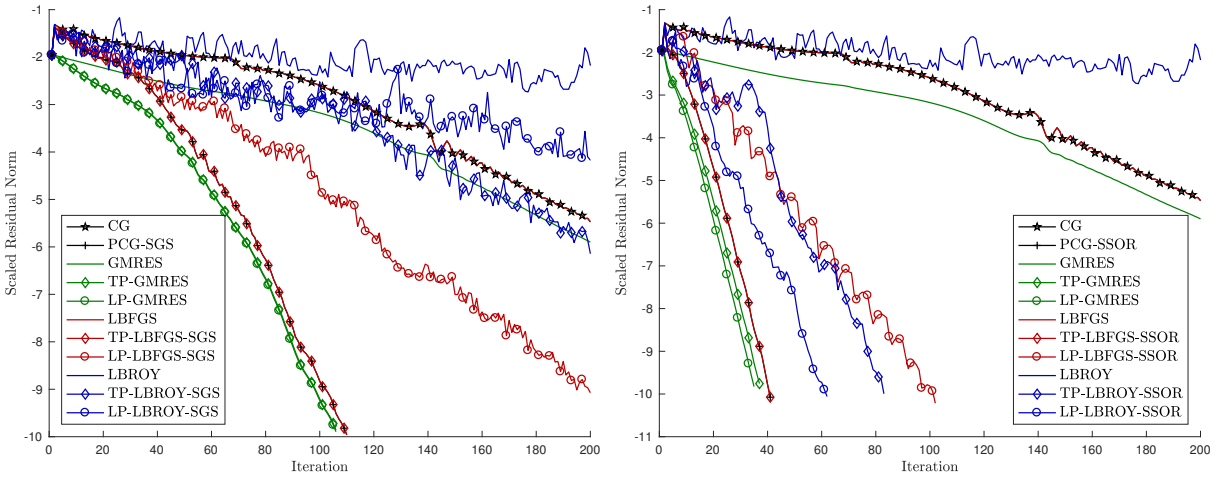

Figure 2.1: Results for GMRES, CG and QN methods using SGS (L) and SSOR (R) based preconditioners.

2.2 Nonlinear Preconditioning Strategies

Having described linear preconditioning strategies for minimizing (2.2), we now consider how to generalize the linear LP and TP strategies to nonlinear preconditioners for more general nonlinear optimization problems (2.1). We first discuss nonlinear preconditioning in general, before describing nonlinearly preconditioned NCG, NGMRES, and QN methods.

2.2.1 Nonlinear Left Preconditioning (LP)

To generalize the linear LP approach to nonlinear preconditioning, we replace the optimality equation $\mathbf{g}(\mathbf{x}) = \mathbf{0}$ with a nonlinearly preconditioned optimality equation

$$\mathcal{P}(\mathbf{g}; \mathbf{x}) = \mathbf{0}.$$

We require solutions of $\mathbf{g}(\mathbf{x}) = \mathbf{0}$ to also be solutions of $\mathcal{P}(\mathbf{g}; \mathbf{x}) = \mathbf{0}$. The notation $\mathcal{P}(\mathbf{g}; \mathbf{x})$ emphasizes that $\mathcal{P}$ is related to solving $\mathbf{g}(\mathbf{x}) = \mathbf{0}$. In the convex quadratic case, $\mathcal{P}(\mathbf{g}; \mathbf{x}) = \mathbf{P}\mathbf{g}(\mathbf{x}) = \mathbf{P}(\mathbf{A}\mathbf{x} - \mathbf{b})$. In the nonlinear case $\mathcal{P}(\mathbf{g}; \mathbf{x})$ is generally derived from a nonlinear fixed-point equation $\mathbf{x} = \mathcal{Q}(\mathbf{g}; \mathbf{x})$, in which case we write

$$\mathcal{P}(\mathbf{g}; \mathbf{x}) := \mathbf{x} - \mathcal{Q}(\mathbf{g}; \mathbf{x}). \tag{2.10}$$

Given an iteration $\mathbf{x}_{k+1} = \mathcal{Q}(\mathbf{g}; \mathbf{x}_k) = \mathbf{x}_k - \mathcal{P}(\mathbf{g}; \mathbf{x}_k)$, we see that

$$\mathcal{P}(\mathbf{g}; \mathbf{x}_k) = \mathbf{x}_k - \mathbf{x}_{k+1}$$

is the (negative) update direction provided by the iteration, and for a suitable $\mathcal{Q}(\mathbf{g}; \mathbf{x}_k)$ it should be an improvement on the direction provided by the gradient, $\mathbf{g}(\mathbf{x})$. In analogy with the linear case where $\mathcal{P}(\mathbf{g}; \mathbf{x}) = \mathcal{P}\mathbf{g}(\mathbf{x})$, we interpret $\mathcal{P}(\mathbf{g}; \mathbf{x})$ as the preconditioned gradient direction. By applying an optimization method with iteration $\mathbf{x}_{k+1} = \mathcal{M}(\mathbf{g}; \mathbf{x}_k)$ to $\mathcal{P}(\mathbf{g}; \mathbf{x}) = \mathbf{x} - \mathcal{Q}(\mathbf{g}; \mathbf{x}) = \mathbf{0}$ instead of to $\mathbf{g}(\mathbf{x}) = \mathbf{0}$, we obtain the nonlinearly left-preconditioned optimization update

$$\mathbf{x}_{k+1} = \mathcal{M}(\mathcal{P}(\mathbf{g}; \cdot); \mathbf{x}_k).$$

This means, in practice, that all occurrences of $\mathbf{g}(\mathbf{x})$ in $\mathcal{M}$ are replaced by $\mathcal{P}(\mathbf{g}; \mathbf{x})$ in the LP approach, as in the case of nonlinear left-preconditioning for nonlinear equation systems [18]. An important difference in the optimization context, however, is that we continue using the original $f(\mathbf{x})$ and $\mathbf{g}(\mathbf{x})$ in determining the line-search step α for methods like CG or L-BFGS, so the gradient $\mathbf{g}(\mathbf{x})$ used in the line-search is not replaced by $\mathcal{P}(\mathbf{g}; \mathbf{x})$.

2.2.2 Nonlinear Transformation Preconditioning (TP)

To extend linear transformation preconditioning to nonlinear preconditioning, consider the iteration formulas derived in § 2.1 using the linear change of variable $\mathbf{x} = \mathbf{C}\mathbf{z}$. All occurrences of $\mathbf{P} = \mathbf{C}\mathbf{C}^\intercal$ in the resulting iteration formulas for $\mathbf{x}$ appear in front of $\mathbf{g}(\mathbf{x})$, and we simply replace the linearly preconditioned gradients $\mathbf{P}\mathbf{g}(\mathbf{x})$ by the nonlinearly

preconditioned gradients $\mathcal{P}(\mathbf{g};\mathbf{x})$. This is a natural extension of linear TP to the nonlinear case, with the nonlinear extension reducing to the usual linear preconditioning for nonlinear optimization when $\mathcal{P}(\mathbf{g};\mathbf{x})$ is chosen as $\mathbf{C}\mathbf{C}^\intercal\mathbf{g}(\mathbf{x})$ [54, 72], and, in the specific case of the NCG method, to the well-known formulas for linearly preconditioned CG for SPD linear systems when the objective function is convex quadratic [34].

2.2.3 Multiplicative Composition

A third variant for nonlinear preconditioning, which applies to NGMRES, is the multiplicative composition of solvers, in which method $\mathcal{N}$ is used as a nonlinear accelerator of the iteration $\mathbf{x}_{k+1} = \mathcal{Q}(\mathbf{g};\mathbf{x}_k)$. As discussed in [18], multiplicative composition of solvers $\mathcal{N}$ and $\mathcal{Q}$ can be written as

$$\mathbf{x}_{k+1} = \mathcal{N}(\mathbf{g};\mathcal{Q}(\mathbf{g};\mathbf{x}_k)).$$

An example of multiplicative composition is presented in the following subsection, where NGMRES is used to accelerate a fixed-point iteration.

2.2.4 Nonlinearly Preconditioned NGMRES (NPNGMRES)

NGMRES, as described in Algorithm 5, can be written as $\mathbf{x}_{k+1} = \mathcal{N}(\mathbf{g};\mathcal{Q}(\mathbf{g};\mathbf{x}_k))$, where $\mathcal{Q}(\mathbf{g};\mathbf{x}_k) = \mathbf{x}_k - \gamma_k\mathbf{g}(\mathbf{x}_k)$ is the steepest descent preconditioner. Rather than generate the tentative iterate $\overline{\mathbf{x}}_{k+1}$ by gradient descent, we can use a more general nonlinear preconditioner $\mathcal{Q}$ and replace $\overline{\mathbf{x}}_{k+1} = \mathbf{x}_k - \alpha_k\mathbf{g}(\mathbf{x}_k)$ with $\overline{\mathbf{x}}_{k+1} = \mathcal{Q}(\mathbf{g};\mathbf{x}_k)$. This is the approach that has been used in [32, 45, 104, 105, 107]. As explained in [32, 105], when $\mathbf{g}(\mathbf{x}) = \mathbf{A}\mathbf{x} - \mathbf{b}$ and $\overline{\mathbf{x}}_{k+1} = \mathbf{x}_k - \mathbf{P}(\mathbf{A}\mathbf{x}_k - \mathbf{b})$ (i.e., $\mathcal{Q}(\mathbf{g};\mathbf{x}_k) = \mathbf{x}_k - \mathbf{P}(\mathbf{A}\mathbf{x}_k - \mathbf{b})$), NPNGMRES reduces to right-preconditioned GMRES, i.e. GMRES applied to $\mathbf{A}\mathbf{P}\mathbf{y} = \mathbf{b}$ where $\mathbf{P}\mathbf{y} = \mathbf{x}$. One iteration of NPNGMRES is described in Algorithm 10.

2.2.5 Nonlinearly Preconditioned NCG (NPNCG)

Ideas of using a nonlinear preconditioner with NCG have been around since the 1970s [10, 27, 73], but it has not been widely explored. The paper [34] systematically studied a NPNCG iteration in the optimization context. In the nonlinear LP framework (as in [18]), given the preconditioning iteration $\mathbf{x}_{k+1} = \mathcal{Q}(\mathbf{g};\mathbf{x}_k)$, we define

$$\overline{\mathbf{g}}_k := \mathcal{P}(\mathbf{g};\mathbf{x}_k) = \mathbf{x}_k - \mathcal{Q}(\mathbf{g};\mathbf{x}_k),$$

Algorithm 10 Nonlinearly Preconditioned NGMRES

procedure NPNGMRES($\mathbf{g}, \mathbf{x}_{k-m+1}, \ldots, \mathbf{x}_k$)
  $\overline{\mathbf{x}}_{k+1} = \mathcal{Q}(\mathbf{g}; \mathbf{x}_k)$
  Solve $\left\| \mathbf{g}(\overline{\mathbf{x}}_{k+1}) + \sum_{j=k-m+1}^{k} \beta_j (\mathbf{g}(\overline{\mathbf{x}}_{k+1}) - \mathbf{g}(\mathbf{x}_j)) \right\|_2$ for $\boldsymbol{\beta}$
  $\mathbf{p}_k = \sum_{j=k-m+1}^{k} \beta_j (\overline{\mathbf{x}}_{k+1} - \mathbf{x}_j)$
  $\mathbf{x}_{k+1} = \overline{\mathbf{x}}_{k+1} + \alpha_k \mathbf{p}_k$ ▷ α_k determined by line-search
  return $\mathbf{x}_{k+1}$
end procedure

and replace every instance of $\mathbf{g}_k$ by $\overline{\mathbf{g}}_k$ to obtain the LP NPNCG iteration

$$\begin{aligned} \mathbf{x}_{k+1} &= \mathbf{x}_k + \alpha_k \mathbf{p}_k, \\ \mathbf{p}_{k+1} &= -\overline{\mathbf{g}}_{k+1} + \overline{\beta}_{k+1} \mathbf{p}_k, \quad \mathbf{p}_0 = -\overline{\mathbf{g}}_0. \end{aligned} \tag{2.11}$$

The corresponding β formulas are:

$$\widetilde{\beta}_{k+1}^{\mathrm{PR}} = \frac{\overline{\mathbf{g}}_{k+1}^{\intercal} \overline{\mathbf{y}}_k}{\overline{\mathbf{g}}_k^{\intercal} \overline{\mathbf{g}}_k}, \tag{2.12}$$

$$\widetilde{\beta}_{k+1}^{\mathrm{HS}} = \frac{\overline{\mathbf{g}}_{k+1}^{\intercal} \overline{\mathbf{y}}_k}{\overline{\mathbf{y}}_k^{\intercal} \mathbf{p}_k}, \tag{2.13}$$

$$\widetilde{\beta}_{k+1}^{\mathrm{HZ}} = \left(\overline{\mathbf{y}}_k - 2\mathbf{p}_k \frac{\|\overline{\mathbf{y}}_k\|^2}{\overline{\mathbf{y}}_k^{\intercal} \mathbf{p}_k} \right)^{\intercal} \frac{\overline{\mathbf{g}}_{k+1}}{\overline{\mathbf{y}}_k^{\intercal} \mathbf{p}_k}, \tag{2.14}$$

where $\overline{\mathbf{y}}_k = \overline{\mathbf{g}}_{k+1} - \overline{\mathbf{g}}_k$.

To obtain the TP NPNCG iteration, we replace each $\mathbf{P}\mathbf{g}_k$ in (2.7) with our preconditioner direction $\overline{\mathbf{g}}_k$, obtaining (2.11) again. To obtain the TP β formulas corresponding to (1.6–1.8), we recall that the products of $\mathbf{g}^{\intercal}\mathbf{g}$ transform to $\mathbf{g}^{\intercal}\mathbf{C}\mathbf{C}^{\intercal}\mathbf{g}$ under the linear transformation $\mathbf{x} = \mathbf{C}\mathbf{z}$. This becomes $\mathbf{g}^{\intercal}\overline{\mathbf{g}}$ when using the nonlinear TP preconditioner, resulting in the following alternatives that incorporate both $\mathbf{g}_k$ and $\overline{\mathbf{g}}_k$:

$$\widehat{\beta}_{k+1}^{\mathrm{PR}} = \frac{\mathbf{g}_{k+1}^{\intercal} \overline{\mathbf{y}}_k}{\mathbf{g}_k^{\intercal} \overline{\mathbf{g}}_k}, \tag{2.15}$$

$$\widehat{\beta}_{k+1}^{\mathrm{HS}} = \frac{\mathbf{g}_{k+1}^{\intercal} \overline{\mathbf{y}}_k}{\mathbf{y}_k^{\intercal} \mathbf{p}_k}, \tag{2.16}$$

$$\widehat{\beta}_{k+1}^{\mathrm{HZ}} = \frac{\mathbf{g}_{k+1}^{\intercal} \overline{\mathbf{y}}_k}{\mathbf{y}_k^{\intercal} \mathbf{p}_k} - 2\mathbf{p}_k^{\intercal} \mathbf{g}_{k+1} \frac{\mathbf{y}_k^{\intercal} \overline{\mathbf{y}}_k}{(\mathbf{y}_k^{\intercal} \mathbf{p}_k)^2}. \tag{2.17}$$

The LP and TP versions of NPNCG have been considered previously in [34]. It was observed there numerically that the TP $\widehat{\beta}$ update formulas can give better results than the LP $\widetilde{\beta}$ formulas obtained from nonlinear left-preconditioning.

2.2.6 Nonlinearly Preconditioned QN (NPQN)

L-BFGS Update

LP NPQN iterations are obtained by applying QN update formulas to $\mathcal{P}(\mathbf{g};\mathbf{x}) = 0$. For L-BFGS we continue to use (1.18) as our inverse Hessian approximation, defining $\overline{\mathbf{g}}_k = \mathcal{P}(\mathbf{g};\mathbf{x}_k)$ and $\overline{\mathbf{y}}_k = \overline{\mathbf{g}}_{k+1} - \overline{\mathbf{g}}_k$. We replace each instance of $\mathbf{g}_k$ with $\overline{\mathbf{g}}_k$ and each instance of $\mathbf{y}_k$ with $\overline{\mathbf{y}}_k$ (preserving the symmetry of $\mathbf{H}_k$) to obtain

$$\widetilde{\mathbf{H}}_k = \widetilde{\gamma}_k + \begin{bmatrix}\mathbf{S}_k & \widetilde{\gamma}_k\overline{\mathbf{Y}}_k\end{bmatrix}\begin{bmatrix}\overline{\mathbf{R}}_k^{-\mathsf{T}}(\overline{\mathbf{D}}_k + \widetilde{\gamma}_k\overline{\mathbf{Y}}_k^{\mathsf{T}}\overline{\mathbf{Y}}_k)\overline{\mathbf{R}}_k^{-1} & -\overline{\mathbf{R}}_k^{-\mathsf{T}} \\ -\overline{\mathbf{R}}_k^{-1} & \mathbf{0}\end{bmatrix}\begin{bmatrix}\mathbf{S}_k^{\mathsf{T}} \\ \widetilde{\gamma}_k\overline{\mathbf{Y}}_k^{\mathsf{T}}\end{bmatrix}, \tag{2.18}$$

where

$$\overline{\mathbf{Y}}_k = [\overline{\mathbf{y}}_{k-m}\,|\,\overline{\mathbf{y}}_{k-m+1}\,|\,\cdots\,|\,\overline{\mathbf{y}}_{k-1}], \tag{2.19}$$

$$\overline{\mathbf{D}}_k = \operatorname{diag}[\mathbf{s}_{k-m}^{\mathsf{T}}\overline{\mathbf{y}}_{k-m}, \ldots, \mathbf{s}_{k-1}^{\mathsf{T}}\overline{\mathbf{y}}_{k-1}], \tag{2.20}$$

$$(\overline{\mathbf{R}}_k)_{i,j} = \begin{cases}(\mathbf{s}_{k-m-1+i})^{\mathsf{T}}(\overline{\mathbf{y}}_{k-m-1+j}) & \text{if } i \le j \\ 0 & \text{otherwise}\end{cases} \tag{2.21}$$

and

$$\widetilde{\gamma}_k = \frac{\mathbf{s}_{k-1}^{\mathsf{T}}\overline{\mathbf{y}}_{k-1}}{\overline{\mathbf{y}}_{k-1}^{\mathsf{T}}\overline{\mathbf{y}}_{k-1}}. \tag{2.22}$$

If we instead start from the linear TP L-BFGS update (2.8) and replace each $\mathbf{P}\mathbf{g}_k$ with $\overline{\mathbf{g}}_k$ and each $\mathbf{P}\mathbf{y}_k$ with $\overline{\mathbf{y}}_k$, we obtain for the TP NPQN iteration

$$\widehat{\mathbf{H}}_k(\mathbf{g}_k) = \widehat{\gamma}_k\mathcal{P}(\mathbf{g};\mathbf{x}_k) + \begin{bmatrix}\mathbf{S}_k & \widehat{\gamma}_k\overline{\mathbf{Y}}_k\end{bmatrix}\begin{bmatrix}\mathbf{R}_k^{-\mathsf{T}}(\mathbf{D}_k + \widehat{\gamma}_k\mathbf{Y}_k^{\mathsf{T}}\overline{\mathbf{Y}}_k)\mathbf{R}_k^{-1} & -\mathbf{R}_k^{-\mathsf{T}} \\ -\mathbf{R}_k^{-1} & \mathbf{0}\end{bmatrix}\begin{bmatrix}\mathbf{S}_k^{\mathsf{T}} \\ \widehat{\gamma}_k\overline{\mathbf{Y}}_k^{\mathsf{T}}\end{bmatrix}\mathbf{g}_k, \tag{2.23}$$

where

$$\widehat{\gamma}_k = \frac{\mathbf{s}_{k-1}^{\mathsf{T}}\mathbf{y}_{k-1}}{\mathbf{y}_{k-1}^{\mathsf{T}}\overline{\mathbf{y}}_{k-1}}. \tag{2.24}$$

The quasi-Newton search direction $\mathbf{p}_k = -\widetilde{\mathbf{H}}_k\overline{\mathbf{g}}_k$ for LP, and $\mathbf{p}_k = -\widehat{\mathbf{H}}_k(\mathbf{g}_k)$ for TP.

L-Broyden Update

To simplify equations we assume the initial approximation of the inverse Jacobian has the form $[\mathbf{A}_0^{(k)}]^{-1} = \theta_k \mathbf{I}$, for some scaling factor θ_k. Similar to L-BFGS, the nonlinear LP variant is obtained by replacing $\mathbf{g}_k$ with $\overline{\mathbf{g}}_k = \mathcal{P}(\mathbf{g}; \mathbf{x}_k)$ throughout (1.11) and Algorithm 8, resulting in

$$\widetilde{\mathbf{A}}_k^{-1} = \theta_k \left(\mathbf{I} - \left(\theta_k \overline{\mathbf{Y}}_k - \mathbf{S}_k \right) \left(\mathbf{M}_k + \mathbf{S}_k^\intercal \theta_k \overline{\mathbf{Y}}_k \right)^{-1} \mathbf{S}_k^\intercal \right). \tag{2.25}$$

The idea of applying Broyden's method to a fixed point equation has previously been discussed in [45], though in the context of nonlinear systems of equations rather than optimization.

If we instead take the linear TP L-Broyden update (2.9) into consideration and replace $\mathbf{P}\mathbf{g}_k$ with $\overline{\mathbf{g}}_k$ and $\mathbf{P}\mathbf{y}_k$ with $\overline{\mathbf{y}}_k$, we obtain the operator

$$\widehat{\mathbf{A}}_k^{-1}(\mathbf{g}_k) = \theta_k \left(\mathcal{P}(\mathbf{g}_k; \mathbf{x}_{k+1}) - \left(\theta_k \overline{\mathbf{Y}}_k - \mathbf{S}_k \right) \left(\overline{\mathbf{M}}_k + \theta_k \mathbf{S}_k^\intercal \mathbf{Y}_k \right)^{-1} \mathbf{S}_k^\intercal \mathbf{g}_k \right), \tag{2.26}$$

where

$$(\overline{\mathbf{M}}_k)_{i,j} = \begin{cases} \mathbf{g}_{i-1}^\intercal \mathbf{s}_{j-1} & \text{if } i > j \\ 0 & \text{otherwise} \end{cases}. \tag{2.27}$$

As a final note, both (2.23) and (2.26) combine information from the gradient and preconditioner directions, resulting in some additional storage and computational costs when compared to the left-preconditioning approaches of (2.18) and (2.25).

2.3 Nonlinearly Preconditioned Optimization Methods on Matrix Manifolds

Bold lowercase letters now represent n-tuples of matrices; e.g., $\mathbf{x}_k = (\mathbf{A}_k^{(1)}, \ldots, \mathbf{A}_k^{(n)})^\intercal$. We define the preconditioner direction to be $\overline{\mathbf{g}}_k = -\mathrm{Log}_{\mathbf{x}_k}(\mathcal{Q}(\mathbf{g}; \mathbf{x}_k))$ (see (1.29)), the negative of the tangent vector at $\mathbf{x}_k$ defining the geodesic to $\mathcal{Q}(\mathbf{g}; \mathbf{x}_k)$.

Algorithm 11 Manifold NPNGMRES

procedure MANNPNGMRES(f,$\mathbf{g}, \mathbf{x}_{k-m+1}, \ldots, \mathbf{x}_k$)
  $\overline{\mathbf{x}}_{k+1} = \mathcal{Q}(\mathbf{g}; \mathbf{x}_k)$
  Solve $\left\| \mathbf{g}(\overline{\mathbf{x}}_{k+1}) + \sum_{j=k-m+1}^{k} \beta_j (\mathbf{g}(\overline{\mathbf{x}}_{k+1}) - \mathcal{T}_{\overline{\mathbf{x}}_{k+1}}(\mathbf{g}(\mathbf{x}_j))) \right\|_2$ for $\boldsymbol{\beta}$
  $\mathbf{p}_k = -\sum_{j=k-m+1}^{k} \beta_j \mathrm{Log}_{\overline{\mathbf{x}}_{k+1}}(\mathbf{x}_j)$
  $\mathbf{x}_{k+1} = R_{\overline{\mathbf{x}}_{k+1}}(\alpha_k \mathbf{p}_k)$ ▷ α_k determined by line-search
  return $\mathbf{x}_{k+1}$
end procedure

2.3.1 Manifold NPNGMRES

A number of changes are required to adapt NPNGMRES to the manifold context. To linearize $\mathbf{g}(\mathbf{x}) = \operatorname{grad} f(\mathbf{x})$, the Riemannian gradient of $f(\mathbf{x})$, we use [4, 40]

$$\mathbf{g}(\widehat{\mathbf{x}}_{k+1}) \approx \mathbf{g}(\overline{\mathbf{x}}_{k+1}) + \sum_{j=k-m+1}^{k} \beta_j \mathrm{Hess}\, f(\overline{\mathbf{x}}_{k+1})[\boldsymbol{\xi}_j], \tag{2.28}$$

where $\boldsymbol{\xi}_j = -\mathrm{Log}_{\overline{\mathbf{x}}_{k+1}}(\mathbf{x}_j)$. This requires a known expression for the Riemanning Hessian, $\mathrm{Hess}\, f(\mathbf{x})$, as well as $k+1$ evaluations of $\mathrm{Hess}\, f(\mathbf{x})$ applied to a tangent vector. We may approximate the action of $\mathrm{Hess}\, f(\mathbf{x})$ by adapting the approach from [32]:

$$\mathrm{Hess}\, f(\overline{\mathbf{x}}_{k+1})[\boldsymbol{\xi}_j] \approx \mathbf{g}(\overline{\mathbf{x}}_{k+1}) - \widetilde{\mathcal{T}}_{\overline{\mathbf{x}}_{k+1}}(\mathbf{g}(\mathbf{x}_j)), \tag{2.29}$$

where vector transport (see (1.28)) is applied to the past gradient. The search direction $\mathbf{p}_{k+1}$ is computed as the linear combination of tangent vectors at $\overline{\mathbf{x}}_{k+1}$:

$$\mathbf{p}_{k+1} = -\sum_{j=k-m+1}^{k} \beta_j \mathrm{Log}_{\overline{\mathbf{x}}_{k+1}}(\mathbf{x}_j).$$

The line search is carried out along a retraction starting at $\overline{\mathbf{x}}_{k+1}$ in the direction of $\mathbf{p}_{k+1}$. This process is described in Algorithm 11.

2.3.2 Manifold NPNCG

The manifold NPNCG iteration is

$$\begin{aligned} \mathbf{x}_{k+1} &= R_{\mathbf{x}_k}(\alpha \mathbf{p}_k), \\ \mathbf{p}_{k+1} &= -\overline{\mathbf{g}}_{k+1} + \overline{\beta}_{k+1} \mathcal{T}_{\mathbf{x}_{k+1}}(\mathbf{p}_k), \quad \mathbf{p}_0 = -\overline{\mathbf{g}}_0. \end{aligned} \tag{2.30}$$

The line search is carried out along the retraction curve $R_{\mathbf{x}_k}(\cdot)$ (see (1.26)), and $\mathbf{p}_{k+1}$ requires vector transport of $\mathbf{p}_k$ using $\widetilde{\mathcal{T}}_{\mathbf{x}_{k+1}}(\cdot)$. The $\widetilde{\beta}$ formulas become

$$\widetilde{\beta}_{k+1}^{\text{PR}} = \frac{\langle \overline{\mathbf{g}}_{k+1}, \overline{\mathbf{y}}_k \rangle}{\langle \mathcal{T}_{\mathbf{x}_{k+1}}(\overline{\mathbf{g}}_k), \mathcal{T}_{\mathbf{x}_{k+1}}(\overline{\mathbf{g}}_k) \rangle}, \tag{2.31}$$

$$\widetilde{\beta}_{k+1}^{\text{HS}} = \frac{\langle \overline{\mathbf{g}}_{k+1}, \overline{\mathbf{y}}_k \rangle}{\langle \overline{\mathbf{y}}_k, \mathcal{T}_{\mathbf{x}_{k+1}}(\mathbf{p}_k) \rangle}, \tag{2.32}$$

$$\widetilde{\beta}_{k+1}^{\text{HZ}} = \frac{\langle \overline{\mathbf{y}}_k, \overline{\mathbf{g}}_{k+1} \rangle}{\langle \overline{\mathbf{y}}_k, \mathcal{T}_{\mathbf{x}_{k+1}}(\mathbf{p}_k) \rangle} - 2 \left\| \overline{\mathbf{y}}_k \right\|^2 \frac{\langle \overline{\mathbf{g}}_{k+1}, \mathcal{T}_{\mathbf{x}_{k+1}}(\mathbf{p}_k) \rangle}{\langle \overline{\mathbf{y}}_k, \mathcal{T}_{\mathbf{x}_{k+1}}(\mathbf{p}_k) \rangle^2}, \tag{2.33}$$

where $\overline{\mathbf{y}}_k = \overline{\mathbf{g}}_{k+1} - \mathcal{T}_{\mathbf{x}_{k+1}}(\overline{\mathbf{g}}_k)$. Similarly,

$$\widehat{\beta}_{k+1}^{\text{PR}} = \frac{\langle \mathbf{g}_{k+1}, \overline{\mathbf{y}}_k \rangle}{\langle \mathcal{T}_{\mathbf{x}_{k+1}}(\mathbf{g}_k), \mathcal{T}_{\mathbf{x}_{k+1}}(\overline{\mathbf{g}}_k) \rangle}, \tag{2.34}$$

$$\widehat{\beta}_{k+1}^{\text{HS}} = \frac{\langle \mathbf{g}_{k+1}, \overline{\mathbf{y}}_k \rangle}{\langle \mathbf{y}_k, \mathcal{T}_{\mathbf{x}_{k+1}}(\mathbf{p}_k) \rangle}, \tag{2.35}$$

$$\widehat{\beta}_{k+1}^{\text{HZ}} = \frac{\langle \mathbf{g}_{k+1}, \overline{\mathbf{y}}_k \rangle}{\langle \mathbf{y}_k, \mathcal{T}_{\mathbf{x}_{k+1}}(\mathbf{p}_k) \rangle} - 2 \frac{\langle \mathcal{T}_{\mathbf{x}_{k+1}}(\mathbf{p}_k), \mathbf{g}_{k+1} \rangle \langle \mathbf{y}_k, \overline{\mathbf{y}}_k \rangle}{\langle \mathbf{y}_k, \mathcal{T}_{\mathbf{x}_{k+1}}(\mathbf{p}_k) \rangle^2}, \tag{2.36}$$

where $\mathbf{y}_k = \mathbf{g}_{k+1} - \mathcal{T}_{\mathbf{x}_{k+1}}(\mathbf{g}_k)$. The manifold NPNCG algorithm is given in Algorithm 12.

Algorithm 12 Manifold NPNCG

procedure MANNPNCG($\mathbf{g}(\cdot), \mathbf{x}_k, \mathbf{p}_k, \mathbf{g}_k$)
  $\mathbf{x}_{k+1} = R_{\mathbf{x}_k}(\alpha_k \mathbf{p}_k)$ ▷ α_k determined by line-search
  $\overline{\mathbf{g}}_{k+1} = -\text{Log}_{\mathbf{x}_{k+1}}(\mathcal{Q}(\mathbf{g}; \mathbf{x}_{k+1}))$
  Compute $\overline{\beta}_{k+1}$ by one of (2.31–2.36)
  $\mathbf{p}_{k+1} = -\overline{\mathbf{g}}_{k+1} + \overline{\beta}_{k+1} \mathcal{T}_{\mathbf{x}_{k+1}}(\mathbf{p}_k)$
  return $\mathbf{x}_{k+1}$, $\mathbf{p}_{k+1}$, $\mathbf{g}_{k+1}$
end procedure

2.3.3 Manifold NPQN

As in the previous two cases, the line-search is carried out along the curve defined by the retraction $R_{\mathbf{x}_k}(\cdot)$ (see (1.25)). Furthermore, the vectors $\mathbf{s}_k$, $\mathbf{y}_k$ and $\overline{\mathbf{y}}_k$ require vector transport (the parallel transport variant (1.27) is used here). One iteration of manifold NPQN is described in Algorithm 13.

Algorithm 13 NPQN on Manifolds

1: **procedure** MANNPQN($f, \mathbf{g}, \mathbf{x}_k, \mathbf{S}_k, \mathbf{Y}_k, \overline{\mathbf{Y}}_k$)
2: $\quad \mathbf{g}_k \leftarrow \mathbf{g}(\mathbf{x}_k)$
3: $\quad \overline{\mathbf{g}}_k \leftarrow -\mathrm{Log}_{\mathbf{x}_k}(\mathbf{Q}(\mathbf{x}_k))$
4: $\quad$ compute $\overline{\mathbf{p}}_k$ by (2.23) or (2.26)
5: $\quad \mathbf{x}_{k+1} \leftarrow R_{\mathbf{x}_k}(\alpha_k \overline{\mathbf{p}}_k)$ ▷ α_k determined by line-search
6: $\quad \mathbf{g}_{k+1} = \mathbf{g}(\mathbf{x}_{k+1})$
7: $\quad \overline{\mathbf{g}}_{k+1} \leftarrow -\mathrm{Log}_{\mathbf{x}_{k+1}}(\mathbf{Q}(\mathbf{x}_{k+1}))$
8: $\quad \mathbf{s}_k = \mathcal{T}_{\mathbf{x}_k, \alpha_k \overline{\mathbf{p}}_k}(\alpha_k \overline{\mathbf{p}}_k)$
9: $\quad \mathbf{y}_k = \mathbf{g}_{k+1} - \mathcal{T}_{\mathbf{x}_k, \alpha_k \overline{\mathbf{p}}_k}(\mathbf{g}_k)$
10: $\quad \overline{\mathbf{y}}_k = \overline{\mathbf{g}}_{k+1} - \mathcal{T}_{\mathbf{x}_k, \alpha_k \overline{\mathbf{p}}_k}(\overline{\mathbf{g}}_k)$
11: $\quad$ update $\mathbf{S}_k$ to $\mathbf{S}_{k+1}$, $\mathbf{Y}_k$ to $\mathbf{Y}_{k+1}$, and $\overline{\mathbf{Y}}_k$ to $\overline{\mathbf{Y}}_{k+1}$
12: $\quad$ **return** $\mathbf{x}_{k+1}, \mathbf{S}_{k+1}, \mathbf{Y}_{k+1}, \overline{\mathbf{Y}}_{k+1}$
13: **end procedure**

In most of this algorithm we work with tangent vectors as N-tuples of matrices. Two exceptions are in line 4, where we compute the search direction, and line 11, where we update the storage matrices. To use (2.23) or (2.26) the tangent vectors $\mathbf{g}_k$ and $\overline{\mathbf{g}}_k$ are converted into 1D arrays by vectorizing each factor matrix column-wise and vertically concatenating the results. Once the search direction is computed this process is reversed to produce the tangent vector $\overline{\mathbf{p}}_k$ for use in the retraction. Similarly, before the updates in line 11 the tangent vectors $\mathbf{s}_k$, $\mathbf{y}_k$, and $\overline{\mathbf{y}}_k$ must also be vectorized. When updating $\mathbf{S}_k$, $\mathbf{Y}_k$ and $\overline{\mathbf{Y}}_k$ in the manifold context, the proper approach is to parallel transport these matrices to $\mathbf{x}_{k+1}$ before appending the new column, as described in [89, § 7.1]. However, transport of these matrices can be computationally expensive, and the savings of omitting this step may significantly outweigh the cost of an increase in iteration count resulting from this change.

Chapter 3

Numerical Results

In this chapter we present numerical results that illustrate the efficacy of the nonlinearly preconditioned optimization methods developed in Chapter 2. § 3.1 uses the Tucker HOSVD ATD problem to compare (i) manifold NPNGMRES with exact and linearized Hess f operators, and (ii) manifold NPNCG with different β parameters. Tests for Euclidean NPNCG and NPNGMRES using the CP ATD problem have previously been performed by other authors in [32,34], hence these tests are not repeated here. Next, in § 3.2 we compare the best variants of NPNGMRES and NPNCG to the L-BFGS and L-Broyden NPQN methods, using the CP ATD problem for optimization over standard Euclidean space and the Tucker HOSVD ATD problem for optimization over matrix manifolds. In both contexts we consider multiple choices of line-search, window size, and nonlinear preconditioner for the NPQN methods.

3.1 Comparison of Manifold NPNCG and NPGMRES Methods

All tests within this section were implemented using Matlab R2010a on an Intel Core i7-2630QM computer with 8GB of RAM, using the Tensor Toolbox (V2.5) [7, 8] and the ManOpt Toolbox (V1.0.7) [14] for tensor and manifold computations, respectively. Manifold NPNCG and NPNGMRE, were used to compute Tucker HOSVD ATDs for order-3 tensors by minimizing

$$\frac{1}{2}\left\|\mathcal{X}\cdot(\mathbf{A}^{(1)},\ldots,\mathbf{A}^{(N)})\right\|_F^2 \tag{3.1}$$

on a Cartesian product of three Grassmannians. These methods were nonlinearly preconditioned by using a forward HOOI sweep (Algorithm 3) as $\mathcal{Q}(\mathbf{g};\mathbf{x})$. As matrices returned by $\mathcal{Q}$ have orthonormal columns, they correctly define a point on the manifold.

The gradient of (3.1) can be written as a matrix triplet. All computations in NPNCG are well defined using matrix triplet vectors. NPNGMRES requires vectorization when forming the least squares system: to vectorize a matrix triplet each component matrix is vectorized and then concatenated vertically. Truncated HOSVD (Algorithm 2) was used to generate initial matrix triplets. NPNGMRES used a maximum window of $m = 25$ past iterates. These algorithms used a version of the Moré-Thuente line-search from the Poblano Toolbox (v1.0) [37, 75] modified to carry out the search along a retraction curve. We provide functions for the cost and Riemannian gradient, as well as for the retraction, inner product, and norm. The MT line-search is designed to compute a step that satisfies

$$\text{Sufficient Decrease:} \quad f(\mathbf{x}_k + \texttt{stp} \cdot \mathbf{p}_k) \leq f(\mathbf{x}_k) + \texttt{ftol} \cdot \texttt{stp} \cdot (\mathbf{g}_k^\intercal \mathbf{p}_k), \tag{3.2}$$

$$\text{Curvature:} \quad |\mathbf{g}^\intercal(\mathbf{x}_k + \texttt{stp} \cdot \mathbf{p}_k)\mathbf{p}_k)| <= \texttt{gtol} \cdot |\mathbf{g}_k^\intercal \mathbf{p}_k|. \tag{3.3}$$

We use the same line-search parameters as in [32, 34], which we record in Table 3.1.

Parameter	Description	Value
`ftol`	Tolerance for (3.2)	10^{-4}
`gtol`	Tolerance for (3.3)	10^{-2}
`max_it`	Maximum iterations	20
`init_stp`	Initial step length	1.0

Table 3.1: Moré-Thuente Line-Search Parameters

Successful termination occurred when $\|\mathbf{g}_k\|_F < \text{tol} \cdot |f(\mathbf{x}_k)|$, where $\text{tol} = 10^{-7}$. Note that NCG and NPNCG convergence stalls when $\|\mathbf{g}\|_F / |f(\mathbf{x})| \approx 10^{-7}$, which is a well known phenomenon that can be explained by a loss of accuracy in the linesearch step, where the Wolfe sufficient decrease condition (3.2) is checked [53]. When recording computation time, we omitted time spent checking the termination condition for all methods because less expensive stopping criteria may be used in practice. If NPNGMRES produced an ascent search direction, we restarted by discarding all past iterates and using the negative of the ascent direction. NPNCG methods are restarted by setting $\beta = 0$ every 50 iterations.

We considered three test problems. Each problem compared manifold NPNGMRES and NPNCG to four existing methods: HOOI [30], manifold NCG [60], manifold L-BFGS (non-preconditioned) [89], and the manifold trust region (TR) solver from the Manopt toolbox (V1.0.7) [3, 14, 61]. For L-BFGS we used the same termination conditions and

set $m = 5$. The TR solver used the default finite difference Hessian approximation [13], due to the computational cost of evaluating the exact Hessian, and the default parameters prescribed by the code's authors, which are summarized in Table 3.2.

Parameter	Description	Value
$\overline{\Delta}$	Maximum trust-region radius	$(\sum_n \text{rank}_n(\mathcal{X}))^{1/2}$
Δ_0	Initial trust-region radius	$\overline{\Delta}/8$
κ	tCG linear convergence target rate	0.1
θ	tCG superlinear convergence target rate	1.0
ρ'	Accept/reject threshold	0.1

Table 3.2: Trust Region Parameters

Problem I.

For the first problem, we generated synthetic tensor data with specified size, multilinear rank, and level of noise. Similar tests have been considered in [23, 40, 61, 102]. Given a Tucker tensor $\mathcal{X} \in \mathbb{R}^{I\times I\times I}$ with multilinear rank (R, R, R), such that $\mathcal{S}$ has standard normal distributed entries and each $\mathbf{A}^{(n)}$ has orthonormal columns, test tensors are obtained by adding noise to $\mathcal{X}$. As in [34], given $\mathcal{N}_1$ and $\mathcal{N}_2$ with entries from the standard normal distribution, $\mathcal{N}(0, 1)$, homoskedastic and heteroskedastic noise were added according to

$$\mathcal{X}' = \mathcal{X} + \sqrt{\frac{\ell_1}{100-\ell_1}}\frac{\|\mathcal{X}\|_F}{\|\mathcal{N}_1\|_F}\mathcal{N}_1 \text{ and } \mathcal{X}'' = \mathcal{X}' + \sqrt{\frac{\ell_2}{100-\ell_2}}\frac{\|\mathcal{X}'\|_F}{\|\mathcal{N}_2 * \mathcal{X}'\|_F}\mathcal{N}_2 * \mathcal{X}' \tag{3.4}$$

respectively, with final test tensor $\mathcal{X}''$. Parameters ℓ_1 and ℓ_2 control noise levels: $\ell_i = 0$ corresponding to no noise and $\ell_i = 50$ corresponding to noise of the same magnitude as $\mathcal{X}$.

We considered the two parameter combinations $(I, R, \ell_1, \ell_2) = (60, 20, 10, 10)$ and $(120, 40, 10, 10)$, running 10 trials for each combination. Each trial involved computing rank $\frac{1}{2}(R, R, R)$ ATDs for an $\mathcal{X}''$ obtained from $\mathcal{X}$ and new noise tensors $\mathcal{N}_1$ and $\mathcal{N}_2$. Results from HOOI, L-BFGS, TR, and NCG are compared to NPNGMRES using (2.28) or (2.29) and NPNCG using one of the six β formulas (2.31–2.36). Upper limits of 2000 iterations and 1500 seconds computation time ensured all algorithms eventually terminated.

The minimum, median, maximum, and mean times-to-solution and iteration counts are recorded in Tables 3.3 and 3.4. For the smaller test tensor (Table 3.3) we see that, with the exception of L-BFGS and NPNGMRES using (2.28), all methods outperformed HOOI in terms of time required. L-BFGS mean and median times are approximately the same

		Time				Iterations			
		Min	Med	Max	Mean	Min	Med	Max	Mean
HOOI		3.13	5.19	9.52	5.62	106	172	310	186
L-BFGS		3.84	5.32	9.12	5.40	79	98	181	105
TR		1.73	2.39	4.86	2.74	14	19	27	20
NCG	β^{PR}	1.74	2.25	3.74	2.39	74	86	181	101
	β^{HS}	1.57	**2.09**	3.75	**2.33**	74	86	181	101
	β^{HZ}	1.54	2.19	4.03	2.36	74	86	181	101
NPNCG	$\widehat{\beta}^{PR}$	1.91	2.90	3.23	2.80	35	46	56	47
	$\widehat{\beta}^{HS}$	1.93	2.91	3.54	2.88	35	46	56	47
	$\widehat{\beta}^{HZ}$	1.92	2.85	3.55	2.85	35	47	56	48
	$\widetilde{\beta}^{PR}$	1.75	2.62	3.17	2.61	31	43	53	44
	$\widetilde{\beta}^{HS}$	1.79	2.76	3.42	2.70	32	44	55	45
	$\widetilde{\beta}^{HZ}$	2.54	3.67	19.99	5.79	49	65	395	107
NPNGMRES	(2.28)	10.42	16.56	29.89	16.93	25	33	45	33
	(2.29)	1.68	**2.51**	3.70	**2.52**	21	30	45	30

Table 3.3: Problem I. Results for test case 1: $(I, R, \ell_1, \ell_2) = (60, 20, 10, 10)$. (Bold entries indicate the lowest time to solution for the existing and newly proposed methods.)

as HOOI, whereas NPNGMRES times were approximately three times larger than those for HOOI. The slowness of NPNGMRES using (2.28) is unsurprising, as the evaluation of Hessian-vector products is computationally expensive. Among the methods faster than HOOI, TR was second only to NCG in terms of time required. The three NCG variations gave near identical results in terms of time and iteration count. These were followed by NPNGMRES using (2.29) and NPNCG using $\widetilde{\beta}^{PR}$ or $\widetilde{\beta}^{HS}$, all of which significantly reduced the HOOI iterations required for convergence (viewing these methods as accelerators for HOOI). An interesting observation is that NPNCG using $\widetilde{\beta}^{HZ}$ exhibited significantly worse performance than the other two $\widetilde{\beta}$ methods, whereas the three $\widehat{\beta}$ variants exhibited very similar results.

For the larger test tensor (Table 3.4), once again NPNGMRES using (2.28) and L-BFGS are the slowest non-HOOI methods. While L-BFGS is faster than HOOI in this case, it still lags behind the other methods significantly. TR significantly outperformed the other methods in terms of time-to-solution. It is followed by NPNCG, where the β formulas produce fairly similar results, with the exception of $\widetilde{\beta}^{HZ}$, which exhibited significantly slower convergence. NPNGMRES using (2.29) was slower than NPNCG, except when $\widetilde{\beta}^{HZ}$

		Time				Iterations			
		Min	Med	Max	Mean	Min	Med	Max	Mean
HOOI		20.99	36.91	95.68	40.54	287	506	1322	558
L-BFGS		25.77	32.84	41.03	32.71	139	180	222	180
TR		6.85	**11.99**	18.11	**11.99**	21	33	50	33
NCG	β^{PR}	11.52	18.20	23.46	17.82	130	210	253	198
	β^{HS}	11.61	18.43	24.66	18.01	130	209	266	199
	β^{HZ}	11.52	18.39	25.70	18.06	130	209	272	200
NPNCG	$\widehat{\beta}^{PR}$	11.14	15.81	19.49	**15.43**	58	90	114	87
	$\widehat{\beta}^{HS}$	10.88	16.28	19.94	15.70	57	90	114	87
	$\widehat{\beta}^{HZ}$	10.91	16.15	19.29	15.73	56	91	113	88
	$\widetilde{\beta}^{PR}$	11.97	**15.23**	21.02	15.79	64	86	125	89
	$\widetilde{\beta}^{HS}$	12.60	15.46	22.16	16.13	66	87	130	90
	$\widetilde{\beta}^{HZ}$	17.53	23.48	37.33	25.17	98	139	243	152
NPNGMRES	(2.28)	295.01	391.00	573.67	406.35	64	79	122	82
	(2.29)	13.49	16.98	25.23	18.38	52	68	102	71

Table 3.4: Problem I. Results for test case 2: $(I, R, \ell_1, \ell_2) = (120, 40, 10, 10)$. (Bold entries indicate the lowest time to solution for the existing and newly proposed methods.)

was used. For this problem NCG, while still significantly faster than HOOI, lagged behind the preconditioned methods.

Overall, the results for this synthetic test problem are promising: the best of the new manifold NPNCG and NPNGMRES methods outperform the existing HOOI and L-BFGS methods, and are competitive with the existing TR and NCG methods, which can be somewhat faster. In the following tests we will see that for more difficult test tensors (noisy and of larger size) accurate results can be obtained by NPNCG and NPNGMRES methods much faster and more robustly than for all other existing methods considered.

Problem II.

The second problem used the AT&T Database of Faces [87], a collection of 400 images (40 subjects with 10 images each) that has been previously used as test data in [23, 103, 110]. Images for a given subject feature varying lighting, facial expressions and facial details. 10 images from a single subject, each 92x112 pixels with 256 grey levels, were used to create a $92 \times 112 \times 10$ tensor $\mathcal{X}$ with entries scaled to lie within $[0, 1]$. Noise was then

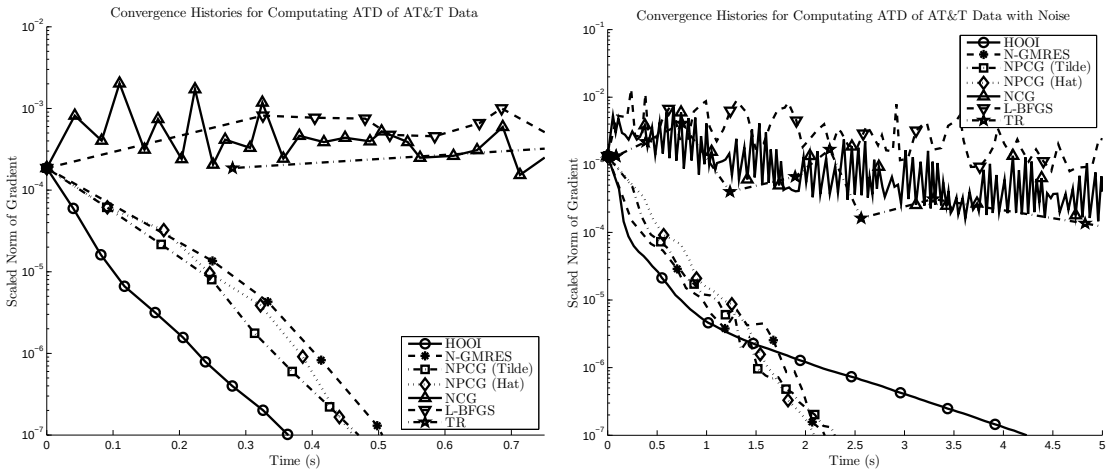

Figure 3.1: Problem II. Sample convergence histories based on AT&T face input data: noise-free (left) and noisy (right).

added according to $\mathcal{X}' = \mathcal{X} + 0.2\frac{\|\mathcal{X}\|}{\|\mathcal{N}\|}\mathcal{N}$, where the entries of $\mathcal{N}$ were drawn from $\mathcal{N}(0,1)$. We computed rank-$(30,35,8)$ HOSVD ATDs of both $\mathcal{X}$ and $\mathcal{X}'$ using the six algorithms from Problem I, restricting NCG and NPNCG to β^{HS} formulas and NPNGMRES to (2.29). Upper limits of 250 iterations and 1500 seconds were enforced. Sample convergence histories in terms of time required are presented in Figure 3.1.

A striking result is that NCG, L-BFGS, and TR all failed to converge within a reasonable amount of time, compared to the other methods. One potential explanation is that initial points from truncated HOSVD may have been too far from the minima for fast convergence. In [89] the authors used 5 to 50 iterations of HOOI to refine initial points before using their BFGS QN methods. To test this hypothesis the experiment was repeated ten times for different $\mathcal{X}'$, using 20 iterations of HOOI to provide a refined initial point for NCG, L-BFGS, and TR. The time (including the 20 refining HOOI), iteration count (excluding the 20 refining HOOI), and scaled gradient norm for each trial are recorded in Table 3.5. These results show that, in spite of the improved initial point, NCG and L-BFGS both failed to converge within 250 iterations for all recorded attempts. Indeed, the oscillations present in the NCG and L-BFGS plots of Figure 3.1 still occur, only at lower scaled gradient norm values. While TR does converge, it much longer than HOOI, let alone the newly proposed methods. However, as the Manopt trust region method is generic in scope, we may be able to obtain significant improvements in efficiency by adapting it and its parameters specifically to the HOSVD ATD problem. However, we expect that an efficiency-minded implementation is not likely to eliminate the gap observed between TR

		Trial									
		1	2	3	4	5	6	7	8	9	10
HOOI	Time	5.85	6.17	3.12	10.70	7.42	13.49	9.31	10.50	6.91	9.88
	Iter	121	131	55	215	149	250*	193	225	145	210
	S. Grad	9.7e-8	9.9e-8	9.6e-8	9.9e-8	9.7e-8	5.0e-7	9.7e-8	9.9e-8	9.9e-8	9.9e-8
NPNGMRES	Time	3.02	3.53	**2.19**	3.94	4.02	7.49	3.87	4.65	2.96	4.83
	Iter	24	27	15	32	29	53	32	35	25	36
	S. Grad	6.9e-8	5.6e-8	9.4e-8	8.5e-8	5.1e-8	7.4e-8	5.4e-8	7.7e-8	7.9e-8	8.8e-8
$\widetilde{\beta}^{HS}$ NPNCG	Time	2.97	3.00	2.30	3.52	3.79	6.10	3.38	4.22	2.99	4.25
	Iter	34	35	21	39	40	66	38	48	34	47
	S. Grad	8.8e-8	7.5e-8	2.6e-8	9.7e-8	8.4e-8	3.6e-8	7.8e-8	7.9e-8	9.2e-8	9.0e-8
$\widehat{\beta}^{HS}$ NPNCG	Time	**2.82**	**2.99**	2.22	**2.85**	**3.46**	**4.90**	**2.87**	**3.74**	**2.64**	**4.02**
	Iter	32	32	20	32	34	48	32	42	30	45
	S. Grad	8.5e-8	9.0e-8	8.3e-8	9.7e-8	7.9e-8	6.2e-8	6.3e-8	1.0e-7	6.9e-8	8.9e-8
β^{HS} NCG	Time	8.33	8.91	9.11	9.15	8.95	9.57	8.25	8.30	8.38	8.31
	Iter	250*	250*	250*	250*	250*	250*	250*	250*	250*	250*
	S. Grad	1.7e-5	5.6e-5	2.2e-6	1.4e-5	1.6e-5	3.3e-5	2.0e-5	9.4e-6	7.2e-6	2.2e-5
L-BFGS	Time	17.51	21.10	19.09	20.26	18.75	18.11	17.96	17.43	17.75	18.87
	Iter	250*	250*	250*	250*	250*	250*	250*	250*	250*	250*
	S. Grad	4.5e-5	7.2e-5	1.5e-5	2.5e-5	4.1e-5	1.4e-4	5.0e-5	4.4e-5	6.4e-5	5.0e-5
TR	Time	28.94	33.13	21.76	47.53	40.64	77.78	40.78	48.05	28.53	50.94
	Iter	11	8	3	8	15	21	9	18	9	18
	S. Grad	4.2e-9	8.5e-9	4.3e-10	2.3e-9	6.6e-9	2.1e-9	3.3e-10	2.7e-8	6.0e-8	2.3e-9

Table 3.5: Problem II. Results for noisy AT&T face data. (Asterisks denote runs which did not converge within iteration or time limits. Initial 20 HOOI iterations for NCG, L-BFGS and TR not included in iteration count, but included in total time.)

and the newly proposed methods.

In the noise-free case (not included in the table), the NPNCG and NPNGMRES iterations converge in roughly the same amount of time, with HOOI being slightly faster. Once noise was introduced in $\mathcal{X}'$, the situation was reversed, with NPNGMRES and NPNCG converging significantly faster than HOOI and the other methods. In this particular case, NPNCG using $\widehat{\beta}^{HS}$ was the fastest in nine of ten cases, followed by $\widetilde{\beta}^{HS}$ and NPNGMRES, which generally required similar amounts of time. To quantify how well the ATD $\widehat{\mathcal{X}}$ represents $\mathcal{X}$, we use the ratio

$$\frac{\|\mathcal{X}\|_F - \|\mathcal{X} - \widehat{\mathcal{X}}\|_F}{\|\mathcal{X}\|_F},$$

where values near 1 indicate close agreement. For the noise free input tensor, $\mathcal{X}$, the ATD of $\mathcal{X}$ had a fit of 0.9492. The fit relative to $\mathcal{X}$ dropped to 0.9312 for the ATD of $\mathcal{X}'$, indicating that while a portion of the noise can be removed, a loss of accuracy still occurs.

Figure 3.2: Problem II. From top to bottom: input tensor image, image from the ATD, input noisy tensor image, and image from the ATD of noisy tensor.

This is illustrated in Figure 3.2, where we show, from left to right, the 10 different facial expressions corresponding to, from top to bottom, $\mathcal{X}$, the ATD of $\mathcal{X}$, $\mathcal{X}'$, and the ATD of $\mathcal{X}'$. Approximations from $\mathcal{X}$ are somewhat blurred, reflecting a loss of fine detail, but facial features are quite well represented.

Problem III.

The final test problem used the MNIST Database of Handwritten Digits [67], previously used in [88, 103]. This is a collection of 70,000 images, each of a digit centered in a 28×28 image. We formed a tensor $\mathcal{X} \in \mathbb{R}^{28\times 28\times 5000}$ consisting of 5000 images of the digit 5. ATDs with multilinear rank $(14, 14, 100)$ were computed by the same methods considered in the previous example. The experiment was then repeated using $\mathcal{X}' = \mathcal{X} + 2.5\frac{\|\mathcal{X}\|}{\|\mathcal{N}\|}\mathcal{N}$, where $\mathcal{N}$ has entries uniformly distributed in $[0, 1]$. Upper limits of 250 iterations and 1500 seconds were imposed. Sample convergence histories are presented in Figure 3.3. As in Problem II, we repeated the experiment on ten different noisy tensors, now using 50 iterations of HOOI to refine initial points for NCG, L-BFGS, and TR. Table 3.6 reports the total times (including the 50 refining HOOI), numbers of iterations (excluding the 50 refining HOOI), and scaled gradient norms for each trial.

As in the previous problem, we observed that NCG and L-BFGS failed to converge for

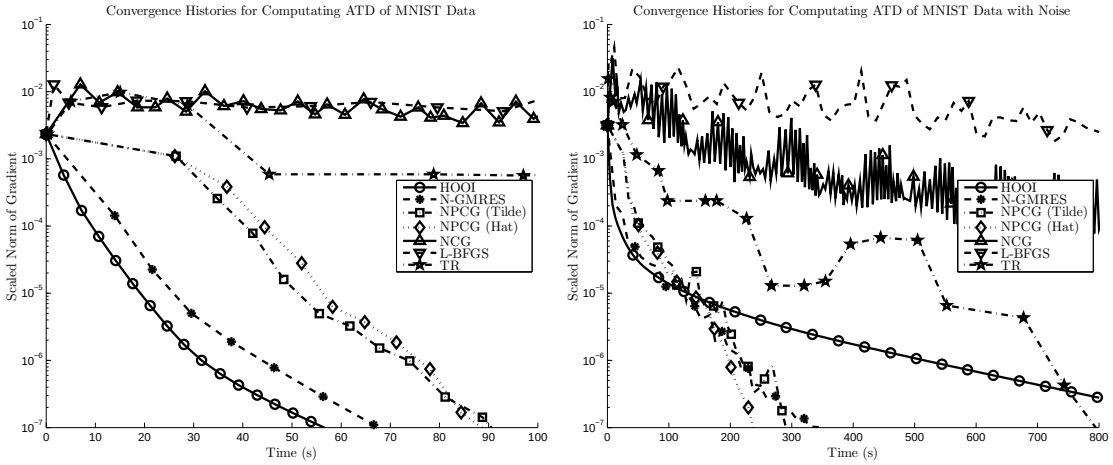

Figure 3.3: Problem III. Sample convergence histories based on MNIST digit input data: noise-free (left) and noisy (right). (Without any initial HOOI iterations.)

all trials. In the noise-free case (not included in table), HOOI converges the fastest, followed by NPNGMRES and then NPNCG. TR did converge, but not as quickly as the other convergent methods, hence it is not observable in the plot. Once noise is introduced, we observe that NPNGMRES and the two NPNCG methods all outperform HOOI, typically satisfying the tolerance in one third of the time. For this problem NPNCG using $\widehat{\beta}^{HS}$ was the fastest method across the board, followed by NPNGMRES and NPNCG using $\widetilde{\beta}^{HS}$. The success of NPNCG using $\widehat{\beta}^{HS}$ may be due to nonlinear transformation preconditioning incorporating both gradient and preconditioner direction information into $\widehat{\beta}$. Finally, while TR required the fewest iterations, it required significantly more time than the newly proposed methods, failing to outperform HOOI on three occasions.

The ATD of $\mathcal{X}$ had a fit of 0.7884, and the ATDs of each $\mathcal{X}'$ had an average fit of 0.6200 relative to $\mathcal{X}$. The first ten approximation images are shown in Figure 3.4. The main discrepancies in the ATD of $\mathcal{X}'$ are image artifacts in the form of dark blotches surrounding the digits. In spite of the extent to which the noise visually obfuscated the data, the ATD was able to successfully identify and recognizably display the correct digits. These results suggest there are significant benefits to nonlinearly preconditioning NGMRES and NCG using HOOI, and that the advantages of these nonlinearly preconditioned methods over NCG, L-BFGS, and TR are significant in terms of convergence speed and robustness, as becomes apparent when moving to larger, more noisy, and more realistic data sets.

		Trial									
		1	2	3	4	5	6	7	8	9	10
HOOI	Time	797	423	755	544	713	460	273	538	739	634
	Iter	193	103	182	132	172	112	65	128	174	146
	S. Grad	9.7e-8	1.0e-7	9.8e-8	9.8e-8	9.6e-8	1.0e-7	9.6e-8	9.7e-8	9.8e-8	9.8e-8
NPNGMRES	Time	288	221	271	242	297	304	170	218	288	235
	Iter	24	19	24	20	25	23	15	19	25	21
	S. Grad	8.1e-8	5.7e-8	5.9e-8	6.7e-8	6.2e-8	7.6e-8	9.8e-8	8.5e-8	7.5e-8	7.3e-8
$\widetilde{\beta}^{HS}$ NPNCG	Time	267	218	251	390	290	1501*	168	263	436	299
	Iter	31	27	32	48	34	116	20	32	55	38
	S. Grad	9.1e-8	6.9e-8	9.7e-8	9.8e-8	7.6e-8	3.0e-7	3.6e-8	2.6e-8	9.6e-8	7.4e-8
$\widehat{\beta}^{HS}$ NPNCG	Time	**258**	**177**	**228**	**206**	**236**	**246**	**164**	**204**	**232**	**217**
	Iter	30	22	28	24	28	27	19	23	28	26
	S. Grad	7.9e-8	7.8e-8	6.7e-8	7.7e-8	8.4e-8	9.8e-8	9.0e-8	9.1e-8	9.4e-8	7.5e-8
β^{HS} NCG	Time	1045	1120	1066	1075	1052	1221	1719	1108	1095	1072
	Iter	250*	250*	250*	250*	250*	250*	250*	250*	250*	250*
	S. Grad	8.2e-6	3.7e-7	1.8e-6	1.4e-6	7.2e-6	3.8e-7	1.2e-7	1.7e-6	4.8e-6	1.1e-6
L-BFGS	Time	1712*	1712*	1718*	1715*	1717*	1731*	1719*	1716*	1715*	1712*
	Iter	121	123	122	123	122	119	118	118	119	120
	S. Grad	4.1e-5	2.2e-6	8.3e-6	5.7e-6	3.3e-5	3.9e-6	4.3e-7	5.4e-6	1.9e-5	5.2e-6
TR	Time	621	445	495	418	637	464	298	408	629	395
	Iter	6	2	3	2	6	2	1	2	3	2
	S. Grad	2.3e-8	1.2e-8	1.4e-8	7.7e-8	3.1e-8	1.6e-8	3.4e-8	6.1e-8	3.4e-8	1.0e-7

Table 3.6: Problem III. Results for noisy MNIST digit data. (Asterisks denote runs which did not converge within iteration or time limits. Initial 50 HOOI iterations for NCG, L-BFGS and TR not included in iteration count, but included in total time.)

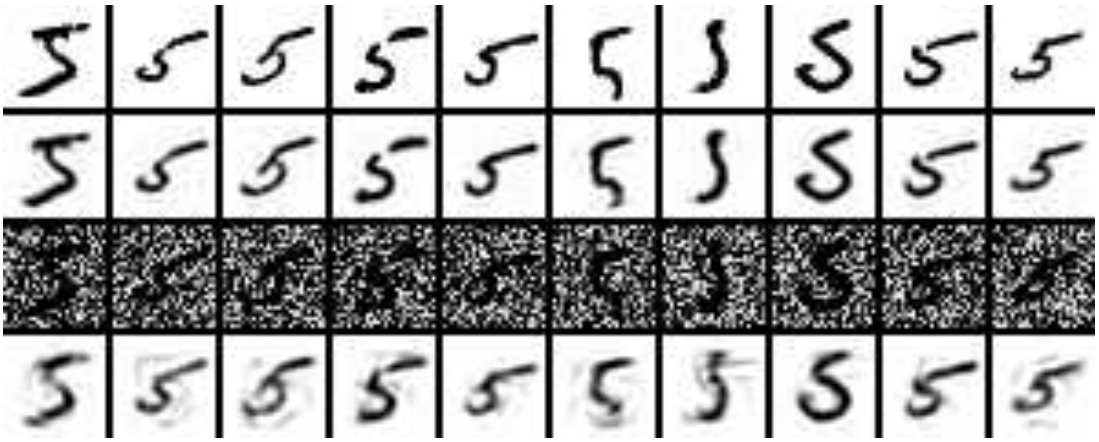

Figure 3.4: Problem III. From top to bottom: input tensor image, image from the ATD, input noisy tensor image, and image from the ATD of noisy tensor.

3.2 Comparison to NPQN Methods

We now compare the L-BFGS and L-Broyden NPQN methods to NPNCG using $\widehat{\beta}^{HS}$ or $\widetilde{\beta}^{HS}$ and NPNGMRES using (2.29), the best choices for these methods as identified in the previous subsection. We again consider problems based on computing ATDs, comparing standard nonlinearly preconditioned methods using the approximate CP decomposition problem and comparing the manifold methods again in terms of the Tucker HOSVD ATD problem. All of the following experiments were implemented on a MacBook Pro (2.5 GHz Intel Core i7-4770HQ, 16 GB 1600 MHz DDR3 RAM) using MATLAB R2017a with the Tensor Toolbox (V2.6) [7, 8] to handle tensor computations and the ManOpt toolbox (V3.0) [14] for manifold operations.

3.2.1 Optimization over Euclidean Space

We consider L-BFGS and L-Broyden, their left preconditioning (LP) variants (2.18) and (2.25), and their variable transformation preconditioning (TP) variants (2.23) and (2.26). As discussed in [32, 34], the CP-ALS fixed point iteration can serve as an effective preconditioner for NPNCG and NPNGMRES, and we shall show this is also the case for NPQN methods. Specifically, for the preconditioner $\mathcal{Q}$ we use either one forward sweep (F) or one forward-backward sweep (FB) of Algorithm 1, lines 2–5 with either $n = 1, \ldots, N$ or $n = 1, 2 \ldots, N-1, N, N-1, \ldots, 2, 1$. For the CP ATD problem we set $\theta_k = 1$ for preconditioned L-Broyden methods and $\theta_k = \widehat{\gamma}_k$ as prescribed in (1.22) for non-preconditioned L-Broyden, as these were observed to give the best results (not included here).

We first test to determine the best window size $m \in \{1, \ldots, 10\}$ and line-search method for subsequent experiments. Two line-searches are considered. The first is the MT algorithm from the Poblano Toolbox (v1.1) [37, 75], using the same default parameters recorded in Table 3.1. The second, which we refer to as modified backtracking (modBT), is an attempt at imposing a "relaxed" line-search condition on the QN step, based on the observation that in certain cases the NPQN methods (and also some of the other methods we compare with) converged faster with a fixed unit step length instead of a line-search. In such cases the sequence of objective values was not monotonic, hence modBT does not require the objective value to decrease at every iteration, but only not to increase too much, with the increase tolerated decreasing as iteration count increases. Step lengths of 1, 1/2, and 1/4 are considered, accepting the step as soon as growth is small enough; and if all three are rejected, it takes a step of length 1/8 in the preconditioner direction. This is a feasible approach if methods work with unit step length close to convergence, such as

QN, but not for those requiring an accurate line-search, such as NCG. This algorithm is summarized in Algorithm 14. The while statement condition assumes our objective values will be non-negative, which is the case for the CP decomposition problem.

Algorithm 14 Modified Backtracking Line-Search

procedure MODBT($\mathbf{x}_k$,f_k,$\overline{\mathbf{g}}_k$,$\mathbf{p}_k$,iter)
  $\alpha_k = 1$, flag $= 0$
  $\mathbf{x}_{k+1} = \mathbf{x}_k + \alpha_k \mathbf{p}_k$
  $f_{k+1} = f(\mathbf{x}_{k+1})$
  while $f_{k+1} > (1 + e^{-2 \cdot \text{iter}}) f_k$ && flag < 4 **do**
    flag $=$ flag $+ 1$
    if flag $== 3$ **then**
      $\mathbf{S}_k = [\,], \mathbf{Y}_k = [\,]$
      $\mathbf{p}_k = -\overline{\mathbf{g}}_k$
    else
      $\alpha_k = 0.5\alpha_k$
    end if
    $\mathbf{x}_{k+1} \leftarrow \mathbf{x}_k + \alpha_k \mathbf{p}_k$
    $f_{k+1} \leftarrow f(\mathbf{x}_{k+1})$
  end while
  return $\mathbf{x}_{k+1}, f_{k+1}, \alpha_k$
end procedure

Both line-searches have a reset condition to recover from bad steps. If the search fails to reduce the objective value sufficiently the QN approximation is reset by clearing $\mathbf{S}_k$ and $\mathbf{Y}_k$, following which we either take a unit length step in the preconditioner direction or, if no preconditioning is used, take a step in the steepest descent direction. For MT we repeat the search in this new direction, whereas for modBT we simply take a step of length $1/8$.

To compare these algorithms we compute CP decompositions of an order-3 tensor, as in [5, 32, 97], a standard test problem. We form a pseudo-random test tensor of size $(I \times I \times I)$ for $I = 100$, with known rank $R = 5$ and specify the colinearity C of the factors in each mode to be 0.9, meaning that

$$\frac{\mathbf{a}_r^{(n)\mathsf{T}} \mathbf{a}_s^{(n)}}{\left\|\mathbf{a}_r^{(n)}\right\| \left\|\mathbf{a}_s^{(n)}\right\|} = C \tag{3.5}$$

for $r \neq s$, $r, s = 1, \ldots, R$, and $n = 1, 2, 3$. Highly colinear columns in the factor matrices indicates an ill-conditioned problem, with slow convergence for ALS and other methods.

		Window Size (m)									
		1	2	3	4	5	6	7	8	9	10
L-BFGS	Time	5.2	4.9	4.6	4.7	3.9	4.3	4.1	4.2	3.9	4.0
	Iter	941	835	824	821	675	745	699	708	658	664
L-BFGS-LP-F	Time	2.2	1.3	**1.3**	1.8	1.7	1.6	2.0	**1.7**	1.9	1.8
	Iter	156	88	87	116	114	100	127	106	119	112
L-BFGS-LP-FB	Time	3.6	1.5	2.7	3.3	2.3	3.5	3.8	7.3	4.8	4.4
	Iter	183	77	124	144	124	151	162	294	195	185
L-BFGS-TP-F	Time	**1.2**	**1.1**	**1.1**	**1.2**	**1.4**	**1.4**	**1.5**	**1.6**	1.9	1.8
	Iter	102	84	83	88	99	98	102	107	124	124
L-BFGS-TP-FB	Time	**1.0**	**1.2**	**1.3**	**1.3**	**1.4**	**1.5**	**1.6**	1.8	1.8	1.8
	Iter	57	71	77	80	83	87	95	101	103	107
L-BROY	Time	5.5*	6.8*	6.2*	6.3	6.1	6.0	5.8	6.0	5.9	6.0
	Iter	1000	1000	1000	989	953	925	906	929	906	933
L-BROY-LP-F	Time	1.8	1.9	1.7	1.4	2.4	2.1	2.5	2.2	**1.7**	2.6
	Iter	142	140	127	102	164	150	168	157	126	183
L-BROY-LP-FB	Time	3.1	2.2	3.1	2.4	3.5	2.0	2.2	2.5	2.7	2.2
	Iter	188	126	183	140	198	116	128	139	153	129
L-BROY-TP-F	Time	1.9	1.5	1.5	1.5	**1.4**	1.6	**1.5**	**1.6**	**1.7**	**1.7**
	Iter	164	115	120	120	105	123	118	123	134	131
L-BROY-TP-FB	Time	2.7	1.6	1.6	1.4	**1.6**	1.6	1.7	**1.6**	**1.7**	**1.6**
	Iter	185	101	101	87	103	102	107	100	104	104
Independent of Window Size											
ALS	Time	58.5*	58.5*	58.5*	58.5*	58.5*	58.5*	58.5*	58.5*	58.5*	58.5*
	Iter	10000	10000	10000	10000	10000	10000	10000	10000	10000	10000
NPNGMRES	Time	1.6	1.6	1.6	1.6	**1.6**	1.6	**1.6**	**1.6**	**1.6**	**1.6**
	Iter	100	100	100	100	100	100	100	100	100	100
NPNCG $\widehat{\beta}_{\text{HS}}$	Time	3.2	3.2	3.2	3.2	3.2	3.2	3.2	3.2	3.2	3.2
	Iter	144	144	144	144	144	144	144	144	144	144
NPNCG $\widetilde{\beta}_{\text{HS}}$	Time	2.1	2.1	2.1	2.1	2.1	2.1	2.1	2.1	2.1	2.1
	Iter	107	107	107	107	107	107	107	107	107	107

Table 3.7: Results for MT line-search for CP decomposition. (Asterisks denote runs which failed to converge. Bold entries indicate the best two times for a given value of m.)

The methodology for creating such a tensor is described in [97]. To this tensor of known rank we then add homoskedastic and heteroskedastic noise as described in (3.4). For each combination of window size m, solver, and line-search we ran ten trials, each corresponding to a different random initial guess for the same random tensor, recording the mean time-to-solution and number of iterations required. The same set of ten initial guesses were used for all test combinations. The iterations ran until a maximum of 1,000 iterations was reached, 10,000 function evaluations had been computed, or $\|\mathbf{g}_k\| \,/\, \text{numel}(\mathbf{x})$ decreased below a tolerance of 10^{-7}, where $\text{numel}(\mathbf{x}) = 3IR$ is the number of unknowns in

		Window Size (m)									
		1	2	3	4	5	6	7	8	9	10
L-BFGS	Time	3.1*	3.1*	3.0*	3.0*	3.0*	3.0*	3.1*	3.1*	3.0*	3.2*
	Iter	1000	1000	1000	1000	1000	1000	1000	1000	1000	1000
L-BFGS-LP-F	Time	**0.7**	0.9	**0.8**	**0.8**	**0.8**	**0.8**	**0.8**	**0.8**	**0.8**	**0.8**
	Iter	67	85	70	72	75	71	77	72	75	72
L-BFGS-LP-FB	Time	0.9	**0.8**	1.3	**0.8**	**0.8**	**0.8**	1.2	**0.9**	**0.8**	**0.8**
	Iter	65	62	84	62	58	62	82	64	59	61
L-BFGS-TP-F	Time	**0.8**	**0.8**	**0.8**	**0.8**	**0.8**	**0.8**	**0.8**	**0.8**	**0.8**	**0.8**
	Iter	79	77	70	75	71	70	72	70	71	74
L-BFGS-TP-FB	Time	**0.8**	0.9	**0.9**	**0.8**	**0.9**	**0.9**	**0.9**	**0.8**	**0.9**	**0.9**
	Iter	62	65	65	60	65	64	67	63	65	64
L-BROY	Time	3.3*	4.7*	4.3*	4.5*	4.6*	4.6*	4.6*	4.6*	4.5*	4.6*
	Iter	1000	1000	1000	1000	1000	1000	1000	1000	1000	1000
L-BROY-LP-F	Time	**0.7**	**0.7**	**0.8**	**0.9**	1.0	**0.9**	**0.9**	1.0	1.0	1.0
	Iter	68	68	78	88	97	88	86	92	94	95
L-BROY-LP-FB	Time	**0.8**	0.9	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0
	Iter	63	69	77	73	79	78	78	77	77	76
L-BROY-TP-F	Time	**0.7**	**0.8**	**0.9**	**0.9**	**0.9**	**0.9**	**0.9**	**0.9**	**0.9**	**0.9**
	Iter	72	83	85	86	85	85	84	84	84	84
L-BROY-TP-FB	Time	**0.8**	1.1	1.0	1.2	1.1	1.1	1.1	1.1	1.1	1.1
	Iter	68	82	76	86	84	85	83	83	83	85
Independent of Window Size											
ALS	Time	58.5*	58.5*	58.5*	58.5*	58.5*	58.5*	58.5*	58.5*	58.5*	58.5*
	Iter	10000	10000	10000	10000	10000	10000	10000	10000	10000	10000
NPNGMRES	Time	8.9	8.9	8.9	8.9	8.9	8.9	8.9	8.9	8.9	8.9
	Iter	452	452	452	452	452	452	452	452	452	452
NPNCG $\widehat{\beta}_{\text{HS}}$	Time	5.0	5.0	5.0	5.0	5.0	5.0	5.0	5.0	5.0	5.0
	Iter	612	612	612	612	612	612	612	612	612	612
NPNCG $\widetilde{\beta}_{\text{HS}}$	Time	5.9	5.9	5.9	5.9	5.9	5.9	5.9	5.9	5.9	5.9
	Iter	708	708	708	708	708	708	708	708	708	708

Table 3.8: Results for modBT line-search for CP decomposition. (Asterisks denote runs which failed to converge. Bold entries indicate the best two times for a given value of m.)

our decomposition. Results for this test are recorded in Tables 3.7 and 3.8, corresponding to MT and modBT, respectively. Entries in bold indicate one of the two lowest times for a given value of m, and entries with asterisks denote a method that failed to converge to the stated tolerance.

First, it is clear that standard L-BFGS and L-Broyden fail to converge within the limits imposed in the vast majority of cases. This agrees with previous experiments which observed that L-BFGS and NCG without preconditioning do not improve upon ALS in terms

of time-to-solution [5]. In comparison, NCG, NGMRES, and L-BFGS methods nonlinearly preconditioned by CP-ALS perform much better when accurate solutions are desired. A general trend observed is that NPQN methods using MT tend to require more time to converge compared to those using modBT, whereas the existing NPNCG and NPNGMRES iterations perform better with the MT line-search. Restricting our attention to the NPQN results in Table 3.8, we see that methods using the forward CP-ALS sweep preconditioner may be slightly faster than the forward-backward variant, though the difference is small enough that we consider both variants for further tests. Increasing window size m may also result in a small increase in computation time. Based on these observations, we will restrict further consideration to NPQN methods using the modBT line-search, NPNCG and NPNGMRES using MT, and window sizes of $m = 1$, $m = 2$, and $m = 10$.

			Trial										
			1	2	3	4	5	6	7	8	9	10	Mean
	CP-ALS	Time	122.0*	122.7*	122.4*	122.8*	123.1*	123.0*	123.2*	130.2*	125.3*	124.6*	123.9*
		Iter	10000	10000	10000	10000	10000	10000	10000	10000	10000	10000	10000
	NPNCG $\widehat{\beta}_{\text{HS}}$	Time	7.5	7.1	3.0	5.1	4.1	4.2	4.6	10.0	4.0	5.0	5.5
		Iter	106	106	67	94	86	78	88	126	80	79	91
	NPNCG $\widetilde{\beta}_{\text{HS}}$	Time	115.5*	3.4	3.5	11.3	4.2	3.5	106.0*	4.2	3.0	10.7	26.5
		Iter	775	81	76	142	92	73	603	93	69	117	212
	NPNGMRES	Time	4.5	5.0	4.1	5.6	3.4	3.8	6.4	4.1	3.6	2.8	4.3
		Iter	91	92	74	108	62	72	118	71	62	52	80
$m = 1$	L-BFGS	Time	14.2*	11.6*	11.7*	12.4*	11.4	* 11.0*	10.9*	11.3*	12.4*	11.5*	11.8*
		Iter	1000	1000	1000	1000	1000	1000	1000	1000	1000	1000	1000
	L-BFGS-LP-F	Time	2.1	1.8	1.6	2.0	2.3	1.5	2.1	2.4	2.4	1.8	2.0
		Iter	71	66	52	67	77	58	72	75	86	60	68
	L-BFGS-LP-FB	Time	2.1	2.4	2.6	2.2	2.0	2.3	2.2	1.9	2.1	1.6	2.1
		Iter	56	55	62	53	51	59	49	49	51	41	53
	L-BFGS-TP-F	Time	2.2	1.9	1.7	2.2	2.1	1.6	2.4	2.3	1.6	1.4	1.9
		Iter	78	67	61	73	80	58	79	77	59	44	68
	L-BFGS-TP-FB	Time	2.3	2.6	2.9	2.5	1.9	2.0	2.5	1.8	2.3	1.4	2.2
		Iter	61	63	72	61	54	58	58	45	60	33	57
	L-BROY	Time	15.8*	12.5*	12.6*	13.0*	11.9*	7.5	4.6	12.2*	13.3*	12.1*	11.5
		Iter	1000	1000	1000	1000	1000	644	359	1000	1000	1000	900
	L-BROY-LP-F	Time	1.7	1.7	1.6	1.8	1.7	1.3	2.1	1.6	1.6	1.5	1.7
		Iter	59	59	59	61	56	48	69	60	56	51	58
	L-BROY-LP-FB	Time	2.4	2.3	2.5	1.9	2.0	2.0	2.8	2.3	1.9	1.2	2.1
		Iter	63	58	65	51	53	54	65	55	48	32	54
	L-BROY-TP-F	Time	2.1	1.4	1.5	1.7	1.7	1.2	1.7	1.5	1.7	1.5	1.6
		Iter	68	56	55	63	65	46	63	58	64	51	59
	L-BROY-TP-FB	Time	2.2	2.6	3.0	2.3	1.8	2.1	2.8	2.4	2.6	1.4	2.3
		Iter	61	68	67	63	53	63	69	58	73	38	61

Table 3.9: Results for computing CP decomposition of a synthetic ($200 \times 200 \times 200$) rank-5 tensor with high colinearity ($C = 0.9$) with significant heteroskedastic ($\ell_1 = 20$) and homoskedastic ($\ell_2 = 10$) noise. Each trial corresponds to a different initial guess. (Asterisks denote runs which failed to converge. Vertical pairs of cells comparing F and FB results are: green, if FB is faster than F; yellow, if they have the same time-to-solution; and red, if F is faster than FB.)

			Trial										
			1	2	3	4	5	6	7	8	9	10	Mean
$m = 2$	L-BFGS	Time	12.8*	11.1*	11.8*	13.0*	11.1*	11.2*	11.0*	11.2*	13.1*	10.8*	11.7*
		Iter	1000	1000	1000	1000	1000	1000	1000	1000	1000	1000	1000
	L-BFGS-LP-F	Time	2.5	2.9	2.3	2.4	2.8	2.2	2.3	2.2	1.8	1.6	2.3
		Iter	78	89	67	73	85	73	73	68	57	53	72
	L-BFGS-LP-FB	Time	2.8	2.6	2.8	2.5	2.2	2.2	2.7	2.2	2.4	1.8	2.4
		Iter	66	61	65	60	53	55	61	49	61	44	58
	L-BFGS-TP-F	Time	2.1	1.6	1.7	2.6	1.6	1.5	2.4	1.8	2.0	1.7	1.9
		Iter	68	55	57	77	55	51	67	60	66	45	60
	L-BFGS-TP-FB	Time	2.8	3.7	3.3	2.1	1.7	2.2	2.6	2.0	2.3	1.9	2.5
		Iter	65	76	70	53	45	58	59	46	60	45	58
	L-BROY	Time	18.1*	18.2*	18.2*	18.2*	19.6*	9.2	11.5	17.6*	18.5*	17.8*	16.7
		Iter	1000	1000	1000	1000	1000	526	642	1000	1000	1000	917
	L-BROY-LP-F	Time	2.0	1.8	1.6	2.2	1.8	1.4	2.1	1.8	1.9	1.5	1.8
		Iter	65	63	54	74	56	48	66	66	61	50	60
	L-BROY-LP-FB	Time	2.3	2.8	2.8	2.3	2.0	2.3	3.1	2.4	2.4	1.4	2.4
		Iter	58	68	71	58	55	61	75	55	58	35	59
	L-BROY-TP-F	Time	2.2	2.3	2.9	2.3	2.6	1.6	2.9	2.2	2.7	1.8	2.4
		Iter	69	76	74	75	88	56	88	77	83	57	74
	L-BROY-TP-FB	Time	2.8	2.8	3.9	2.5	2.4	2.7	3.1	2.3	2.9	1.7	2.7
		Iter	73	72	70	62	60	72	72	55	77	39	65
$m = 10$	L-BFGS	Time	11.9*	11.4*	11.5*	12.6*	16.9*	10.7*	11.2*	11.2*	10.8*	11.1*	11.9*
		Iter	1000	1000	1000	1000	1000	1000	1000	1000	1000	1000	1000
	L-BFGS-LP-F	Time	2.1	2.0	2.4	2.0	2.3	1.8	2.8	2.7	2.0	1.6	2.2
		Iter	62	63	69	64	70	51	81	78	57	54	65
	L-BFGS-LP-FB	Time	2.4	2.1	2.7	2.4	2.2	2.3	2.4	2.4	1.9	1.8	2.3
		Iter	58	52	62	55	53	56	52	56	46	44	53
	L-BFGS-TP-F	Time	2.0	1.9	2.5	2.3	2.3	1.6	2.1	1.9	2.0	1.8	2.0
		Iter	65	62	74	69	75	53	58	59	61	49	63
	L-BFGS-TP-FB	Time	2.7	2.8	3.5	2.4	2.6	2.0	3.7	2.2	2.6	1.5	2.6
		Iter	61	65	80	59	64	55	62	48	64	34	59
	L-BROY	Time	18.4*	18.2*	18.2*	18.3*	18.3*	17.7*	17.6*	17.2*	17.9*	18.7*	18.1*
		Iter	1000	1000	1000	1000	1000	1000	1000	1000	1000	1000	1000
	L-BROY-LP-F	Time	2.4	1.9	1.9	2.0	2.3	2.4	5.5	2.0	2.2	1.4	2.4
		Iter	79	66	66	65	77	79	107	69	72	48	73
	L-BROY-LP-FB	Time	2.6	2.7	2.6	2.6	2.3	2.4	3.8	2.7	2.3	1.7	2.6
		Iter	67	69	67	65	58	62	76	63	60	43	63
	L-BROY-TP-F	Time	2.5	2.1	2.7	2.0	2.5	1.9	2.8	2.3	2.5	2.0	2.3
		Iter	75	67	89	63	79	66	78	81	81	60	74
	L-BROY-TP-FB	Time	3.0	3.3	3.2	2.9	2.7	2.2	3.9	2.8	3.0	2.2	2.9
		Iter	73	86	74	71	69	58	89	64	78	49	71

Table 3.9: continued

As a basis of comparison we again use the previous method to form a test tensor with specified colinearity and noise levels. We decompose an order-3 tensor of size $(I \times I \times I)$ for $I = 200$ with known rank $R = 5$, colinearity $C = 0.9$, and set noise parameters $\ell_1 = 20$ and $\ell_2 = 10$. For this problem we run ten trials, with each trial corresponding to a different random initial guess. The results for this test are recorded in Table 3.9. For this problem we observe that the CP-ALS algorithm fails to converge for each trial, and that NPNCG fails twice for $\widetilde{\beta}$. Outside of the NPQN methods, NPNGMRES converged consistently and in most cases exhibited the quickest time-to-solution. However, we observed that all

				Trial										
				1	2	3	4	5	6	7	8	9	10	Mean
$m = 1$	L-BFGS	LP	Time	2.1	1.8	1.6	2.0	2.3	1.5	2.1	2.4	2.4	1.8	2.0
			Iter	71	66	52	67	77	58	72	75	86	60	68
		TP	Time	2.2	1.9	1.7	2.2	2.1	1.6	2.4	2.3	**1.6**	**1.4**	1.9
			Iter	78	67	61	73	80	58	79	77	59	44	68
	L-BROY	LP	Time	**1.7**	1.7	1.6	1.8	1.7	1.3	2.1	1.6	**1.6**	1.5	1.7
			Iter	59	59	59	61	56	48	69	60	56	51	58
		TP	Time	2.1	**1.4**	**1.5**	**1.7**	1.7	**1.2**	**1.7**	**1.5**	1.7	1.5	**1.6**
			Iter	68	56	55	63	65	46	63	58	64	51	59
$m = 2$	L-BFGS	LP	Time	2.5	2.9	2.3	2.4	2.8	2.2	2.3	2.2	1.8	1.6	2.3
			Iter	78	89	67	73	85	73	73	68	57	53	72
		TP	Time	2.1	1.6	1.7	2.6	**1.6**	1.5	2.4	1.8	2.0	1.7	1.9
			Iter	68	55	57	77	55	51	67	60	66	45	60
	L-BROY	LP	Time	2.0	1.8	1.6	2.2	1.8	1.4	2.1	1.8	1.9	1.5	1.8
			Iter	65	63	54	74	56	48	66	66	61	50	60
		TP	Time	2.2	2.3	2.9	2.3	2.6	1.6	2.9	2.2	2.7	1.8	2.4
			Iter	69	76	74	75	88	56	88	77	83	57	74
$m = 10$	L-BFGS	LP	Time	2.1	2.0	2.4	2.0	2.3	1.8	2.8	2.7	2.0	1.6	2.2
			Iter	62	63	69	64	70	51	81	78	57	54	65
		TP	Time	2.0	1.9	2.5	2.3	2.3	1.6	2.1	1.9	2.0	1.8	2.0
			Iter	65	62	74	69	75	53	58	59	61	49	63
	L-BROY	LP	Time	2.4	1.9	1.9	2.0	2.3	2.4	5.5	2.0	2.2	1.4	2.4
			Iter	79	66	66	65	77	79	107	69	72	48	73
		TP	Time	2.5	2.1	2.7	2.0	2.5	1.9	2.8	2.3	2.5	2.0	2.3
			Iter	75	67	89	63	79	66	78	81	81	60	74

Table 3.10: Excerpt of Table 3.9 showing only NPQN results which used the forward CP-ALS sweep as preconditioner. (Bold entries indicate the best time for a given trial.)

NPQN methods tested outperformed the remaining methods in terms of solution time in nearly all cases, with improvements of more than 50% for the best new method.

To compare the forward and forward-backward CP-ALS preconditioners for a given method, we color the pair of cells red if the forward sweep is faster, yellow if the times are equal (when rounded to the nearest 0.1 second), and green if the forward-backward sweep is faster. By doing so we can see at a glance that the forward sweep ALS preconditioner is the best option in the majority of cases. For further examination we turn to Table 3.10, which contains only the results for forward CP-ALS preconditioned NPQN methods. Here the best time for each trial is indicated in bold. The results indicate increasing window size generally increases the time-to-solution, though there are some exceptions. Overall, the

lowest times for the majority of trials are for $m = 1$. It is interesting to observe that the L-Broyden iterations typically gave the best performance in this case, rather than L-BFGS.

3.2.2 Optimization over Matrix Manifolds

We performed three sets of tests focused on NPQN methods: the first to determine what combinations of methods, line-searches, and parameters worked best, and the remaining two to compare NPQN L-BFGS and L-Broyden to NPNGMRES and NPNCG for synthetic and real-life data tensors of different sizes and noise levels.

We first considered all possible combinations of L-BFGS and L-Broyden with left preconditioning, transformation preconditioning, or no preconditioning; this time using a forward or forward-backward HOOI sweep (Algorithm 3, lines 2–5 with either $n = 1, \ldots, N$ or $n = 1, 2 \ldots, N, N-1, \ldots, 1$) as $\mathcal{Q}(\mathbf{g}; \mathbf{x})$. For L-Broyden we set θ_k to use the γ_k formula corresponding to the equivalent L-BFGS variant because this gave the best results. To narrow down the set of variants we (i) compared all methods in terms of choice of line-search, and (ii) compared the L-BFGS and L-Broyden methods in terms of window parameter $m \in \{1, \ldots, 10\}$ and whether or not vector transport is used when updating the Hessian approximations.

We again consider MT and modBT, using the same parameters as for the CP problem. The same reset conditions are used for MT, and for modBT the `while` condition now uses $f_{k+1} > (1 - e^{-2 \cdot \text{iter}}) f_k$, as objective values are negative for the Tucker HOSVD problem. In both cases the QN approximation is reset by clearing $\mathbf{S}_k$ and $\mathbf{Y}_k$. The NPNGMRES system grows to a maximum of $w = 25$. If NPNGMRES produces an ascent search direction $\mathbf{p}_k$, we discard all past iterates and search in the direction $-\mathbf{p}_k$. NPNCG methods are restarted every 50 iterations. We again use the two HS β update parameters from (2.13) and (2.16). Successful termination occurs when $\|\mathbf{g}_k\|_F / |f(\mathbf{x}_k)| < 10^{-7}$.

The tests involved decomposing a large order-3 tensor formed using a subset of the MNIST Database of Handwritten Digits [67]. Our test consisted of computing a rank $(14, 14, 100)$ approximation of a tensor representing 2500 images of the digit 5 with significant additive noise from a uniform distribution over $[0, 1]$: $\mathcal{X}' = \mathcal{X} + 2.5 \frac{\|\mathcal{X}\|}{\|\mathcal{N}\|} \mathcal{N}$. Ten trials were ran for each combination, with each trial corresponding to a different tensor $\mathcal{N}$. Iterations continued until reaching a maximum of 250 iterations, a total execution time greater than 1500 seconds, or $\|\mathbf{g}_k\| / |f(\mathbf{x}_k)| < 10^{-7}$. The mean time-to-solution and iterations required for each test case are recorded in Tables 3.11 and 3.12. Entries in bold denote one of the two lowest times for a given window size, and asterisks indicate runs that did not converge.

		Window Size (m)									
		1	2	3	4	5	6	7	8	9	10
Hessian Update without Vector Transport											
L-BFGS	Time	155.0*	193.6*	243.9*	267.9*	275.7*	297.1*	309.4*	320.0*	331.8*	332.3*
	Iter	251	251	251	251	251	251	251	251	251	251
L-BFGS-LP-F	Time	98.7	133.1	132.1	128.6	145.5	145.3	147.8	149.1	138.3	149.0
	Iter	67	95	100	101	111	108	112	114	101	118
L-BFGS-LP-FB	Time	72.9	111.6	124.0	113.0	135.1	139.9	143.0	126.9	138.0	127.8
	Iter	45	74	85	81	101	98	107	90	96	94
L-BFGS-TP-F	Time	85.1	93.8	95.3	111.3	82.4	86.8	85.2	84.8	87.4	85.6
	Iter	54	50	52	58	51	52	54	55	56	57
L-BFGS-TP-FB	Time	75.6	87.4	86.6	82.3	83.6	80.8	88.0	73.1	82.1	84.3
	Iter	46	46	47	47	48	50	51	49	54	56
L-BROY	Time	250.1*	270.3*	281.1*	277.4*	288.3*	291.7*	295.8*	294.9*	293.7*	297.8*
	Iter	251	251	251	251	251	251	251	251	251	251
L-BROY-LP-F	Time	113.0	123.6	125.7	120.0	134.1	138.8	120.5	132.5	132.5	129.7
	Iter	77	94	92	95	98	105	95	103	100	99
L-BROY-LP-FB	Time	108.3	104.1	131.1	124.9	115.9	130.1	128.9	129.8	138.5	129.0
	Iter	70	79	94	92	84	99	94	97	99	97
L-BROY-TP-F	Time	96.3	100.0	119.2	110.6	101.1	113.0	107.5	103.7	104.0	104.9
	Iter	60	51	58	55	53	56	54	53	55	53
L-BROY-TP-FB	Time	75.1	93.5	91.2	98.3	98.3	89.1	96.9	104.6	97.3	96.9
	Iter	46	48	46	47	49	46	50	50	48	49
Hessian Update with Vector Transport											
L-BFGS-LP-F	Time	117.0	161.5	183.1	176.9	208.6	188.1	193.9	211.9	200.5	196.5
	Iter	67	92	108	96	119	106	106	119	110	107
L-BFGS-LP-FB	Time	85.6	128.5	144.3	177.0	177.6	183.7	183.8	163.1	182.8	178.8
	Iter	45	68	78	96	96	101	96	85	97	93
L-BFGS-TP-F	Time	98.3	113.9	114.8	111.9	115.3	111.5	108.0	124.6	114.9	117.9
	Iter	54	51	51	52	52	53	53	60	55	59
L-BFGS-TP-FB	Time	87.9	110.7	98.2	127.5	105.4	105.8	104.4	110.3	108.3	110.4
	Iter	46	48	45	56	49	48	50	51	52	54
L-BROY-LP-F	Time	134.1	160.3	143.9	167.2	155.8	181.5	188.0	175.9	180.3	193.0
	Iter	77	96	82	95	88	104	104	99	98	104
L-BROY-LP-FB	Time	129.4	146.8	143.8	152.5	149.3	187.5	180.7	196.6	190.2	185.6
	Iter	70	91	81	86	85	104	94	107	100	96
L-BROY-TP-F	Time	113.9	116.5	126.7	118.5	122.9	118.6	119.6	118.9	123.3	117.5
	Iter	60	53	54	53	55	54	54	53	54	53
L-BROY-TP-FB	Time	89.3	113.3	109.9	102.8	107.7	109.1	106.4	110.7	111.5	105.4
	Iter	46	49	48	46	48	48	47	47	48	46

Table 3.11: Tucker decomposition results for MT line-search with varying m. (Asterisks denote runs which failed to converge. Bold entries indicate the best two times for a given value of m.)

		Window Size (m)									
		1	2	3	4	5	6	7	8	9	10
Independent of Window Size											
HOOI	Time	112.8	112.8	112.8	112.8	112.8	112.8	112.8	112.8	112.8	112.8
	Iter	210	210	210	210	210	210	210	210	210	210
NPNGMRES	Time	**44.9**	**44.9**	**44.9**	**44.9**	**44.9**	**44.9**	**44.9**	**44.9**	**44.9**	**44.9**
	Iter	35	35	35	35	35	35	35	35	35	35
NPNCG $\widehat{\beta}_{\text{HS}}$	Time	**33.0**	**33.0**	**33.0**	**33.0**	**33.0**	**33.0**	**33.0**	**33.0**	**33.0**	**33.0**
	Iter	37	37	37	37	37	37	37	37	37	37
NPNCG $\widetilde{\beta}_{\text{HS}}$	Time	67.2	67.2	67.2	67.2	67.2	667.2	67.2	67.2	67.2	67.2
	Iter	72	72	72	72	72	72	72	72	72	72

Table 3.11: continued

		Window Size (m)									
		1	2	3	4	5	6	7	8	9	10
Hessian Update without Vector Transport											
L-BFGS	Time	61.7*	65.2*	65.8*	65.7*	66.4*	66.9*	66.3*	66.2*	66.1*	66.3*
	Iter	251	251	251	251	251	251	251	251	251	251
L-BFGS-LP-F	Time	31.6	33.2	38.0	35.3	36.2	34.1	34.3	34.3	34.3	33.7
	Iter	50	51	58	54	54	51	51	51	51	50
L-BFGS-LP-FB	Time	30.9	33.6	34.1	31.8	**32.0**	33.2	32.5	32.8	**31.8**	**31.3**
	Iter	45	48	49	45	45	46	45	46	44	43
L-BFGS-TP-F	Time	**28.0**	31.0	31.6	**31.2**	32.7	**32.7**	**31.5**	33.6	**32.6**	33.2
	Iter	39	42	43	41	43	42	41	43	41	42
L-BFGS-TP-FB	Time	**29.1**	**28.5**	**28.8**	**26.8**	**30.4**	**31.0**	**31.7**	**30.5**	33.0	33.7
	Iter	38	37	37	33	38	38	39	37	40	41
L-BROY	Time	60.4*	62.7*	63.0*	64.1*	63.4*	63.8*	64.3*	63.6*	63.7*	64.0*
	Iter	251	251	251	251	251	251	251	251	251	251
L-BROY-LP-F	Time	45.8	53.8	45.7	48.6	49.8	44.1	35.9	50.1	52.9	47.6
	Iter	71	81	69	72	75	66	54	75	79	71
L-BROY-LP-FB	Time	32.7	38.0	42.2	31.3	40.0	39.4	33.7	35.5	35.3	33.2
	Iter	48	55	60	44	57	55	48	50	50	47
L-BROY-TP-F	Time	40.5	34.3	32.8	36.2	45.2	37.7	37.3	37.1	37.3	38.0
	Iter	57	47	44	48	60	49	49	48	49	49
L-BROY-TP-FB	Time	36.6	**29.1**	**29.5**	32.1	34.5	33.4	34.9	**32.7**	33.5	**32.9**
	Iter	48	38	37	40	43	41	43	40	41	40

Table 3.12: Tucker decomposition results for modBT line-search with varying m. (Asterisks denote runs which failed to converge. Bold entries indicate the best two times for a given value of m.)

		Window Size (m)									
		1	2	3	4	5	6	7	8	9	10
Hessian Update with Vector Transport											
L-BFGS-LP-F	Time	46.4	52.8	53.9	54.4	54.5	54.5	53.4	52.2	57.3	60.5
	Iter	50	52	52	53	52	51	48	48	51	54
L-BFGS-LP-FB	Time	44.3	51.5	49.3	47.2	47.1	47.6	49.8	48.6	52.8	52.3
	Iter	45	49	45	44	42	42	44	42	46	45
L-BFGS-TP-F	Time	44.8	55.7	54.8	55.3	58.2	58.5	61.2	62.3	61.5	63.4
	Iter	39	43	40	42	42	42	42	43	41	43
L-BFGS-TP-FB	Time	45.6	49.5	52.3	46.7	54.8	56.8	58.8	56.6	62.4	61.6
	Iter	38	37	38	34	38	39	39	37	41	40
L-BROY-LP-F	Time	63.0	58.6	52.5	57.9	68.5	67.9	54.5	53.9	63.7	60.8
	Iter	72	64	53	60	72	72	53	54	67	60
L-BROY-LP-FB	Time	53.8	61.1	53.3	48.0	52.8	58.1	52.7	49.5	47.9	49.9
	Iter	56	63	52	47	53	55	50	46	44	45
L-BROY-TP-F	Time	62.9	60.5	62.2	65.3	64.0	60.8	65.2	64.0	62.7	61.5
	Iter	55	48	47	56	49	47	50	51	51	50
L-BROY-TP-FB	Time	60.3	48.5	53.6	56.5	54.6	59.0	59.6	59.2	57.5	57.1
	Iter	50	37	40	44	41	44	43	43	42	42
Independent of Window Size											
HOOI	Time	112.8	112.8	112.8	112.8	112.8	112.8	112.8	112.8	112.8	112.8
	Iter	210	210	210	210	210	210	210	210	210	210
NPNGMRES	Time	35.8	35.8	35.8	35.8	35.8	35.8	35.8	35.8	35.8	35.8
	Iter	33	33	33	33	33	33	33	33	33	33
NPNCG $\widehat{\beta}_{\text{HS}}$	Time	146.0	146.0	146.0	146.0	146.0	146.0	146.0	146.0	146.0	146.0
	Iter	210	210	210	210	210	210	210	210	210	210
NPNCG $\widetilde{\beta}_{\text{HS}}$	Time	130.1	130.1	130.1	130.1	130.1	130.1	130.1	130.1	130.1	130.1
	Iter	210	210	210	210	210	210	210	210	210	210

Table 3.12: continued

When working in a manifold framework, we may or may not use a vector transport operation when updating the Hessian approximation between iterations (details are provided in § 1.4). These tables contain results for both of these possibilities, from which it is clear that omitting this vector transport step results in faster methods, in particular when the modBT line-search is used. Because of this, we exclude the vector transport option from further consideration. Next, note that the non-preconditioned methods again failed to converge within the maximum number of steps in every case, which was expected based on the CP results. When comparing the line-search methods, MT results in slower convergence for all but the NPNCG algorithms. With respect to window size m, while the results are often quite good for all values considered, it is clear that smaller values of m typically produce better results. As such, we only consider the values of $m = 1$ and $m = 2$

for subsequent tests, and use the modBT line-search for all methods except NPNCG.

For our second test we computed rank-$(20, 20, 20)$ HOSVDs of rank-$(40, 40, 40)$ synthetic tensors of size $(120, 120, 120)$, using noise parameters $\ell_1 = \ell_2 = 10$ and noise tensors $\mathcal{N}_1$, $\mathcal{N}_2$ with elements from $\mathcal{N}(0, 1)$; see (3.4). Ten trials using different noise tensors $\mathcal{N}_1$ and $\mathcal{N}_2$ were carried out for each method. The results, recorded in Table 3.13, indicate that HOOI is the slowest of the pre-existing methods, typically followed by NPNGMRES and then NPNCG, with no clear winner between the $\widehat{\beta}$ and $\widetilde{\beta}$ variants. For this problem the forward HOOI sweep is by far the best preconditioner for NPQN iterations. The majority of L-BFGS results are competitive with or improve upon the existing solvers. The L-Broyden iterations are less effective, though many are still competitive with the NPNCG and NPNGMRES results.

Referring to the subset of data in Table 3.14, which includes only the NPQN methods using the forward HOOI preconditioner, we see that the fastest results are for nonlinearly left-preconditioned L-BFGS in all but one case, which is a narrow loss. More generally, from the average times we see that all of the proposed methods using this preconditioner generally perform at least as well as the best pre-existing methods, with an improvement of more than 50% for the best new method.

		Trial										
		1	2	3	4	5	6	7	8	9	10	Mean
HOOI	Time	16.1	42.8	53.8	35.2	10.4	24.7	15.8	17.9	19.8	10.8	24.7
	Iter	398	1138	1436	933	283	650	421	470	508	284	652
NPNCG $\widehat{\beta}_{\text{HS}}$	Time	9.2	15.6	12.8	11.4	5.8	9.7	6.5	8.4	9.5	6.6	9.5
	Iter	101	161	135	118	62	100	68	86	101	64	100
NPNCG $\widetilde{\beta}_{\text{HS}}$	Time	9.1	19.3	14.0	12.0	6.0	9.5	8.1	7.6	8.9	6.5	10.1
	Iter	91	193	151	124	63	101	70	78	95	62	103
NPNGMRES	Time	11.4	13.3	13.4	21.0	5.8	12.5	8.0	10.9	8.1	7.8	11.2
	Iter	92	114	107	171	48	103	66	79	68	60	91

Table 3.13: Results for decomposing a noisy $(120 \times 120 \times 120)$ tensor into a rank-$(20, 20, 20)$ Tucker HOSVD ATD. (Each trial corresponds to different noise tensors with entries from the standard normal distribution. Asterisks denote runs which failed to converge. Vertical pairs of cells comparing F and FB results are: green, if FB is faster than F; and red, if F is faster than FB.)

			Trial										
			1	2	3	4	5	6	7	8	9	10	Mean
$m = 1$	L-BFGS	Time	6.3	6.9	9.4	13.4	8.3	9.9	7.7	15.0	6.4	6.6	9.0
		Iter	163	220	303	402	261	304	250	479	202	198	278
	L-BFGS-LP-F	Time	5.6	5.0	5.0	6.7	3.3	4.2	2.8	4.1	4.1	3.0	4.4
		Iter	113	103	104	135	66	87	58	79	83	59	89
	L-BFGS-LP-FB	Time	5.5	7.3	11.2	10.3	4.4	6.5	4.6	8.1	4.7	5.6	6.8
		Iter	74	97	149	130	58	84	60	95	62	73	88
	L-BFGS-TP-F	Time	6.3	5.7	6.0	6.6	4.7	6.1	4.7	6.5	4.9	4.4	5.6
		Iter	93	84	87	92	68	87	67	85	69	63	80
	L-BFGS-TP-FB	Time	7.6	9.0	16.0	11.5	5.5	8.5	6.1	7.8	6.7	7.4	8.6
		Iter	80	88	159	114	57	84	61	74	69	77	86
	L-BROY	Time	13.2	13.6	12.1	10.6	11.3	9.1	10.4	7.9	20.2	7.3	11.6
		Iter	397	424	380	300	298	245	302	238	617	212	341
	L-BROY-LP-F	Time	5.2	6.9	15.9	10.8	4.5	6.2	5.5	6.4	4.5	4.3	7.0
		Iter	105	121	285	192	90	122	103	113	89	86	131
	L-BROY-LP-FB	Time	8.3	9.2	18.8	12.3	5.5	8.8	7.0	13.2	7.5	10.0	10.1
		Iter	109	122	233	160	66	114	86	158	94	129	127
	L-BROY-TP-F	Time	7.9	18.1	9.5	13.3	6.8	10.2	7.3	9.5	6.6	6.8	9.6
		Iter	114	261	128	185	96	132	104	128	92	97	134
	L-BROY-TP-FB	Time	10.7	13.0	24.7	13.8	7.0	12.3	8.4	18.0	8.3	9.8	12.6
		Iter	111	136	242	142	72	118	85	186	85	102	128
$m = 2$	L-BFGS	Time	6.7	8.6	6.5	7.5	13.7	7.0	9.7	8.2	7.6	7.0	8.3
		Iter	182	266	210	243	375	201	283	265	237	227	249
	L-BFGS-LP-F	Time	4.2	4.5	5.0	6.8	3.0	4.6	2.9	5.7	3.7	3.2	4.3
		Iter	87	90	104	141	61	92	59	112	76	65	89
	L-BFGS-LP-FB	Time	5.0	7.5	11.2	10.1	4.3	7.6	4.7	8.3	5.7	6.5	7.1
		Iter	66	96	147	132	55	99	58	110	75	86	92
	L-BFGS-TP-F	Time	6.8	6.2	5.9	11.4	4.4	6.3	5.2	6.7	4.8	4.3	6.2
		Iter	95	90	85	154	61	91	69	95	68	63	87
	L-BFGS-TP-FB	Time	8.3	7.0	15.1	12.9	5.7	9.7	6.8	7.1	6.8	7.5	8.7
		Iter	80	72	156	130	58	99	63	72	70	77	88
	L-BROY	Time	71.4*	67.6*	72.5*	70.9*	36.3	71.1*	70.4*	69.0*	16.1	11.6	55.7
		Iter	2000	2000	2000	2000	1029	2000	2000	2000	440	348	1581
	L-BROY-LP-F	Time	5.8	10.1	15.5	11.5	3.9	6.1	4.7	4.7	4.8	3.7	7.1
		Iter	109	188	278	197	72	114	89	89	88	72	130
	L-BROY-LP-FB	Time	7.5	10.6	15.6	10.3	4.7	12.9	6.6	13.9	5.7	9.2	9.7
		Iter	91	124	192	129	59	154	80	167	70	113	118
	L-BROY-TP-F	Time	8.1	14.5	19.4	12.7	5.0	8.2	5.5	7.6	5.6	6.2	9.3
		Iter	112	199	254	175	67	116	76	104	78	84	127
	L-BROY-TP-FB	Time	9.9	11.2	16.9	17.2	6.4	11.6	7.9	8.5	8.1	13.4	11.1
		Iter	99	113	170	171	61	117	70	81	80	126	109

Table 3.13: continued

				Trial										
				1	2	3	4	5	6	7	8	9	10	Mean
$m = 1$	L-BFGS	LP	Time	5.6	5.0	**5.0**	6.7	3.3	**4.2**	**2.8**	**4.1**	4.1	**3.0**	4.4
			Iter	113	103	104	135	66	87	58	79	83	59	89
		TP	Time	6.3	5.7	6.0	**6.6**	4.7	6.1	4.7	6.5	4.9	4.4	5.6
			Iter	93	84	87	92	68	87	67	85	69	63	80
	L-BROY	LP	Time	5.2	6.9	15.9	10.8	4.5	6.2	5.5	6.4	4.5	4.3	7.0
			Iter	105	121	285	192	90	122	103	113	89	86	131
		TP	Time	7.9	18.1	9.5	13.3	6.8	10.2	7.3	9.5	6.6	6.8	9.6
			Iter	114	261	128	185	96	132	104	128	92	97	134
$m = 2$	L-BFGS	LP	Time	**4.2**	**4.5**	**5.0**	6.8	**3.0**	4.6	2.9	5.7	**3.7**	3.2	**4.3**
			Iter	87	90	104	141	61	92	59	112	76	65	89
		TP	Time	6.8	6.2	5.9	11.4	4.4	6.3	5.2	6.7	4.8	4.3	6.2
			Iter	95	90	85	154	61	91	69	95	68	63	87
	L-BROY	LP	Time	5.8	10.1	15.5	11.5	3.9	6.1	4.7	4.7	4.8	3.7	7.1
			Iter	109	188	278	197	72	114	89	89	88	72	130
		TP	Time	8.1	14.5	19.4	12.7	5.0	8.2	5.5	7.6	5.6	6.2	9.3
			Iter	112	199	254	175	67	116	76	104	78	84	127

Table 3.14: Excerpt of Table 3.13 showing only NPQN results which used the forward HOOI sweep as preconditioner. (Bold entries indicate the best time for a given trial.)

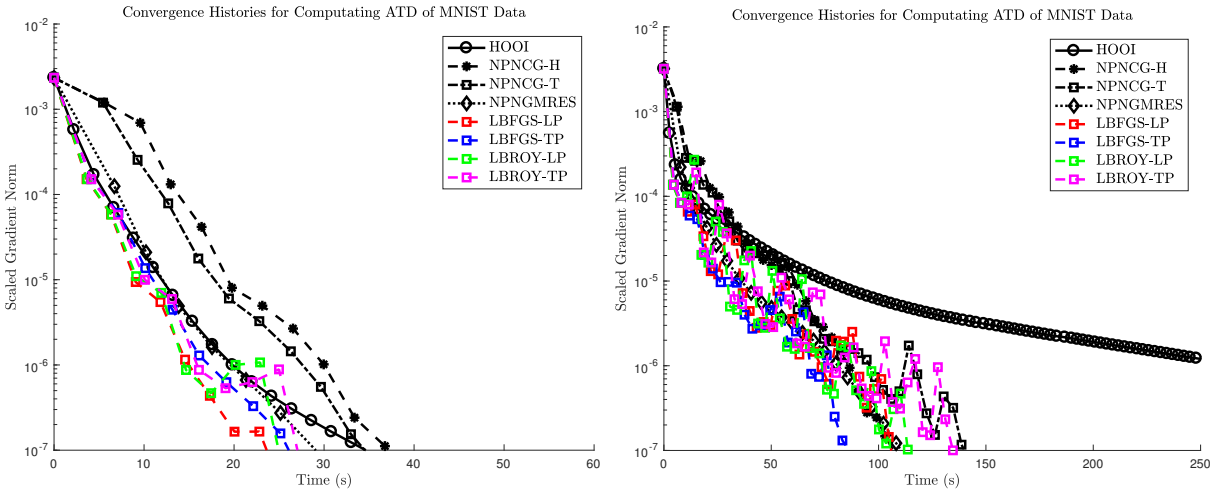

Figure 3.5: Convergence histories showing scaled gradient norms for rank-$(14, 14, 100)$ Tucker HOSVD decompositions of the $(28 \times 28 \times 5000)$ digit tensor without noise (left) and with added noise (right).

			Trial										
			1	2	3	4	5	6	7	8	9	10	Mean
	HOOI	Time	534.8	332.9	383.3	408.8	405.0	387.5	263.3	666.3*	434.3	666.4*	448.3
		Iter	187	125	144	153	152	145	99	250	163	250	167
	NPNCG $\widehat{\beta}_{\text{HS}}$	Time	105.4	92.4	97.7	96.7	134.8	105.9	83.7	151.7	118.2	170.6	115.7
		Iter	25	23	24	24	33	26	21	37	29	43	29
	NPNCG $\widetilde{\beta}_{\text{HS}}$	Time	146.4	123.1	108.8	126.9	287.7	349.2	91.1	999.9	155.8	211.0	260.0
		Iter	35	31	27	32	70	85	23	218	39	54	61
	NPNGMRES	Time	114.8	96.7	102.4	102.5	141.2	137.8	85.3	172.1	113.5	150.6	121.7
		Iter	22	20	21	21	29	27	18	36	23	32	25
$m = 1$	L-BFGS	Time	207.4*	197.3*	213.8*	206.5*	218.6*	194.6*	200.0*	197.1*	189.2*	191.7*	201.6*
		Iter	250	250	250	250	250	250	250	250	250	250	250
	L-BFGS-LP-F	Time	120.2	98.3	91.6	92.0	183.4	131.0	78.7	136.4	91.8	167.9	119.1
		Iter	36	31	29	29	57	41	25	43	29	53	37
	L-BFGS-LP-FB	Time	127.5	78.3	81.6	114.2	152.1	131.3	65.3	124.5	110.8	134.2	112.0
		Iter	32	24	25	35	46	40	20	38	34	41	34
	L-BFGS-TP-F	Time	127.2	98.8	88.5	95.9	119.3	109.0	78.5	132.6	109.1	159.6	111.9
		Iter	30	29	26	28	35	32	23	39	32	47	32
	L-BFGS-TP-FB	Time	90.9	84.4	87.8	91.1	122.9	87.8	73.7	122.6	95.0	154.1	101.0
		Iter	24	24	25	26	35	25	21	35	27	44	29
	L-BROY	Time	197.2*	195.2*	206.4*	200.2*	185.6*	209.8*	195.2*	191.5*	189.8*	189.5*	196.0*
		Iter	250	250	250	250	250	250	250	250	250	250	250
	L-BROY-LP-F	Time	113.6	113.8	129.5	142.3	182.0	104.0	78.9	222.9	122.4	180.1	138.9
		Iter	34	36	41	45	57	33	25	70	39	57	44
	L-BROY-LP-FB	Time	119.2	97.3	130.1	110.5	216.9	118.1	87.9	190.0	136.5	185.6	139.2
		Iter	35	30	40	34	66	36	27	58	42	57	43
	L-BROY-TP-F	Time	194.2	122.6	142.9	136.2	146.7	160.0	115.5	203.3	163.4	190.0	157.5
		Iter	53	36	42	40	43	47	34	60	48	56	46
	L-BROY-TP-FB	Time	144.8	119.5	108.6	136.4	105.2	129.1	101.9	178.2	143.4	167.8	133.5
		Iter	39	34	31	39	30	37	29	51	41	48	38
$m = 2$	L-BFGS	Time	201.9*	213.4*	208.2*	197.0*	202.2*	219.5*	209.4*	199.3*	194.0*	201.6*	204.7*
		Iter	250	250	250	250	250	250	250	250	250	250	250
	L-BFGS-LP-F	Time	120.6	82.4	94.8	101.6	287.8	112.5	88.9	160.1	102.0	201.6	135.2
		Iter	37	26	30	32	89	35	28	50	32	63	42
	L-BFGS-LP-FB	Time	101.4	86.0	85.3	84.9	208.8	91.8	72.3	151.3	101.3	144.1	112.7
		Iter	31	26	26	26	63	28	22	46	31	44	34
	L-BFGS-TP-F	Time	130.2	92.8	102.6	106.8	134.0	105.8	79.0	136.9	99.3	140.0	112.7
		Iter	38	27	30	31	39	31	23	40	29	41	33
	L-BFGS-TP-FB	Time	98.7	84.8	98.7	88.4	123.7	98.5	74.3	113.0	98.6	115.7	99.5
		Iter	28	24	28	25	35	28	21	32	28	33	28
	L-BROY	Time	198.5*	213.2*	208.3*	197.4*	204.0*	205.3*	209.2*	197.6*	196.8*	196.9*	202.7*
		Iter	250	250	250	250	250	250	250	250	250	250	250
	L-BROY-LP-F	Time	121.2	125.5	98.9	112.0	222.1	160.0	79.8	153.5	131.1	217.7	142.2
		Iter	38	39	31	35	69	50	25	48	41	68	44
	L-BROY-LP-FB	Time	115.1	119.0	91.9	95.2	158.4	121.5	72.0	190.8	101.3	164.1	123.0
		Iter	35	36	28	29	48	37	22	58	31	50	37
	L-BROY-TP-F	Time	113.1	99.4	129.9	109.6	140.8	99.6	89.3	147.4	116.9	184.4	123.0
		Iter	33	29	38	32	41	29	26	43	34	54	36
	L-BROY-TP-FB	Time	105.4	88.2	88.2	84.6	123.7	109.1	81.0	154.4	98.7	158.6	109.2
		Iter	30	25	25	24	35	31	23	44	28	45	31

Table 3.15: Results corresponding to decomposing a noisy $(5000 \times 28 \times 28)$ tensor into a rank-$(100, 14, 14)$ Tucker HOSVD ATD. (Each trial corresponds to a different noise tensor. Asterisks denote runs which failed to converge. Vertical pairs of cells comparing F and FB results are: green, if FB is faster than F; and red, if F is faster than FB.)

For the final test we again used the MNIST Database from the initial test, doubling the number of images to form a tensor $\mathcal{X} \in \mathbb{R}^{28\times 28\times 5000}$ consisting of 5000 images of the digit 5. We add uniformly distributed noise to obtain $\mathcal{X}' = \mathcal{X} + 2.5\frac{\|\mathcal{X}\|}{\|\mathcal{N}\|}\mathcal{N}$, where $\mathcal{N}$ has entries in $[0, 1]$. Convergence histories in Figure 3.5 compare the performance of HOOI, NPNCG, NGMRES, and NPQN methods for a test tensor without (left) and with (right) noise. L-BFGS and L-Broyden without preconditioning are not convergent for these kinds of problems, and hence plots for these solvers are omitted. These plots show that, in the easier noise-free case, there is, unsurprisingly, only a small benefit to accelerating HOOI, with NPNGMRES and the NPQN methods all performing slightly better than HOOI, and the NPNCG methods performing slightly worse. Once noise is introduced, however, the convergence of HOOI slows down significantly, and there are clear benefits to using nonlinearly preconditioned methods. In general: nonlinear preconditioning is useful for difficult problems when high accuracy is required, and the low amount of overhead required means it does not harm convergence in other circumstances, improving the overall robustness of solvers.

				Trial										
				1	2	3	4	5	6	7	8	9	10	Mean
$m = 1$	L-BFGS	LP	Time	127.5	**78.3**	**81.6**	114.2	152.1	131.3	**65.3**	124.5	110.8	134.2	112.0
			Iter	32	24	25	35	46	40	20	38	34	41	34
		TP	Time	**90.9**	84.4	87.8	91.1	122.9	**87.8**	73.7	122.6	**95.0**	154.1	101.0
			Iter	24	24	25	26	35	25	21	35	27	44	29
	L-BROY	LP	Time	119.2	97.3	130.1	110.5	216.9	118.1	87.9	190.0	136.5	185.6	139.2
			Iter	35	30	40	34	66	36	27	58	42	57	43
		TP	Time	144.8	119.5	108.6	136.4	**105.2**	129.1	101.9	178.2	143.4	167.8	133.5
			Iter	39	34	31	39	30	37	29	51	41	48	38
$m = 2$	L-BFGS	LP	Time	101.4	86.0	85.3	84.9	208.8	91.8	72.3	151.3	101.3	144.1	112.7
			Iter	31	26	26	26	63	28	22	46	31	44	34
		TP	Time	98.7	84.8	98.7	88.4	123.7	98.5	74.3	**113.0**	98.6	**115.7**	**99.5**
			Iter	28	24	28	25	35	28	21	32	28	33	28
	L-BROY	LP	Time	115.1	119.0	91.9	95.2	158.4	121.5	72.0	190.8	101.3	164.1	123.0
			Iter	35	36	28	29	48	37	22	58	31	50	37
		TP	Time	105.4	88.2	88.2	**84.6**	123.7	109.1	81.0	154.4	98.7	158.6	109.2
			Iter	30	25	25	24	35	31	23	44	28	45	31

Table 3.16: Excerpt of Table 3.15 showing only NPQN results which used the forward-backward HOOI sweep as preconditioner. (Bold entries indicate the best time for a given trial.)

This test is repeated 10 times for different $\mathcal{N}$, with the results recorded in Table 3.15. The fastest time for each trial is indicated in bold. For existing methods, the general trend is NPNCG using $\widehat{\beta}_{\text{HS}}$ gives the fastest convergence, followed by NPNGMRES, then NPNCG using $\widetilde{\beta}_{\text{HS}}$, and finally HOOI, being the slowest iteration considered by far. As in Table

3.9, we see that NPNCG using $\widehat{\beta}_{\text{HS}}$ exhibits slower convergence than the $\widetilde{\beta}_{\text{HS}}$ variant, in the worst cases more than doubling the iteration count and requiring at least twice as long to converge. This is one example where the benefit of basing nonlinear preconditioning on analogous linear problems is clear.

The fastest result for every trial corresponds to one of the NPQN iterations, with one of the L-BFGS variants typically being the best. Nearly all proposed methods outperform at least one of the NPNCG or NPNGMRES algorithms in each trial. From the cell coloring, we see that using the forward-backward sweep can result in significant reductions in time to solution in the majority of cases. This suggests that this preconditioning strategy may be the better choice for larger, more noisy problems.

For a clearer comparison we refer to Table 3.16 which contains the results for NPQN methods using forward-backward HOOI as preconditioner. Here the entries in bold indicate the best time for a given trial. In the majority of cases the L-BFGS results in faster convergence, and by looking at the mean time-to-solution, we see that in general the use of mixed preconditioning strategies results in approximately a 10% decrease in the time required for both window sizes. For L-Broyden we note that increasing window size from $m = 1$ to $m = 2$ results in noticeably faster methods, whereas the difference between mean values for different window sizes is very small for L-BFGS.

Part II

Parallel-In-Time Methods for Linear Advection

Chapter 4

Introduction to Multigrid Reduction in Time

Since modern high performance computing resources are massively parallel in scope, with systems potentially consisting of thousands of nodes and hundreds of thousands of cores, algorithms need to exhibit similar levels of concurrency in order to effectively make use of these resources. Domain decomposition in the spatial dimensions is an effective and commonly used way of introducing concurrency at the algorithmic level, but there is a point at which the additional communication costs outweigh the benefits of further spatial parallelism.

In order to increase the amount of concurrency possible, a wide variety of parallel-in-time methods have been developed with the aim of reducing/removing computational bottlenecks due to time integration. Variants include direct methods and iterative methods based on deferred corrections [41], domain decomposition [51], multigrid [58], multiple shooting [22], and waveform relaxation [101] approaches. These methods have had significant success in providing further speedup in the solution of parabolic equations, or equations with significant diffusivity, but have had markedly less success with hyperbolic or advection dominated problems [83].

For example, one of the most influential parallel-in-time methods is parareal [71], an iterative predictor-corrector method that combines the use of a coarse time integrator in serial and a fine time integrator in parallel. Parareal has been shown to have stability issues for the constant coefficient linear advection equation [50]. A number of variants and modifications have been proposed, and analysis has identified that issues arise when solutions lack regularity [28] due to phase errors in the coarse propagator [83]. A number of

variants have been proposed to stabilize and improve the convergence of parareal for such problems [24, 28, 49, 84], but with increased memory requirements or other restrictions. As a result, parallel-in-time methods that can be effectively applied to hyperbolic or advection dominated problems are still highly sought after.

4.1 Multigrid Reduction in Time

In this section we introduce the multigrid reduction-in-time (MGRIT) [42] approach for parallelizing time integration, which we will adapt to solve one-dimensional scalar hyperbolic PDEs over the following two chapters. A key strength of the MGRIT framework is its non-intrusive nature, which allows existing time-stepping routines to be easily used within the MGRIT implementation. MGRIT has been successfully implemented using time-stepping routines for linear [42] and nonlinear [44] parabolic PDEs in multiple dimensions, Navier-Stokes fluid dynamics problems [43], and power system models [66].

Consider a system of ordinary differential equations of the form

$$\mathbf{u}'(t) = \mathbf{f}(t, \mathbf{u}(t)), \quad \mathbf{u}(0) = \mathbf{u}_0, \quad t \in [0, T],$$

which can represent, for example, a system obtained from a method-of-lines discretization of a hyperbolic or parabolic PDE. This system is discretized on a uniform temporal mesh $t_i = i\Delta t$, $i = 0, 1 \ldots, N_t$, $\Delta t = T/N_t$, with $\mathbf{u}_i \approx \mathbf{u}(t_i)$. A general one-step iteration for computing the discrete solution is

$$\begin{aligned} \mathbf{u}_0 &= \mathbf{g}_0, \\ \mathbf{u}_i &= \mathbf{\Phi}_{i,\Delta x,\Delta t}(\mathbf{u_{i-1}}) + \mathbf{g}_i, \quad i = 1, 2, \ldots, N_t, \end{aligned}$$

where $\mathbf{\Phi}_{i,\Delta x,\Delta t}$ is a time-stepping function that depends on t_i and the spatial and temporal resolutions Δx and Δt. The right-hand side $\mathbf{g}_i$ contains solution-independent terms. We write this as the equivalent matrix equation (abusing notation in the nonlinear case)

$$\mathbf{Au} \equiv \begin{bmatrix} \mathbf{I} & & & \\ -\mathbf{\Phi}_{1,\Delta x,\Delta t} & \mathbf{I} & & \\ & \ddots & \ddots & \\ & & -\mathbf{\Phi}_{N_t,\Delta x,\Delta t} & \mathbf{I} \end{bmatrix} \begin{bmatrix} \mathbf{u}_0 \\ \mathbf{u}_1 \\ \vdots \\ \mathbf{u}_{N_t} \end{bmatrix} = \begin{bmatrix} \mathbf{g}_0 \\ \mathbf{g}_1 \\ \vdots \\ \mathbf{g}_{N_t} \end{bmatrix} \equiv \mathbf{g}. \tag{4.1}$$

Note that applying block forward substitution to this matrix corresponds to sequential time-stepping.

MGRIT is an iterative solver based on approximating an exact cyclic reduction strategy applied to system (4.1). For cyclic reduction we choose a constant temporal coarsening factor m and define a coarse time grid $T_{i_c} = i_c \Delta T$, $i_c = 0, 1, \ldots, N_T = N_t/m$, $\Delta T = m\Delta t$, as pictured in Figure 4.1 [42, original]. The $T_{i_c} = t_{mi_c}$ are coarse time points (*C-points*) and the remaining t_i are fine time points (*F-points*).

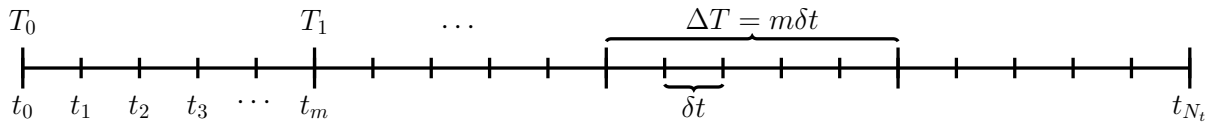

Figure 4.1: Fine and coarse temporal grids.

To illustrate the case $m = 2$, we first make the simplifying assumption that $\mathbf{\Phi}_{i,\Delta x,\Delta t} = \mathbf{\Phi}_{\Delta x,\Delta t}$ for $i = 1, \ldots, N_t$. Next, we partition the matrix $\mathbf{A}$ from (4.1) into C and F blocks:

$$\mathbf{A} = \begin{array}{c} \begin{array}{cccccc} \text{C} & & \text{F} & \text{C} & \cdots & \text{F} \end{array} \\ \begin{bmatrix} \mathbf{I}_{N_x} & & & & & \\ -\mathbf{\Phi}_{\Delta x,\Delta t} & \mathbf{I}_{N_x} & & & & \\ & -\mathbf{\Phi}_{\Delta x,\Delta t} & \mathbf{I}_{N_x} & & & \\ & & \ddots & \ddots & & \\ & & & -\mathbf{\Phi}_{\Delta x,\Delta t} & \mathbf{I}_{N_x} \end{bmatrix} \end{array} \begin{array}{c} \text{C} \\ \text{F} \\ \text{C} \\ \vdots \\ \text{F} \end{array},$$

that, when permuted to separate the F and C blocks (each set of blocks arranged in increasing order), results in the 2×2 block matrix

$$\begin{bmatrix} \mathbf{A}_{\text{ff}} & \mathbf{A}_{\text{fc}} \\ \mathbf{A}_{\text{cf}} & \mathbf{A}_{\text{cc}} \end{bmatrix}$$

where

$$\mathbf{A}_{\text{ff}} = \mathbf{A}_{\text{cc}} = \begin{bmatrix} \mathbf{I}_{N_x} & & \\ & \ddots & \\ & & \mathbf{I}_{N_x} \end{bmatrix}, \qquad \mathbf{A}_{\text{fc}} = \begin{bmatrix} -\mathbf{\Phi}_{\Delta x,\Delta t} & & \\ & \ddots & \\ & & -\mathbf{\Phi}_{\Delta x,\Delta t} \end{bmatrix},$$

$$\mathbf{A}_{\text{cf}} = \begin{bmatrix} \mathbf{0} & & & \\ -\mathbf{\Phi}_{\Delta x,\Delta t} & \mathbf{0} & & \\ & \ddots & \ddots & \\ & & -\mathbf{\Phi}_{\Delta x,\Delta t} & \mathbf{0} \end{bmatrix}.$$

The Schur complement for this partitioned matrix is

$$\mathbf{S}_\Delta = \mathbf{A}_{\rm cc} - \mathbf{A}_{\rm cf}\mathbf{A}_{\rm ff}^{-1}\mathbf{A}_{\rm fc} = \begin{bmatrix} \mathbf{I}_{N_x} & & & \\ -\mathbf{\Phi}^2_{\Delta x,\Delta t} & \mathbf{I}_{N_x} & & \\ & \ddots & \ddots & \\ & & -\mathbf{\Phi}^2_{\Delta x,\Delta t} & \mathbf{I}_{N_x} \end{bmatrix},$$

and the resulting Schur complement coarse-grid system $\mathbf{S}_\Delta \mathbf{u}_\Delta = \mathbf{g}_\Delta$ is:

$$\begin{bmatrix} \mathbf{I}_{N_x} & & & \\ -\mathbf{\Phi}^2_{\Delta x,\Delta t} & \mathbf{I}_{N_x} & & \\ & \ddots & \ddots & \\ & & -\mathbf{\Phi}^2_{\Delta x,\Delta t} & \mathbf{I}_{N_x} \end{bmatrix} \begin{bmatrix} \mathbf{u}_0 \\ \mathbf{u}_2 \\ \vdots \\ \mathbf{u}_{N_t} \end{bmatrix} = \begin{bmatrix} \mathbf{g}_0 \\ \mathbf{g}_2 + \mathbf{\Phi}_{\Delta x,\Delta t}\mathbf{g}_1 \\ \vdots \\ \mathbf{g}_{N_t} + \mathbf{\Phi}_{\Delta x,\Delta t}\mathbf{g}_{N_t-1} \end{bmatrix}. \tag{4.2}$$

The coarse-grid right hand side is computed using ideal restriction, $\mathbf{g}_\Delta = \mathbf{R}_{\mathbf{\Phi}}\mathbf{g}$, and the fine-grid solution $\mathbf{u}$ is recovered using ideal interpolation, $\mathbf{u} = \mathbf{P}_{\mathbf{\Phi}}\mathbf{u}_\Delta$ ("ideal" as they generate the Schur complement as the Petrov-Galerkin coarse-grid operator), where

$$\mathbf{R}_{\mathbf{\Phi}} = \begin{bmatrix} \mathbf{I}_{N_x} & & & & \\ & \mathbf{\Phi} & \mathbf{I}_{N_x} & & \\ & & \ddots & & \\ & & & \mathbf{\Phi} & \mathbf{I}_{N_x} \end{bmatrix} \quad \text{and} \quad \mathbf{P}_{\mathbf{\Phi}} = \begin{bmatrix} \mathbf{I}_{N_x} & \mathbf{\Phi}^\intercal & & & \\ & & \ddots & & \\ & & & \mathbf{I}_{N_x} & \mathbf{\Phi}^\intercal \\ & & & & \mathbf{I}_{N_x} \end{bmatrix}^\intercal.$$

Solving (4.2) will return the exact solution in one iteration, and it is just as expensive to solve as the original fine-grid problem due to the use of $\mathbf{\Phi}^2_{\Delta x,\Delta t}$. Instead, we replace the coarse-grid Schur complement matrix $\mathbf{S}_\Delta$ with the rediscretization of the underlying equation on the coarse time grid, $\mathbf{A}_\Delta$:

$$\begin{bmatrix} \mathbf{I}_{N_x} & & & \\ -\mathbf{\Phi}_{\Delta x,2\Delta t} & \mathbf{I}_{N_x} & & \\ & \ddots & \ddots & \\ & & -\mathbf{\Phi}_{\Delta x,2\Delta t} & \mathbf{I}_{N_x} \end{bmatrix} \begin{bmatrix} \mathbf{u}_{\Delta,0} \\ \mathbf{u}_{\Delta,1} \\ \vdots \\ \mathbf{u}_{\Delta,N_t/2} \end{bmatrix} = \begin{bmatrix} \mathbf{g}_{\Delta,0} \\ \mathbf{g}_{\Delta,1} \\ \vdots \\ \mathbf{g}_{\Delta,N_t/2} \end{bmatrix}. \tag{4.3}$$

As a result of the rediscretization, the coarse-grid solution will only approximate the fine-grid solution, and hence will not give the exact solution in one iteration. This is the coarse-grid problem corresponding to fine-grid problem (4.1), that we may now solve via a multigrid approach, i.e., MGRIT. In two-level MGRIT, this coarse-grid problem is solved exactly, whereas multilevel MGRIT applies this process recursively.

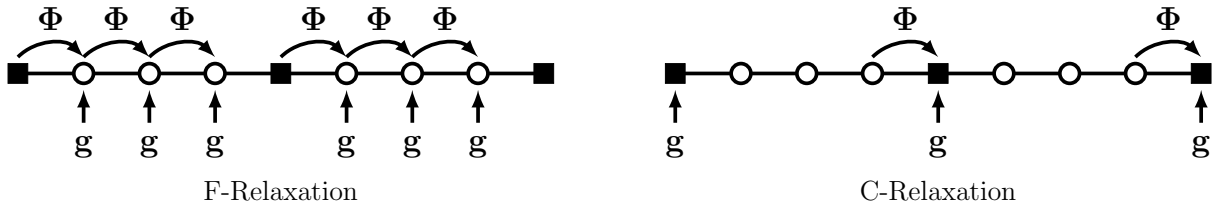

Figure 4.2: Illustration of F- and C-relaxation on a 9-point temporal grid with coarsening factor 4.

Two fundamental types of temporal relaxation are used in MGRIT: *F-relaxation* and *C-relaxation*. F-relaxation updates the F-point values $\mathbf{u}_i$ in the interval (T_{i_c}, T_{i_c+1}) by starting with the C-point value $\mathbf{u}_{mi_c}$ and then applying each $\mathbf{\Phi}_{i,\delta t}$ in sequence. Since each interval is updated independently, the intervals can be processed in parallel. Similarly, C-relaxation updates C-point values $\mathbf{u}_{mi_c}$ based on current F-point values $\mathbf{u}_{mi_c-1}$, which can also be done in parallel. These relaxation strategies are illustrated in Figure 4.2 [42, original]. In particular, note that two-level MGRIT with F-relaxation is equivalent to parareal [42, 50]. These sweeps can also be combined into FCF-relaxation: F-relaxation followed by C-relaxation followed by a second F-relaxation. We note here that ideal restriction and prolongation are equivalent to particular combinations of injection and F-relaxation: ideal restriction is injection $\mathbf{R_I}$ preceded by an F-relaxation, and ideal prolongation is injection $\mathbf{P_I}$ followed by an F-relaxation [42], where

$$\mathbf{R_I} = \begin{bmatrix} \mathbf{I}_{N_x} & & & & \\ & \mathbf{0} & \mathbf{I}_{N_x} & & \\ & & & \ddots & \\ & & & & \mathbf{0} \quad \mathbf{I}_{N_x} \end{bmatrix} = \mathbf{P}_\mathbf{I}^\intercal.$$

MGRIT uses the Full Approximation Storage (FAS) framework [15] for solving both linear and nonlinear problems, which involves computing the coarse-grid correction by solving a coarsened version of the residual equation $\mathcal{A}(\mathbf{u}+\mathbf{e}) - \mathcal{A}(\mathbf{u}) = \mathbf{r}$, where $\mathcal{A}$ is the (potentially nonlinear) operator to be inverted. The two-grid MGRIT FAS algorithm first appeared in [43]. We include the variant from [44] instead (see Algorithm 15), as it accounts for the possibility of spatial coarsening (with restriction and prolongation operators $\mathbf{R}_\text{s}$ and $\mathbf{P}_\text{s}$), described in the next section. In the case of $\mathcal{A}$ being a matrix $\mathbf{A}$ this reduces to the standard multigrid algorithm. This is easily extended to more than two levels by replacing the exact coarse-grid solve on line 5 with a recursive call.

Algorithm 15 FAS-MGRIT

procedure FAS-MGRIT($\mathcal{A}, \mathbf{u}, \mathbf{g}$)
  Apply F- or FCF-relaxation to $\mathcal{A}(\mathbf{u}) = \mathbf{g}$
  Inject the fine-grid approximation and residual to the coarse grid
    $\mathbf{u}_\Delta = \mathbf{R}_\mathrm{I}(\mathbf{u}), \quad \mathbf{r}_\Delta = \mathbf{R}_\mathrm{I}(\mathbf{g} - \mathcal{A}(\mathbf{u}))$
  If using spatial coarsening then:
    $\mathbf{u}_\Delta = \mathbf{R}_\mathrm{s}(\mathbf{u}_\Delta), \quad \mathbf{r}_\Delta = \mathbf{R}_\mathrm{s}(\mathbf{r}_\Delta)$
  Solve $\mathcal{A}_\Delta(\mathbf{v}_\Delta) = \mathcal{A}_\Delta(\mathbf{u}_\Delta) + \mathbf{r}_\Delta$
  Compute the coarse-grid error approximation: $\mathbf{e}_\Delta = \mathbf{v}_\Delta - \mathbf{u}_\Delta$
  If using spatial coarsening then: $\mathbf{e}_\Delta = \mathbf{P}_\mathrm{s}(\mathbf{e}_\Delta)$
  Correct using ideal interpolation: $\mathbf{u} = \mathbf{u} + \mathbf{P}_{\mathbf{\Phi}}(\mathbf{e}_\Delta)$
end procedure

4.2 MGRIT with Spatial Coarsening

As a multigrid method, MGRIT primarily involves temporal coarsening, but spatial coarsening may be necessary for explicit time integration to ensure that stability conditions are satisfied on all levels of the grid hierarchy. Spatial coarsening may also be used with implicit time integration to produce smaller coarse-grid problems and hence cheaper multigrid cycles. There are many choices of possible spatial restriction and prolongation operators that have been developed for multigrid [98]. We use a standard choice of full weighting restriction and linear interpolation operators, which are defined by

$$\begin{aligned} u_{2h}(x) &= 0.25[u_h(x-h) + 2u_h(x) + u_h(x+h)], \\ u_h(x) &= \tfrac{1}{2}[u_{2h}(x-h) + u_{2h}(x+h)], \end{aligned}$$

and have matrix representations

$$\mathbf{R}_\mathrm{s} = \frac{1}{4}\begin{bmatrix} 2 & 1 & & & & & & 1 \\ & 1 & 2 & 1 & & & & \\ & & & & \ddots & & & \\ & & & & & 1 & 2 & 1 \end{bmatrix} = \frac{1}{2}\mathbf{P}_\mathrm{s}^\intercal,$$

which correspond to the case of periodic boundary conditions being enforced on the spatial interval of interest.

For temporal coarsening, the coarse-grid time-stepper $\mathbf{\Phi}_{i_c,\Delta x,m\Delta t}$ was obtained by multiplying the time step Δt by the temporal coarsening factor m. For spatial coarsening

we handle the explicit and implicit cases in different ways. For explicit time-stepping we rediscretize on the fully coarsened grid, obtaining $\mathbf{\Phi}_{i_c,2\Delta x,m\Delta t}$, a $N_x/2 \times N_x/2$ matrix. For implicit time-stepping we instead use a Galerkin definition involving $\mathbf{\Phi}_{i_c,\Delta x,m\Delta t}$. Galerkin-type discretizations lead to optimal results in the A-norm for SPD problems [16], and they have also been used for nonsymmetric matrices, for example, in [94]. We use a Galerkin approach in this paper for implicit timestepping, because we find it leads to better results than rediscretization.

To describe this method, we first note that the MGRIT matrix equation described in (4.1) typically corresponds to cases where $\mathbf{\Phi}$ is a sparse matrix. If $\mathbf{\Phi}$ is the *inverse* of a sparse matrix, as is the case for many implicit time-stepping methods, we may instead write $-\mathbf{I}$ on the first block subdiagonal and $\mathbf{\Phi}^{-1}_{i,\Delta x,\Delta t}$ on the block main diagonal:

$$\widehat{\mathbf{A}}\mathbf{u} \equiv \begin{bmatrix} \mathbf{\Phi}^{-1}_{0,\Delta x,\Delta t} & & & \\ -\mathbf{I}_{N_x} & \mathbf{\Phi}^{-1}_{1,\Delta x,\Delta t} & & \\ & \ddots & \ddots & \\ & & -\mathbf{I}_{N_x} & \mathbf{\Phi}^{-1}_{N_t,\Delta x,\Delta t} \end{bmatrix} \begin{bmatrix} \mathbf{u}_0 \\ \mathbf{u}_1 \\ \vdots \\ \mathbf{u}_{N_t} \end{bmatrix} = \begin{bmatrix} \mathbf{\Phi}^{-1}_{0,\Delta x,\Delta t}\mathbf{g}_0 \\ \mathbf{\Phi}^{-1}_{1,\Delta x,\Delta t}\mathbf{g}_1 \\ \vdots \\ \mathbf{\Phi}^{-1}_{N_t,\Delta x,\Delta t}\mathbf{g}_{N_t} \end{bmatrix} = \widehat{\mathbf{g}}. \tag{4.4}$$

In this case, applying $\mathbf{\Phi}_{i,\Delta x,\Delta t}$ is a linear solve and $\mathbf{\Phi}^{-1}_{i,\Delta x,\Delta t}$ is the matrix to be inverted.

Working with (4.4) on the temporally coarsened grid (assuming $m = 2$ for simplicity) and defining spatial restriction $\mathbf{R}_{s,i}$ and prolongation $\mathbf{P}_{s,i}$ to correspond to time t_i, we multiply each block row by $\mathbf{R}_{s,i}$ and replace each $\mathbf{u}_i$ with $\mathbf{P}_{s,i}\mathbf{u}_{s,i}$ to obtain

$$\begin{bmatrix} \mathbf{R}_{s,0}\mathbf{\Phi}^{-1}_{0,\Delta x,2\Delta t}\mathbf{P}_{s,0} & & & \\ -\mathbf{R}_{s,1}\mathbf{P}_{s,0} & \mathbf{R}_{s,1}\mathbf{\Phi}^{-1}_{1,\Delta x,2\Delta t}\mathbf{P}_{s,1} & & \\ & \ddots & & \\ & -\mathbf{R}_{s,N_t/2}\mathbf{P}_{s,N_t/2-1} & \mathbf{R}_{s,N_t/2}\mathbf{\Phi}^{-1}_{N_t/2,\Delta x,2\Delta t}\mathbf{P}_{s,N_t/2} \end{bmatrix} \begin{bmatrix} \mathbf{u}_{s,0} \\ \mathbf{u}_{s,1} \\ \vdots \\ \mathbf{u}_{s,N_t/2} \end{bmatrix} = \begin{bmatrix} \mathbf{R}_{s,0}\widehat{\mathbf{g}}_0 \\ \mathbf{R}_{s,1}\widehat{\mathbf{g}}_1 \\ \vdots \\ \mathbf{R}_{s,N_t/2}\widehat{\mathbf{g}}_{N_t/2} \end{bmatrix}.$$

Each block row of this matrix equation (except the first) can be written as

$$-\mathbf{R}_{s,i}\mathbf{P}_{s,i-1}\mathbf{u}_{s,i-1} + \mathbf{R}_{s,i}\Phi^{-1}_{i,\Delta x,m\Delta t}\mathbf{P}_{s,i}\mathbf{u}_{s,i} = \mathbf{R}_{s,i}\widehat{\mathbf{g}}_i,$$

thus we compute

$$\mathbf{u}_{s,i} = \left(\mathbf{R}_{s,i}\Phi^{-1}_{i,\Delta x,m\Delta t}\mathbf{P}_{s,i}\right)^{-1}\left[\mathbf{R}_{s,i}\mathbf{P}_{s,i-1}\mathbf{u}_{s,i-1} + \mathbf{R}_{s,i}\widehat{\mathbf{g}}_i\right]. \tag{4.5}$$

For linear time-steppers the matrix $\mathbf{R}_{s,i}\Phi^{-1}_{i,\Delta x,\Delta T}\mathbf{P}_{s,i}$ is computed as the product of the three sparse matrices $\mathbf{R}_{s,i}$, $\Phi^{-1}_{i,\Delta x,\Delta T}$, and $\mathbf{P}_{s,i}$, which is then factored and stored for future use. In the nonlinear case we first prolong the coarse-grid vector to the previous intermediate grid

(coarse-in-time, fine-in-space), evaluate and compute the Jacobian for $\Phi^{-1}_{i,\Delta x,\Delta T}(\mathbf{P}_{s,i}\mathbf{u}_{s,i}) - \mathbf{P}_{s,i-1}\mathbf{u}_{s,i-1} - \mathbf{g}_i = \mathbf{0}$, then restrict both and solve the resulting coarse-grid linear system. Compared to rediscretization we find this definition can result in cheaper overall algorithms, both in terms of iterations required and overall time to solution.

We do not consider defining an explicit time-stepping coarse-grid operator in this way for two reasons. First, it would result in a stricter stability condition when compared to the rediscretized coarse-grid operator. Second, compared to the implicit case, where this definition adds a matrix-vector product to the computational cost of the iteration, in the explicit case the Galerkin definition adds a linear system solve (computing the product as above for the explicit formulation results in a matrix $\mathbf{R}_{s,i}\mathbf{P}_{s,i}$ multiplying $\mathbf{u}_{s,i}$ that will need to be inverted), which is not as parallelizable as the initial matrix-vector product required, becoming a significant bottleneck as spatial parallelism is added.

4.3 Problem of Interest

In this thesis we are primarily interested in the conservative hyperbolic PDE

$$\frac{\partial u}{\partial t} + \frac{\partial (f(u,x,t))}{\partial x} = 0, \tag{4.6}$$

In particular, we consider the variable coefficient linear advection equation, $f(u,x,t) = a(x,t)u$, and the inviscid Burgers equation, $f(u,x,t) = \frac{1}{2}u^2$.

We consider the numerical solution of (4.6) on a finite spatial interval $[a,b]$ and assume periodic boundary conditions in all that follows. We use the vertex-centered approach to construct spatial grids [59, § III.4]: a grid is defined by points $\{x_j\}_{j=0}^{N-1}$ and has cells $\Omega_j = [x_{j-1/2}, x_{j+1/2}]$, where $x_{j\pm 1/2} = \frac{1}{2}(x_j + x_{j\pm 1})$; i.e., the vertices (boundaries/cell interfaces) are *centered* between x_j and $x_{j\pm 1}$. When performing spatial coarsening, the vertex-centered approach allows us to use a subset of $\{x_j\}_{j=0}^{N-1}$ to describe the grid on each level: no new reference points are required. Dividing $[a,b]$ into N_x cells of equal width, the fine-grid points $\{x_j\}$ are

$$x_j = a + \frac{1}{N_x}(b-a)\left(\tfrac{1}{2} + j\right), j = 0, 1, \ldots, N_x - 1,$$

Defining $\delta x_j = \frac{1}{2}(x_{j+1} - x_{j-1})$, (4.6) is semi-discretized in space as [59]

$$\frac{\partial u_j}{\partial t} + \frac{f^*_{j+1/2}(t) - f^*_{j-1/2}(t)}{\delta x_j} = 0, \tag{4.7}$$

where $f^*_{j+1/2}(t)$ is chosen as the local Lax-Friedrichs flux approximation:

$$f^*_{j+1/2}(t) = \frac{f(u_{j+1}(t), x_{j+1/2}, t) + f(u_j(t), x_{j+1/2}, t)}{2} - \frac{1}{2}\frac{|\partial_u f(u_{j+1}(t), x_{j+1/2}, t)| + |\partial_u f(u_j(t), x_{j+1/2}, t)|}{2}(u_{j+1}(t) - u_j(t)). \tag{4.8}$$

For variable coefficient linear advection, this reduces to

$$f^*_{j+1/2}(t) = \tfrac{1}{2}\left[a(x_{j+1/2}, t)\left(u_{j+1}(t) + u_j(t)\right) - |a(x_{j+1/2}, t)|\left(u_{j+1}(t) - u_j(t)\right)\right], \tag{4.9}$$

and for Burgers' equation

$$f^*_{j+1/2}(t) = \tfrac{1}{4}\left[(u_{j+1}(t))^2 + (u_j(t))^2 - \left(|u_{j+1}(t)| + |u_j(t)|\right)\left(u_{j+1}(t) - u_j(t)\right)\right]. \tag{4.10}$$

This conservative discretization was chosen to make our approach applicable to nonlinear conservation laws $\partial_t u + \partial_x f(u) = 0$, where (4.8) guarantees correct shock speeds. In this paper we consider the forward and backward Euler time discretizations, which result in the fully discrete equations (space index j, time index i)

$$\left(a^i_{j-1/2} + |a^i_{j-1/2}|\right)\frac{\delta t}{2\delta x_j}u^i_{j-1} - \left(a^i_{j+1/2} - |a^i_{j+1/2}|\right)\frac{\delta t}{2\delta x_j}u^i_{j+1} + \left[1 - \left(a^i_{j+1/2} - a^i_{j-1/2} + |a^i_{j+1/2}| + |a^i_{j-1/2}|\right)\frac{\delta t}{2\delta x_j}\right]u^i_j = u^{i+1}_j \tag{4.11}$$

and

$$-\left(a^{i+1}_{j-1/2} + \left|a^{i+1}_{j-1/2}\right|\right)\frac{\delta t}{2\delta x_j}u^{i+1}_{j-1} + \left(a^{i+1}_{j+1/2} - \left|a^{i+1}_{j+1/2}\right|\right)\frac{\delta t}{2\delta x_j}u^{i+1}_{j+1} + \left[1 + \left(a^{i+1}_{j+1/2} - a^{i+1}_{j-1/2} + \left|a^{i+1}_{j+1/2}\right| + \left|a^{i+1}_{j-1/2}\right|\right)\frac{\delta t}{2\delta x_j}\right]u^{i+1}_j = u^i_j \tag{4.12}$$

for linear advection, and the fully discrete equations

$$\left(u^i_{j-1} + |u^i_j| + |u^i_{j-1}|\right)\frac{\delta t}{4\delta x_j}u^i_{j-1} - \left(u^i_{j+1} - |u^i_{j+1}| - |u^i_j|\right)\frac{\delta t}{4\delta x_j}u^i_{j+1} + \left[1 - \left(|u^i_{j+1}| + 2|u^i_j| + |u^i_{j-1}|\right)\frac{\delta t}{4\delta x_j}\right]u^i_j = u^{i+1}_j \tag{4.13}$$

and

$$-\left(u^{i+1}_{j-1} + |u^{i+1}_j| + |u^{i+1}_{j-1}|\right)\frac{\delta t}{4\delta x_j}u^{i+1}_{j-1} + \left(u^{i+1}_{j+1} - |u^{i+1}_{j+1}| - |u^{i+1}_j|\right)\frac{\delta t}{4\delta x_j}u^{i+1}_{j+1} + \left[1 + \left(|u^{i+1}_{j+1}| + 2|u^{i+1}_j| + |u^{i+1}_{j-1}|\right)\frac{\delta t}{4\delta x_j}\right]u^{i+1}_j = u^i_j \tag{4.14}$$

for Burgers' equation.

Chapter 5

MGRIT For Hyperbolic Problems

In this chapter we use MGRIT to solve the 1D constant coefficient linear advection equation both with and without spatial coarsening. In the case of implicit time-stepping we compare these results to those for the 1D diffusion equation over the same space-time domain to illustrate the increase in difficulty when considering non-diffusive problems. We apply a variant of Fourier analysis called semi-algebraic mode analysis (SAMA) [46] to further elucidate the differences between implicit MGRIT applied to advection and diffusion and the effect of introducing spatial coarsening to these methods. We then introduce the idea of combining MGRIT with waveform relaxation multigrid (WRMG) [95] and show how this can offer improved convergence at the cost of being a more intrusive approach.

For the sake of brevity in this and the following chapter, we define "Explicit (Implicit) MGRIT" to mean "MGRIT using an explicit (implicit) time-stepping function $\mathbf{\Phi}_{\Delta x,\Delta t}$". Furthermore, for the analysis we use the advection-diffusion equation

$$\frac{\partial u}{\partial t} + a\frac{\partial u}{\partial x} = \varepsilon\frac{\partial^2 u}{\partial x^2} \tag{5.1}$$

so that both pure advection and pure diffusion cases can be considered by setting ε or a equal to zero, respectively.

5.1 The Effect of Spatial Coarsening

To begin this section we wish to illustrate how the choices between (i) multigrid V-cycles or F-cycles, (ii) temporal F-relaxation or FCF-relaxation, and (iii) spatial coarsening or

no spatial coarsening can affect the performance of MGRIT. To do so we consider the simplest hyperbolic problem possible: the linear advection equation with unit wave speed ((5.1) with $a = 1$, $\varepsilon = 0$): $u_t + u_x = 0$. We solve this problem for $(x, t) \in [-2, 2] \times [0, 4]$ with periodic boundary conditions and initial condition (5.2), taking $u_A = 2$ and $u_B = 1$.

$$u(x) = \begin{cases} u_A & x \in [-2, -1], \\ u_B - x(u_A - u_B) & x \in (-1, 0], \\ u_B & x \in (0, 1], \\ u_B + (x - 1)(u_A - u_B) & x \in (1, 2]. \end{cases} \tag{5.2}$$

We let MGRIT iterate until the residual norm decreases below 10^{-10}. For simplicity we use the coarse-grid operators obtained by rediscretization rather than the Galerkin definition. For implicit MGRIT we set $N_x = N_t$ and for explicit MGRIT we set $2N_x = N_t$, which ensures the Courant-Friedrichs-Lewy (CFL) stability condition [69]

$$a\frac{\Delta t}{\Delta x} \leq 1$$

is satisfied on all levels of the grid hierarchy when uniform spatial coarsening is used.

$N_x \times N_t$			$2^7 \times 2^8$	$2^8 \times 2^9$	$2^9 \times 2^{10}$	$2^{10} \times 2^{11}$	$2^{11} \times 2^{12}$
F	No SC	2-level	100*	100*	100*	100*	100*
		V-cycle	100*	100*	100*	100*	100*
		F-cycle	100*	100*	100*	100*	100*
	SC-2	2-level	100*	100*	100*	100*	100*
		V-cycle	100*	100*	100*	100*	100*
		F-cycle	100*	100*	100*	100*	100*
FCF	No SC	2-level	51	96	100*	100*	100*
		V-cycle	100*	100*	100*	100*	100*
		F-cycle	100*	100*	100*	100*	100*
	SC-2	2-level	36	37	38	39	40
		V-cycle	49	65	89	100*	100*
		F-cycle	39	45	52	61	73

Table 5.1: Results for explicit MGRIT applied to the linear advection equation. No SC: no spatial coarsening; SC-2: factor-two spatial coarsening. Asterisks denote tests which failed to converge within 100 iterations.

Tables 5.1 and 5.2 record the iteration count for each test completed using explicit and implicit MGRIT, respectively. From these results we can draw some clear conclusions.

$N_x \times N_t$			$2^7 \times 2^7$	$2^8 \times 2^8$	$2^9 \times 2^9$	$2^{10} \times 2^{10}$	$2^{11} \times 2^{11}$
F	No SC	2-level	18	19	20	21	21
		V-cycle	30	45	70	100*	100*
		F-cycle	20	24	29	37	49
	SC-2	2-level	100*	100*	100*	100*	100*
		V-cycle	100*	100*	100*	100*	100*
		F-cycle	100*	100*	100*	100*	100*
FCF	No SC	2-level	15	17	18	19	19
		V-cycle	17	23	31	45	65
		F-cycle	15	18	21	25	30
	SC-2	2-level	21	25	28	29	30
		V-cycle	23	32	45	64	95
		F-cycle	21	27	33	40	50

Table 5.2: Results for implicit MGRIT applied to the linear advection equation. No SC: no spatial coarsening; SC-2: factor-two spatial coarsening. Asterisks denote tests which failed to converge within 100 iterations.

We see that FCF-relaxation is necessary for MGRIT with spatial coarsening to converge, as error modes in the null-space of the spatial restriction operator are damped solely by FCF-relaxation [44]. Furthermore, FCF-relaxation produces better scaling than F-relaxation for MGRIT without spatial coarsening, which has previously been noted in [42]. As expected, spatial coarsening is necessary for explicit MGRIT to converge because of stability requirements. Finally, while neither V-cycles nor F-cycles exhibit the desired multigrid optimality, the number of MGRIT iterations required increasing with problem size, we see that F-cycles scale better than V-cycles. Based on these results we will restrict our consideration to FCF-relaxation and F-cycles in subsequent tests for the advection equation, as these consistently produce the best results out of all the cases considered.

As an example of the type of results we would ideally like to achieve, Table 5.3 records the iterations and time required for convergence when implicit MGRIT is applied to the 1D diffusion equation ((5.1) with $a = 0$, $\varepsilon = 1$), $u_t = u_{xx}$, for the same space-time domain and initial condition as the advection equation. Explicit MGRIT tests are omitted due the more strict stability condition for the diffusion equation in this case. From these results it is clear that MGRIT exhibits the desired multigrid optimality for FCF-relaxation, and spatial coarsening does not result in an increased iteration count. Comparing these results to those of Table 5.2 leads to the obvious conclusion that the hyperbolic problem is significantly more difficult, in most cases requiring at least twice as many iterations for convergence.

$N_x \times N_t$			$2^7 \times 2^7$	$2^8 \times 2^8$	$2^9 \times 2^9$	$2^{10} \times 2^{10}$	$2^{11} \times 2^{11}$
F	No SC	2-level	10	11	11	11	11
		V-cycle	18	21	25	29	32
		F-cycle	10	11	11	11	11
	SC-2	2-level	100*	100*	100*	100*	100*
		V-cycle	100*	100*	100*	100*	100*
		F-cycle	100*	100*	100*	100*	100*
FCF	No SC	2-level	7	7	8	8	8
		V-cycle	8	9	9	10	10
		F-cycle	7	7	8	8	8
	SC-2	2-level	7	7	8	8	8
		V-cycle	8	9	9	10	10
		F-cycle	7	7	8	8	8

Table 5.3: Results for implicit MGRIT applied to the diffusion equation. No SC: no spatial coarsening; SC-2: factor-two spatial coarsening. Asterisks denote tests which failed to converge within 100 iterations.

5.2 Semi-Algebraic Mode Analysis

In this section we use the SAMA method described in [46] to analyze the implicit MGRIT iteration matrix and better understand how convergence is affected by parameters Δx and Δt, as well as the inclusion of spatial coarsening. It has been observed that local Fourier analysis (LFA) may fail to accurately predict convergence behaviour for methods applied to parabolic or hyperbolic problems [47]. SAMA was designed to take into account the structure of the matrices being analyzed, using LFA to handle circulant or Toeplitz blocks/matrices and algebraic computation to handle those with different structure [46].

We can obtain a worst-case convergence estimate for two-level MGRIT by computing the 2-norm of the iteration matrix. The iteration matrix for MGRIT without spatial coarsening is

$$\mathbf{T}_{\text{MGRIT}}^{(1,2)} = \left(\mathbf{I} - \mathbf{P}\mathbf{A}_{\Delta}^{-1}\mathbf{R}\mathbf{A}\right)\mathbf{S}_t^{\text{F}}\mathbf{S}_t^{\text{C}}\mathbf{S}_t^{\text{F}},$$

and the iteration matrix for MGRIT with spatial coarsening is

$$\mathbf{T}_{\text{MGRIT,sc}}^{(1,2)} = \left(\mathbf{I} - \mathbf{P}\mathbf{P}_s\mathbf{A}_s^{-1}\mathbf{R}_s\mathbf{R}\mathbf{A}\right)\mathbf{S}_t^{\text{F}}\mathbf{S}_t^{\text{C}}\mathbf{S}_t^{\text{F}}.$$

The SAMA methodology uses the matrix $\mathbf{\Psi}$ of N_x discretized spatial Fourier modes,

$$\psi(x, \theta_k) = e^{i\theta_k x/\Delta x}, \quad \theta_k = \frac{2\pi k}{N_x}, \quad k = -\frac{N_x}{2} + 1, \ldots, \frac{N_x}{2},$$

to diagonalize the circulant blocks of the iteration matrix by computing $\mathcal{F}^{-1}\mathbf{T}\mathcal{F}$, where $\mathcal{F} = \mathbf{I}_{N_t} \otimes \mathbf{\Psi}$ and we omit subscripts and superscripts for clarity. The resulting matrix is then permuted to obtain a block diagonal structure, with different blocks corresponding to the error reduction in different Fourier modes: $\widehat{\mathbf{T}} = \mathcal{P}^{-1}\mathcal{F}^{-1}\mathbf{T}\mathcal{F}\mathcal{P}$. In order to transform and block diagonalize the overall iteration matrix we apply these transformations to each individual matrix in the product that defines it.

The matrices $\mathbf{A}_\Delta$ and $\mathbf{A}_s$ are the analogues of $\mathbf{A}$ on semi-coarsened (time only) and fully coarsened (both time and space) grids: $\mathbf{A}_\Delta$ is a $N_t/m \times N_t/m$ block matrix with blocks of size $N_x \times N_x$, the subdiagonal blocks being $-\mathbf{\Phi}_{\Delta x, m\Delta t}$; and $\mathbf{A}_s$ is a $N_t/m \times N_t/m$ block matrix with blocks of size $N_x/2 \times N_x/2$, the subdiagonal blocks being $-\mathbf{\Phi}_{2\Delta x, m\Delta t}$. The eigenvalues of $\mathbf{\Phi}_{\Delta t, \Delta x}$ corresponding to (5.1) are

$$\lambda_k = \left(1 + \frac{a\Delta t}{\Delta x}[1 - e^{-i\theta_k}] + \frac{2\varepsilon\Delta t}{\Delta x^2}[1 - \cos\theta_k]\right)^{-1}.$$

The eigenvalues of $\mathbf{\Phi}_{\Delta x, m\Delta t}$ are obtained by replacing Δt by $m\Delta t$ in the above equation; the eigenvalues of $\mathbf{\Phi}_{2\Delta x, m\Delta t}$ are obtained by further replacing Δx by $2\Delta x$ and θ_k by $2\theta_k$. The transformed matrix $\mathcal{F}^{-1}\mathbf{A}\mathcal{F}$ is permuted to have $N_x \times N_x$ blocks, each of size $N_t \times N_t$, with diagonal blocks corresponding to the evolution of one spatial Fourier mode over time:

$$\mathcal{P}^{-1}\mathcal{F}^{-1}\mathbf{A}\mathcal{F}\mathcal{P} = \begin{bmatrix} \widehat{\mathbf{A}}_1 & & \\ & \ddots & \\ & & \widehat{\mathbf{A}}_{N_x} \end{bmatrix}, \quad \text{where} \quad \widehat{\mathbf{A}}_k = \begin{bmatrix} 1 & & & \\ -\lambda_k & 1 & & \\ & \ddots & \ddots & \\ & & -\lambda_k & 1 \end{bmatrix}.$$

Analogous expressions can be found for $\widehat{\mathbf{A}}_{\Delta,k}$ and $\widehat{\mathbf{A}}_{s,k}$.

$\mathbf{S}_t^{\mathrm{F}}$ and $\mathbf{S}_t^{\mathrm{C}}$ are the error propagation matrices for temporal F and C-relaxations:

$$\mathbf{S}_t^{\mathrm{F}} = \mathbf{I}_{N_t/m} \otimes \begin{bmatrix} \mathbf{I} & & & \\ & \mathbf{\Phi} & & \\ & & \ddots & \\ & & & \mathbf{\Phi}^{m-1} \end{bmatrix}, \quad \mathbf{S}_t^{\mathrm{C}} = \mathbf{K}_{N_t/m} \otimes \mathbf{E}_m^{1,m} \otimes \mathbf{\Phi} + \mathbf{I}_{N_t/m} \otimes \begin{bmatrix} \mathbf{0} & & & \\ & \mathbf{I} & & \\ & & \ddots & \\ & & & \mathbf{I} \end{bmatrix},$$

where $\mathbf{K}_{N_t/m} \in \mathbb{R}^{(N_t/m)\times(N_t/m)}$ and $\mathbf{E}_m^{1,m} \in \mathbb{R}^{m\times m}$ are defined by

$$\mathbf{K}_{N_t/m}(i,j) = \begin{cases} 1 & \text{if } i-j=1, \\ 0 & \text{otherwise,} \end{cases} \qquad \mathbf{E}_m^{1,m}(i,j) = \begin{cases} 1 & \text{if } i=1, j=m, \\ 0 & \text{otherwise.} \end{cases}$$

We note that an alternate (but computationally equivalent) description of these matrices is provided in [36, 46], which corresponds to reordering the fine-grid operator $\mathbf{A}$ to first consider all F points, then all C points. These are $N_t \times N_t$ block matrices with blocks of size $N_x \times N_x$ that are transformed and then permuted into $N_x \times N_x$ block matrices with blocks of size $N_t \times N_t$, the resulting $N_t \times N_t$ blocks themselves being block bidiagonal matrices with blocks of size $m \times m$:

$$\mathcal{P}^{-1}\mathcal{F}^{-1}\mathbf{S}_t^{\mathrm{F}}\mathcal{F}\mathcal{P} = \begin{bmatrix} \widehat{\mathbf{S}}_{t,1}^{\mathrm{F}} & & \\ & \ddots & \\ & & \widehat{\mathbf{S}}_{t,N_x}^{\mathrm{F}} \end{bmatrix} \quad \text{and} \quad \mathcal{P}^{-1}\mathcal{F}^{-1}\mathbf{S}_t^{\mathrm{C}}\mathcal{F}\mathcal{P} = \begin{bmatrix} \widehat{\mathbf{S}}_{t,1}^{\mathrm{C}} & & \\ & \ddots & \\ & & \widehat{\mathbf{S}}_{t,N_x}^{\mathrm{C}} \end{bmatrix},$$

where

$$\widehat{\mathbf{S}}_{t,k}^{\mathrm{F}} = \begin{bmatrix} \mathbf{Z}_k^{\mathrm{F}} & & & \\ & \mathbf{Z}_k^{\mathrm{F}} & & \\ & & \ddots & \\ & & & \mathbf{Z}_k^{\mathrm{F}} \end{bmatrix} \quad \text{and} \quad \widehat{\mathbf{S}}_{t,k}^{\mathrm{C}} = \begin{bmatrix} \mathbf{Z}_k^{\mathrm{C}} & & & \\ \lambda_k \mathbf{e}_1 \mathbf{e}_m^{\intercal} & \mathbf{Z}_k^{\mathrm{C}} & & \\ & \ddots & \ddots & \\ & & \lambda_k \mathbf{e}_1 \mathbf{e}_m^{\intercal} & \mathbf{Z}_k^{\mathrm{C}} \end{bmatrix},$$

with

$$\mathbf{Z}_k^{\mathrm{F}} = \begin{bmatrix} 1 & & & \\ \lambda_k & 0 & & \\ \lambda_k^2 & & 0 & \\ \vdots & & & \ddots \\ \lambda_k^{m-1} & & & 0 \end{bmatrix}, \quad \mathbf{Z}_k^{\mathrm{C}} = \begin{bmatrix} 0 & & & \\ & 1 & & \\ & & 1 & \\ & & & \ddots \\ & & & & 1 \end{bmatrix}, \quad \mathbf{e}_1 = \begin{bmatrix} 1 \\ 0 \\ \vdots \\ 0 \\ 0 \end{bmatrix}, \quad \text{and} \quad \mathbf{e}_m = \begin{bmatrix} 0 \\ 0 \\ \vdots \\ 0 \\ 1 \end{bmatrix}.$$

The ideal prolongation operator $\mathbf{P}$, a $N_t \times N_t/m$ block matrix with blocks of size $N_x \times N_x$, becomes a $N_x \times N_x$ block matrix with blocks of size $N_t \times N_t/m$:

$$\mathcal{P}^{-1}\mathcal{F}^{-1}\mathbf{P}\mathcal{F}_c\mathcal{P}_c = \begin{bmatrix} \widehat{\mathbf{P}}_1 & & \\ & \ddots & \\ & & \widehat{\mathbf{P}}_{N_x} \end{bmatrix},$$

where $\widehat{\mathbf{P}}_k = \mathbf{I}_{N_t/m} \otimes \mathbf{v}$ and $\mathbf{v} = [1, \lambda_k, \lambda_k^2, \ldots, \lambda_k^{m-1}]^{\intercal}$. The injection restriction operator, $\mathbf{R}$, a $N_t/m \times N_t$ block matrix with blocks of size $N_x \times N_x$, becomes a $N_x \times N_x$ block matrix with blocks of size $N_t/m \times N_t$:

$$\mathcal{P}_c^{-1}\mathcal{F}_c^{-1}\mathbf{R}\mathcal{F}\mathcal{P} = \begin{bmatrix} \widehat{\mathbf{R}}_1 & & \\ & \ddots & \\ & & \widehat{\mathbf{R}}_{N_x} \end{bmatrix},$$

where $\widehat{\mathbf{R}}_k(i, 1+m(i-1)) = 1$ for $i = 1, \ldots, N_t/m$, and is zero otherwise.

Spatial prolongation and restriction operators become $N_x/2 \times N_x/2$ diagonal block matrices with blocks of size $2N_t/m \times N_t/m$ and $N_t/m \times 2N_t/m$, respectively. The diagonal blocks of $\widehat{\mathbf{P}}_s$ ($\widehat{\mathbf{R}}_s$) are formed by concatenating two $N_t/m \times N_t/m$ identity matrices vertically (horizontally), each identity matrix scaled by the corresponding entry of the Fourier symbol for the prolongation (restriction) operator. These Fourier symbols are 2×1 and 1×2 matrices, respectively, with entries depending on θ_k. For full weighting and linear interpolation these symbols are $0.5[1+\cos\theta_k, 1-\cos\theta_k]^\intercal$ and $0.5[1+\cos\theta_k, 1-\cos\theta_k]$, thus

$$\widehat{\mathbf{R}}_{s,k} = \begin{bmatrix} \frac{1}{2}(1+\cos\theta_k) & & \frac{1}{2}(1-\cos\theta_k) & \\ & \ddots & & \ddots \\ & \frac{1}{2}(1+\cos\theta_k) & & \frac{1}{2}(1-\cos\theta_k) \end{bmatrix} = \widehat{\mathbf{P}}_{s,k}^\intercal.$$

Spatial coarsening couples together the Fourier harmonics θ_k and $\theta_{k'} = \theta_k - \text{sign}(\theta_k)\pi$, represented by the coupling of fine-grid operator blocks by spatial restriction and prolongation blocks. Let a final permutation place θ_k blocks immediately before $\theta_{k'}$ blocks along the block diagonals. We replace a pair of blocks from $\widehat{\mathbf{A}}_\Delta^{-1}$ (left) with a single, larger block from $\widehat{\mathbf{P}}_s\widehat{\mathbf{A}}_s^{-1}\widehat{\mathbf{R}}_s$ (right):

$$\begin{bmatrix} \widehat{\mathbf{A}}_{\Delta,k}^{-1} & \mathbf{0} \\ \mathbf{0} & \widehat{\mathbf{A}}_{\Delta,k'}^{-1} \end{bmatrix} \quad \longrightarrow \quad \begin{bmatrix} \widehat{\mathbf{P}}_{s,k}\widehat{\mathbf{A}}_{s,k}^{-1}\widehat{\mathbf{R}}_{s,k} & \widehat{\mathbf{P}}_{s,k}\widehat{\mathbf{A}}_{s,k}^{-1}\widehat{\mathbf{R}}_{s,k'} \\ \widehat{\mathbf{P}}_{s,k'}\widehat{\mathbf{A}}_{s,k}^{-1}\widehat{\mathbf{R}}_{s,k} & \widehat{\mathbf{P}}_{s,k'}\widehat{\mathbf{A}}_{s,k}^{-1}\widehat{\mathbf{R}}_{s,k'} \end{bmatrix}.$$

Having transformed all component matrices, we compute the norm of the transformed iteration matrix, $\|\widehat{\mathbf{T}}\|_2$, which serves as an upper bound for error reduction after one MGRIT iteration. As $\widehat{\mathbf{T}}$ has diagonal blocks $\widehat{\mathbf{B}}_k$, $k = 1, 2, \ldots, N_x$, the task of computing its norm simplifies as follows:

$$\|\widehat{\mathbf{T}}\|^2 = \sup_{\mathbf{x}\neq\mathbf{0}} \frac{\|\widehat{\mathbf{T}}\mathbf{x}\|^2}{\|\mathbf{x}\|^2} = \sup_{\mathbf{x}_k\neq\mathbf{0}} \frac{\|\widehat{\mathbf{B}}_1\mathbf{x}_1\|^2 + \cdots + \|\widehat{\mathbf{B}}_{N_x}\mathbf{x}_{N_x}\|^2}{\|\mathbf{x}_1\|^2 + \cdots + \|\mathbf{x}_{N_x}\|^2} = \max_k \sup_{\mathbf{x}_k\neq\mathbf{0}} \frac{\|\widehat{\mathbf{B}}_k\mathbf{x}_k\|^2}{\|\mathbf{x}_k\|^2} = \max_k \|\widehat{\mathbf{B}}_k\|^2.$$

The third equality follows from the fact that we can compute the SVD of $\widehat{\mathbf{T}}$ blockwise from the SVDs of each $\widehat{\mathbf{B}}_k$. Thus to compute $\|\widehat{\mathbf{T}}^n\|$ we maximize $\|\widehat{\mathbf{B}}_k^n\|$ over the set of Fourier frequencies. For time-only coarsening, $\widehat{\mathbf{B}}_k$ is

$$\left(\mathbf{I} - \widehat{\mathbf{P}}_k\widehat{\mathbf{A}}_{\Delta,k}^{-1}\widehat{\mathbf{R}}_k\widehat{\mathbf{A}}_k\right)\widehat{\mathbf{S}}_{t,k}^{\text{FCF}},$$

and for space-time coarsening, $\widehat{\mathbf{B}}_k$ is

$$\left(\begin{bmatrix} \mathbf{I} & \mathbf{0} \\ \mathbf{0} & \mathbf{I} \end{bmatrix} - \begin{bmatrix} \widehat{\mathbf{P}}_k & \mathbf{0} \\ \mathbf{0} & \widehat{\mathbf{P}}_{k'} \end{bmatrix} \begin{bmatrix} \widehat{\mathbf{P}}_{s,k} \end{bmatrix} \begin{bmatrix} \widehat{\mathbf{A}}_{s,k}^{-1} \end{bmatrix} \begin{bmatrix} \widehat{\mathbf{R}}_{s,k} \end{bmatrix} \begin{bmatrix} \widehat{\mathbf{R}}_k & \mathbf{0} \\ \mathbf{0} & \widehat{\mathbf{R}}_{k'} \end{bmatrix} \begin{bmatrix} \widehat{\mathbf{A}}_k & \mathbf{0} \\ \mathbf{0} & \widehat{\mathbf{A}}_{k'} \end{bmatrix}\right) \begin{bmatrix} \widehat{\mathbf{S}}_{t,k}^{\text{FCF}} & \mathbf{0} \\ \mathbf{0} & \widehat{\mathbf{S}}_{t,k'}^{\text{FCF}} \end{bmatrix}.$$

5.3 Analysis of MGRIT With Spatial Coarsening

In this section we compare the results of SAMA for implicit MGRIT with and without spatial coarsening, solving (5.1) over a 128×128 space-time grid for different combinations of the parameters a, ε, Δ_x, and Δ_t. In the case of spatial coarsening we consider both rediscretized and Galerkin definitions for the fully coarsened operator $\mathbf{A}_s$. To do so we plot the iteration matrix block norm after 1 iteration ($\|\widehat{\mathbf{B}}_k\|_2$, left panels) and after 5 iterations ($\|\widehat{\mathbf{B}}_k^5\|_2$, right panels) as a function of Fourier frequency θ. As the plots are symmetric in the y-axis, for clarity we only show the results for $\theta \in [0, \pi]$. In each panel the red plots correspond to (primarily) advection and the blue plots to (primarily) diffusion. The six parameter combinations considered are recorded in Table 5.4.

Case	Advection (a, ε)	Diffusion (a, ε)	(Δ_x, Δ_t)
1	(1,0)	(0,1)	$(2^{-5}, 2^{-5})$
2	(0.1,0)	(0,0.1)	$(2^{-5}, 2^{-5})$
3	(10,0)	(0,10)	$(2^{-5}, 2^{-5})$
4	(1,0.1)	(0.1,1)	$(2^{-5}, 2^{-5})$
5	(1,0)	(0,1)	$(2^{-5}, 2^{-8})$
6	(1,0)	(0,1)	$(2^{-8}, 2^{-5})$

Table 5.4: Parameter combinations for SAMA tests illustrated in Figures 5.1–5.6.

Figure 5.1 is used as the basis of comparison, as it considers pure advection and diffusion with unit parameters. The maximum value attained by a given curve is the value of $\|\widehat{\mathbf{T}}\|$ for this case, representing the worst-case estimate of the MGRIT convergence factor. As we observe for MGRIT with spatial coarsening applied to pure linear advection, several iterations may be required for this to be less than one, which is due to the non-normality of the matrices considered [16]. From this we see that MGRIT performance for the advective case is much worse than in the diffusive case, which agrees with the numerical results from § 5.1. The replacement of the rediscretized coarse-grid operator with the Galerkin operator results in some moderate improvements to spatial coarsening: but it is still significantly worse than no spatial coarsening for the advection problem; for the diffusion problem with spatial coarsening the norm is significantly larger for $\theta \in [0.5, 2.5]$, but it still results in a matrix norm smaller than 10^{-6} after five iterations.

Figures 5.2, 5.3, and 5.4 illustrate how varying a and ε can affect convergence. In Figure 5.2 we see that, for advection with $a = 0.1$, MGRIT with no spatial coarsening has a much lower matrix norm value compared to Figure 5.1 except for θ near zero or π,

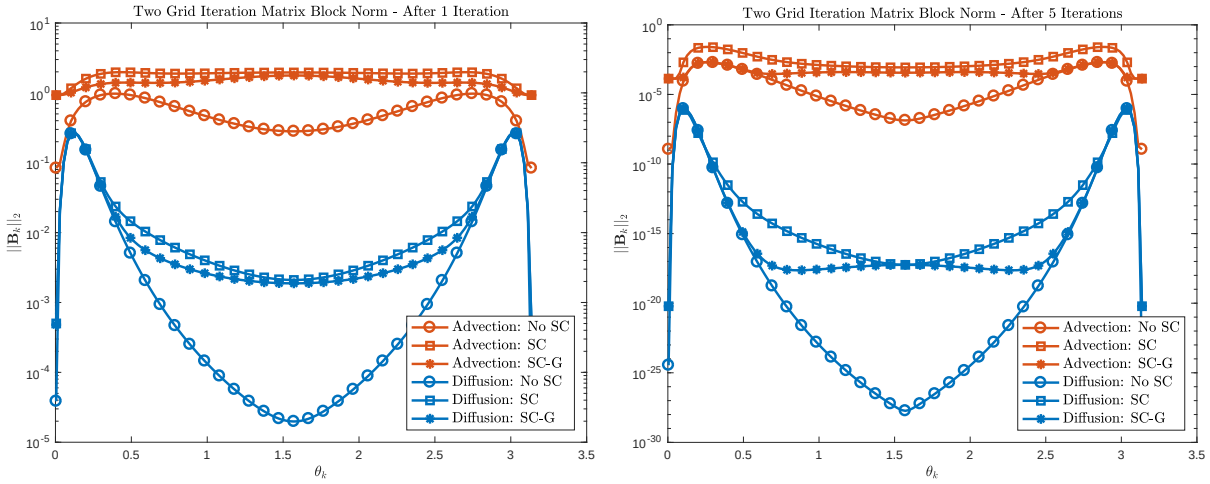

Figure 5.1: Implicit MGRIT: iteration matrix norm dependence on Fourier frequency. Advection: $a = 1$, $\varepsilon = 0$, Diffusion: $a = 0$, $\varepsilon = 1$, $\Delta_x = \Delta_t$.

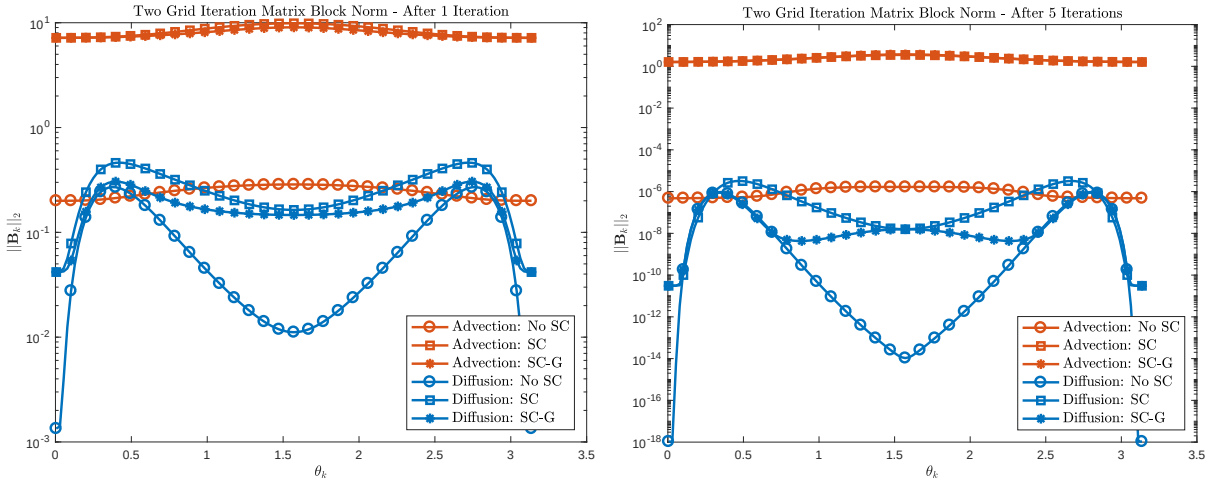

Figure 5.2: Implicit MGRIT: iteration matrix norm dependence on Fourier frequency. Advection: $a = 0.1$, $\varepsilon = 0$, Diffusion: $a = 0$, $\varepsilon = 0.1$, $\Delta_x = \Delta_t$.

corresponding to much faster convergence. In contrast, MGRIT with spatial coarsening performs incredibly poorly, showing almost no decrease in norm value even after 5 iterations. These observations are both due to the fact that as $a \to 0$ the spatial connections become weaker, meaning that MGRIT without SC is more like an exact solver (which is the case for $a = 0$), and the spatial restriction/prolongation operations of MGRIT with SC become increasingly ineffective. In the case of diffusion with $\varepsilon = 0.1$ we see that the matrix

norm values increase, being nearly on par with those for the no SC advection case. Including spatial coarsening results in noticeable deterioration, but the difference is nowhere near as large as that in the advection case.

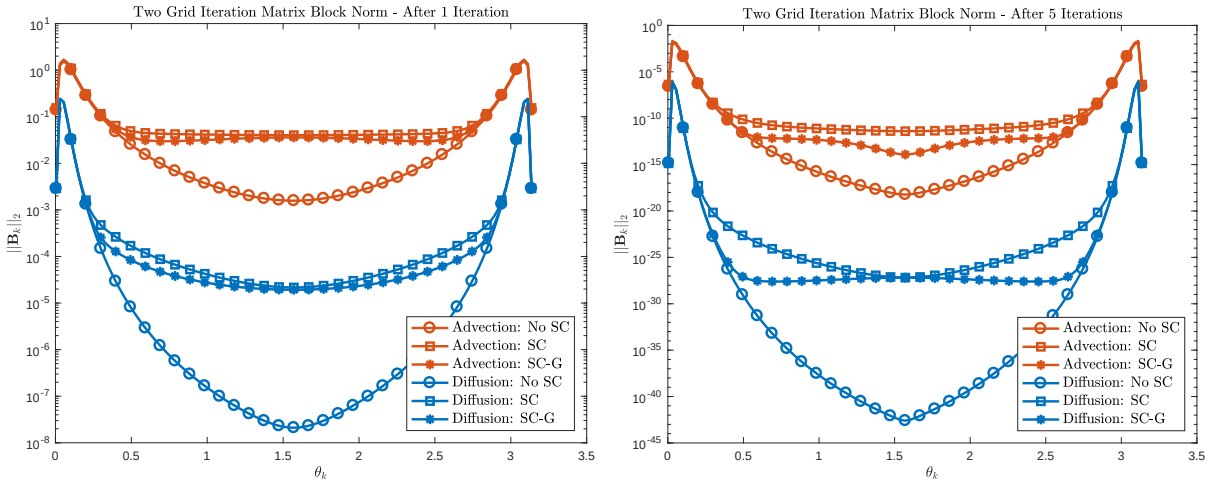

Figure 5.3: Implicit MGRIT: iteration matrix norm dependence on Fourier frequency. Advection: $a = 10$, $\varepsilon = 0$, Diffusion: $a = 0$, $\varepsilon = 10$, $\Delta_x = \Delta_t$.

As observed in 5.3, increasing either a or ε to 10 while keeping the other fixed at 0 results in across the board improvements for all cases, with the only exceptions being for θ values near 0 or π. While MGRIT with spatial coarsening is still less effective than MGRIT without, in such cases the benefits of cheaper iterations will far outweigh the slightly worse rate of convergence indicated.

In Figure 5.4 we consider the inclusion of both spatial derivative terms of (5.1) with the coefficient of the dominant term being an order of magnitude larger than the other. In the advection dominated case (with $a = 1.0$ and $\varepsilon = 0.1$) we see substantial improvements over the results of Figure 5.1, suggesting that advection dominated problems with small to moderate amounts of diffusion can be effectively solved by MGRIT both with or without spatial coarsening. For the diffusion dominated problem (with $a = 0.1$ and $\varepsilon = 1.0$) we see minimal difference between these results and those from Figure 5.1, which suggests that small to moderate amounts of advection will not negatively impact MGRIT convergence for diffusion dominated problems.

In Figures 5.5 and 5.6 we examine the effects of having a significant difference in the sizes of Δ_x and Δ_t (which can be obtained by varying the space-time domain or the number of grid points used in each dimension; the two approaches generate similar results). In Figure 5.5 we have $\Delta_x = 8\Delta_t$, which produces results extremely similar to those in Figure 5.2,

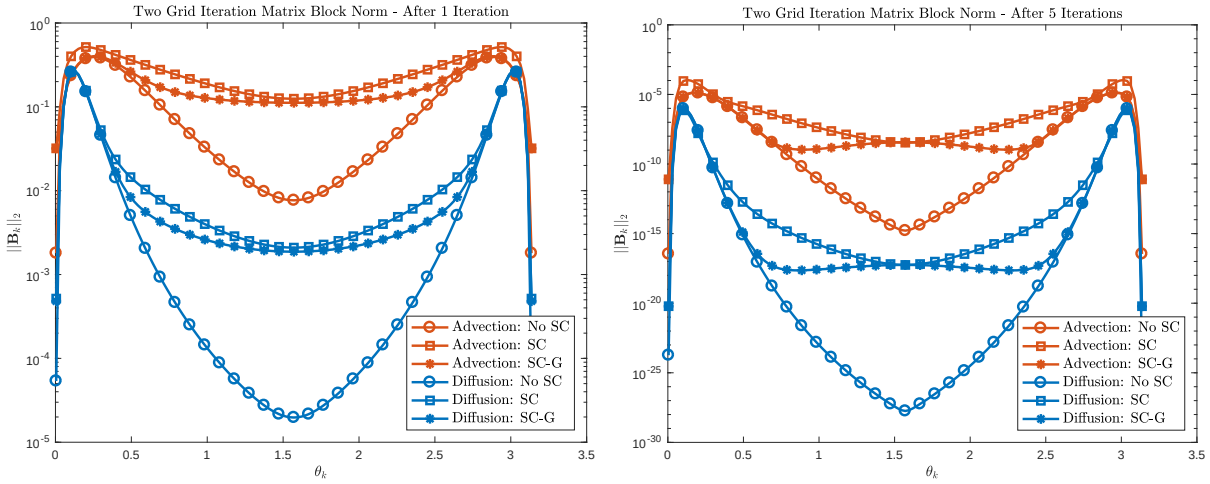

Figure 5.4: Implicit MGRIT: iteration matrix norm dependence on Fourier frequency. Advection: $a = 1$, $\varepsilon = 0.1$, Diffusion: $a = 0.1$, $\varepsilon = 1$, $\Delta_x = \Delta_t$.

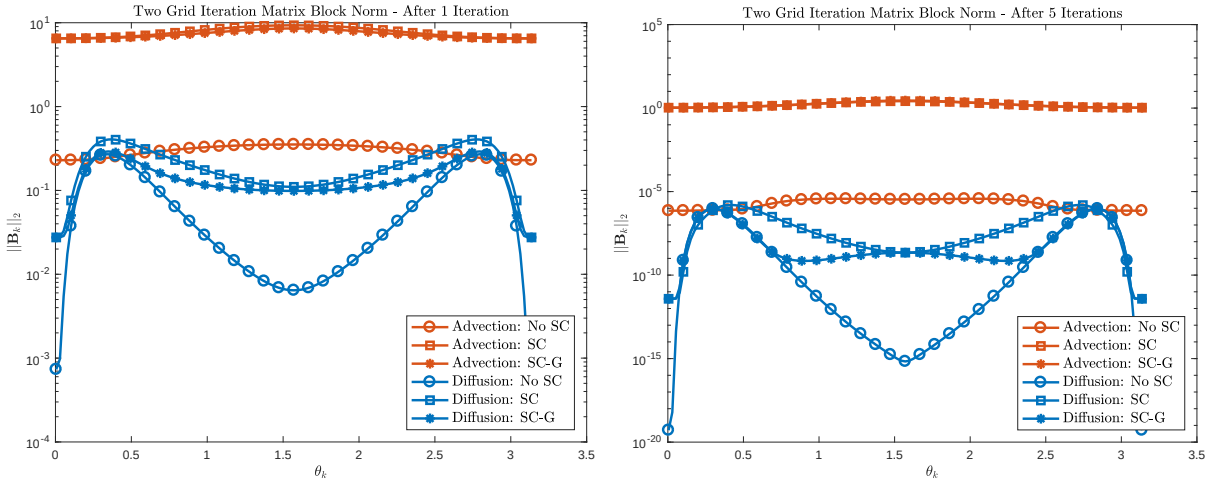

Figure 5.5: Implicit MGRIT: iteration matrix norm dependence on Fourier frequency. Advection: $a = 1$, $\varepsilon = 0$, Diffusion: $a = 0$, $\varepsilon = 1$, $\Delta_x = 8\Delta_t$.

which is not unexpected, as this has the same effect on spatial connections as choosing a or ε to be 0.125. Similarly, the results of 5.6, corresponding to $8\Delta_x = \Delta_t$, are extremely close to those of Figure 5.3, as this had the same effect as taking a or ε to be 8. The conclusion to be drawn from these tests is that coarsening in space without coarsening in time can lead to significant convergence issues, whereas coarsening in time without coarsening in space can lead to improved MGRIT convergence, though one has to be mindful of the CFL

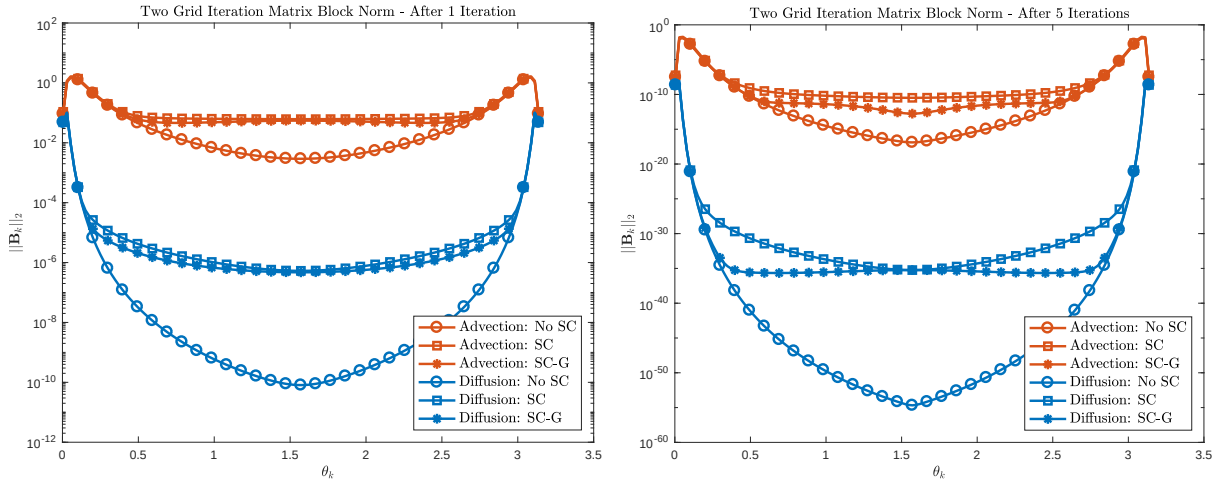

Figure 5.6: Implicit MGRIT: iteration matrix norm dependence on Fourier frequency. Advection: $a = 1$, $\varepsilon = 0$, Diffusion: $a = 0$, $\varepsilon = 1$, $8\Delta_x = \Delta_t$.

condition for explicit MGRIT. These principles will be revisited when discussing adaptive spatial coarsening in the following chapter.

5.4 MGRIT and Waveform Relaxation Multigrid

In this section we discuss how MGRIT with spatial coarsening can be improved by introducing a spatial relaxation on the intermediate grid (coarse-in-time only) based on WRMG [95, 96]. As before, we do this for both the linear advection and the heat equation to illustrate the additional difficulties in the hyperbolic case.

5.4.1 For Implicit MGRIT

To combine WRMG with implicit MGRIT we work with the system for sparse $\mathbf{\Phi}^{-1}_{\Delta x,\Delta t}$ matrices to carry out spatial relaxations:

$$\begin{bmatrix} \mathbf{\Phi}^{-1}_{\Delta x,\Delta t} & & & \\ -\mathbf{I}_{N_x} & \mathbf{\Phi}^{-1}_{\Delta x,\Delta t} & & \\ & \ddots & \ddots & \\ & & -\mathbf{I}_{N_x} & \mathbf{\Phi}^{-1}_{\Delta x,\Delta t} \end{bmatrix} \begin{bmatrix} \mathbf{u}_0 \\ \mathbf{u}_1 \\ \vdots \\ \mathbf{u}_{N_t} \end{bmatrix} = \begin{bmatrix} \mathbf{\Phi}^{-1}_{\Delta x,\Delta t}\mathbf{g}_0 \\ \mathbf{\Phi}^{-1}_{\Delta x,\Delta t}\mathbf{g}_1 \\ \vdots \\ \mathbf{\Phi}^{-1}_{\Delta x,\Delta t}\mathbf{g}_{N_t} \end{bmatrix} = \widetilde{\mathbf{g}}. \tag{5.3}$$

For time-then-space coarsening this necessitates (i) applying $\mathbf{\Phi}_{\Delta x,\Delta t}^{-1}$ to the right-hand-side vector before carrying out spatial relaxations, and (ii) applying $\mathbf{\Phi}_{\Delta x,\Delta t}$ to the spatial residual once computed. For coarsening in space-then-time we instead work with (5.3) by default, converting to the $\mathbf{\Phi}$-on-subdiagonal form to carry out temporal relaxations.

By permuting (5.3) so that blocks correspond to spatial slices rather than time slices, with blocks ordered from 0 to $N_x - 1$ and variables within blocks in increasing temporal order, we obtain

$$\begin{bmatrix} \mathbf{Q} & \mathbf{R} & & \mathbf{P} \\ \mathbf{P} & \mathbf{Q} & \mathbf{R} & \\ & \ddots & \ddots & \\ \mathbf{R} & & \mathbf{P} & \mathbf{Q} \end{bmatrix} \begin{bmatrix} \widehat{\mathbf{u}}_0 \\ \widehat{\mathbf{u}}_1 \\ \vdots \\ \widehat{\mathbf{u}}_{N_x-1} \end{bmatrix} = \begin{bmatrix} \widehat{\mathbf{g}}_0 \\ \widehat{\mathbf{g}}_1 \\ \vdots \\ \widehat{\mathbf{g}}_{N_x-1} \end{bmatrix},$$

where

$$\widehat{\mathbf{u}}_k = \begin{bmatrix} u_k^1 & u_k^2 & \cdots & u_k^{N_t/2} \end{bmatrix}^\intercal \text{ and } \widehat{\mathbf{g}}_k = \begin{bmatrix} \widetilde{g}_k^1 & \widetilde{g}_k^2 & \cdots & \widetilde{g}_k^{N_t/2} \end{bmatrix}^\intercal.$$

For convenience we recall that $\mathbf{K}_n$ is the matrix with ones along the first subdiagonal:

$$\mathbf{K}_n = \begin{bmatrix} 0 & 0 & 0 & \cdots & 0 & 0 \\ 1 & 0 & 0 & \cdots & 0 & 0 \\ 0 & 1 & 0 & \cdots & 0 & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & 0 & \cdots & 1 & 0 \end{bmatrix}.$$

The matrices $\mathbf{P}$, $\mathbf{Q}$, and $\mathbf{R}$ are

$$\mathbf{P} = \alpha \mathbf{I}_{N_t/2}, \quad \mathbf{Q} = (1+\beta)\mathbf{I}_{N_t/2} - \mathbf{K}_{N_t/2}, \quad \mathbf{R} = \gamma \mathbf{I}_{N_t/2},$$

where α, β and γ are the coefficients of u_{k-1}, u_k, and u_{k+1} for the discretized spatial derivative term of the differential equation. In particular:

$$(\alpha, \beta, \gamma) = \begin{cases} \frac{a\Delta t}{\Delta x}(-1, 1, 0) & \text{for linear advection,} \\ \frac{\varepsilon \Delta t}{\Delta x^2}(-1, 2, -1) & \text{for diffusion.} \end{cases}$$

For any three point discretization in space an exact cyclic reduction solve is carried out by

(i) Spatial F-relaxation via timeline solves on F points,
(ii) Residual evaluation at C points,
(iii) Exact solution of the Schur complement coarse-grid system, and

(iv) F-relaxation to propagate correction to F points,

where the spatial relaxation mentioned above is

$$\widehat{\mathbf{u}}_k = \mathbf{Q}^{-1}(\widehat{\mathbf{g}}_k - \mathbf{P}\widehat{\mathbf{u}}_{k-1} - \mathbf{R}\widehat{\mathbf{u}}_{k+1}).$$

The exact cyclic reduction is just as expensive as a linear solve due to the dense Schur complement. To avoid the cost associated with forming and inverting the Schur complement, WRMG replaces it with the matrix corresponding to the differential equation rediscretized on the coarse spatial grid. One step of the WRMG algorithm [95,96] consists of the following steps:

(i) Spatial CF pre-relaxation via timeline solves on C points, then F points,
(ii) Residual evaluation at C points,
(iii) Full weighting restriction (equivalent to half injection for the residuals)
(iv) Exact solution of the rediscretized coarse-grid system,
(v) linear interpolation of coarse-grid correction,
(vi) Spatial FC post-relaxation via solves on F points, then C points.

In Table 5.5 we record the number of iterations required for the two level WRMG algorithm to reach a residual error of 10^{-10} when applied to the advection and diffusion equations. We consider the cases of $N_x = N_t$, $N_x = 2N_t$, and $2N_x = N_t$ for varying problem sizes. In these cases the results for the diffusion equation are independent of problem size, whereas those for advection slightly improve with problem size when $N_x \geq N_t$. Furthermore, advection generally requires approximately twice as many iterations as diffusion, providing further evidence that this problem is intrinsically harder.

We now use this two-level WRMG algorithm as a component of our overall MGRIT iteration. We can either coarsen in time or in space first; in the former case WRMG is applied on intermediate grids (coarse in time only), and in the latter it is applied on the fine and fully coarsened grids, with temporal FCF relaxation occurring on the intermediate grids. We consider both coarsening orderings for advection and diffusion, with results for time-then-space in Table 5.6 and for space-then-time in Table 5.7.

These results are very similar, with the iteration counts for time-then-space coarsening being slightly better, in part due to how residuals are computed on different levels. For advection, while F-cycles exhibit significantly better iteration counts compared to V-cycles, in both cases the iteration count increases with problem size and number of levels, with the combined effects approximately quadrupling the iteration count for the largest V-cycle cases and doubling for the largest F-cycle cases. In contrast, for the diffusion equation both V-cycles and F-cycles scale well with problem size, and F-cycles exhibit no growth as the

(N_x, N_t)	$(2^7, 2^7)$	$(2^8, 2^8)$	$(2^9, 2^9)$	$(2^{10}, 2^{10})$	$(2^{11}, 2^{11})$
Advection	21	21	19	19	18
Diffusion	11	11	11	11	11*

(N_x, N_t)	$(2^6, 2^7)$	$(2^7, 2^8)$	$(2^8, 2^9)$	$(2^9, 2^{10})$	$(2^{10}, 2^{11})$
Advection	21	21	21	22	22
Diffusion	10	11	11	11	11

(N_x, N_t)	$(2^7, 2^6)$	$(2^8, 2^7)$	$(2^9, 2^8)$	$(2^{10}, 2^9)$	$(2^{11}, 2^{10})$
Advection	21	21	19	17	16
Diffusion	11	11	11	11	11

Table 5.5: Iteration counts for two-level WRMG applied to the linear advection and diffusion equations. Asterisks denote stalled convergence at 1.7509e-10.

			(N_x, N_t)					
		levels	$(2^7, 2^7)$	$(2^8, 2^8)$	$(2^9, 2^9)$	$(2^{10}, 2^{10})$	$(2^{11}, 2^{11})$	$(2^{12}, 2^{12})$
Advection	V-cycles	2	19	23	26	28	29	29
		3	20	28	38	50	60	66
		4	20	28	39	55	78	111
		5	20	28	39	55	80	119
	F-cycles	2	19	23	26	28	29	29
		3	19	24	29	34	38	41
		4	19	24	29	36	43	52
		5	19	24	29	36	44	54
Diffusion	V-cycles	2	7	7	8	8	8	8
		3	8	8	8	9	9	9
		4	8	9	9	9	9	9
		5	8	9	9	9	10	10
	F-cycles	2	7	7	8	8	8	8
		3	7	7	8	8	8	8
		4	7	7	8	8	8	8
		5	7	7	8	8	8	8

Table 5.6: Iteration counts for implicit MGRIT combined with WRMG applied to the linear advection and diffusion equations. Time-then-space coarsening.

number of levels in the multigrid hierarchy is increased. Comparing the time-then-space

coarsening results of Tables 5.2, 5.3 and 5.6, we see a significant improvement in iteration counts for advection, with the diffusion results being essentially unchanged.

			(N_x, N_t)					
		levels	$(2^7, 2^7)$	$(2^8, 2^8)$	$(2^9, 2^9)$	$(2^{10}, 2^{10})$	$(2^{11}, 2^{11})$	$(2^{12}, 2^{12})$
Advection	V-cycles	2	24	24	26	28	29	30
		3	24	31	40	51	61	66
		4	24	31	42	58	82	114
		5	24	31	42	59	85	127
	F-cycles	2	24	24	26	28	29	30
		3	24	25	30	35	39	41
		4	24	25	30	37	44	52
		5	24	25	30	37	45	56
Diffusion	V-cycles	2	11	11	12	12	12	12
		3	11	12	12	12	13	13
		4	12	12	12	13	13	13
		5	12	12	12	13	13	13
	F-cycles	2	11	11	12	12	12	12
		3	11	11	12	12	12	12
		4	11	11	12	12	12	12
		5	11	11	12	12	12	12

Table 5.7: Iteration counts for implicit MGRIT combined with WRMG applied to the linear advection and diffusion equations. Space-then-time coarsening.

			(N_x, N_t)				
		levels	$(2^7, 2^7)$	$(2^8, 2^8)$	$(2^9, 2^9)$	$(2^{10}, 2^{10})$	$(2^{11}, 2^{11})$
Advection	V-cycles	2	20	25	27	29	30
		3	21	30	41	55	65
		4	21	30	42	61	90
		5	21	30	42	61	90
	F-cycles	2	20	25	27	29	30
		3	20	26	31	37	41
		4	20	26	31	39	47
		5	20	26	31	39	48

Table 5.8: Iteration counts for implicit MGRIT combined with WRMG applied to the linear advection equation for $(x, t) \in [-2, 2] \times [0, 32]$. Time-then-space coarsening.

			(N_x, N_t)				
		levels	$(2^{10}, 2^7)$	$(2^{11}, 2^8)$	$(2^{12}, 2^9)$	$(2^{13}, 2^{10})$	$(2^{14}, 2^{11})$
Advection	V-cycles	2	20	25	28	29	30
		3	21	30	41	55	65
		4	21	30	42	61	90
		5	21	30	42	61	91
	F-cycles	2	20	25	28	29	30
		3	20	25	31	36	41
		4	20	25	32	39	47
		5	20	25	32	39	48

Table 5.9: Iteration counts for implicit MGRIT combined with WRMG applied to the linear advection equation for $(x, t) \in [-2, 2] \times [0, 4]$. Time-then-space coarsening.

As we are using implicit MGRIT, setting $\Delta t = \Delta x$ does not take advantage of the relaxed stability condition on Δt. We now consider the case of $\Delta t = 8\Delta x$, which can correspond to either (i) keeping $N_t = N_x$ and taking $T_f - T_i = 8(x_f - x_i)$, or (ii) taking $N_t = 8N_x$ while keeping $T_f - T_i = x_f - x_i$. In fact, as illustrated in Tables 5.8 and 5.9, we obtain essentially the same results for cases (i) and (ii) in terms of iteration counts, which suggests that the deterioration in convergence with increasing problem size is linked to the number of time points, N_t, rather than the spatial problem size, N_x. Furthermore, when comparing results for problems of equal size (Tables 5.6 and 5.8), we see that the iteration count exhibits an increase of up to 15% for V-cycles and up to 10% for F-cycles when a ratio of $\Delta t/\Delta x$ 8 times larger.

5.4.2 For Explicit MGRIT

The same method can be used with explicit MGRIT, where the $\mathbf{\Phi}_{\Delta x,\Delta t}$-on-subdiagonal system (4.1) is used on all levels, hence there is no need to adjust $\mathbf{g}_k$ by $\mathbf{\Phi}_{\Delta x,\Delta t}$ or $\mathbf{\Phi}^{-1}_{\Delta x,\Delta t}$ when implementing spatial relaxations on the intermediate grid.

Using the same permutation as before, we again obtain a block tridiagonal system composed of matrices $\mathbf{P}$, $\mathbf{Q}$, and $\mathbf{R}$, that are now given by

$$\mathbf{P} = \alpha\mathbf{K}_{N_t/2}, \quad \mathbf{Q} = \mathbf{I}_{N_t/2} - (1-\beta)\mathbf{K}_{N_t/2}, \quad \mathbf{R} = \gamma\mathbf{K}_{N_t/2},$$

where α, β and γ are the same coefficients of $\mathbf{u}_{k-1}$, $\mathbf{u}_k$, and $\mathbf{u}_{k+1}$ as before. We consider both time-then-space coarsening and space-then-time coarsening, but in contrast to implicit

MGRIT there are significant differences in results for these two orderings. For a basis of comparison an expanded set of results for FCF-relaxation are presented in Table 5.10, which illustrates how iteration counts grow with the number of levels.

			(N_x, N_t)						
		levels	$(2^6, 2^7)$	$(2^7, 2^8)$	$(2^8, 2^9)$	$(2^9, 2^{10})$	$(2^{10}, 2^{11})$	$(2^{11}, 2^{12})$	$(2^{12}, 2^{13})$
Advection	V-cycles	2	33	36	37	38	39	40	42
		3	37	49	62	70	74	77	79
		4	37	49	65	89	118	137	148
		5	37	49	65	89	122	173	234
		MAX	38	49	65	89	122	173	242
	F-cycles	2	33	36	37	38	39	40	42
		3	34	39	44	47	49	51	53
		4	34	39	45	52	60	65	68
		5	34	39	45	52	61	73	88
		MAX	34	39	45	52	61	73	90

Table 5.10: Results for explicit MGRIT with time-then-space coarsening.

Let (ν_1, ν_2) denote ν_1 spatial CF pre-relaxations and ν_2 spatial FC post-relaxations. We consider FCF temporal relaxation with $(1, 0)$, $(0, 1)$, $(1, 1)$, and $(2, 0)$ spatial relaxations, recording the results in Table 5.11. We first note that pre-relaxations can be used without post-relaxations, but not vice versa. Doing more spatial than temporal relaxations on all levels can lead to increased iteration counts for smaller problems and/or fewer levels in the grid hierarchy, which may be a result of information propagating "faster" in the spatial direction than in the temporal direction, violating the CFL condition. Note, however, that results are somewhat improved for the largest problem size and maximum level V-cycles in the $(2, 0)$ case and for F-cycles in the $(1, 1)$ case.

Inspired by the previous results, we considered the possibility of varying the number of spatial relaxations performed as we coarsen. For instance, assuming that $\ell = 1$ is the finest level, Table 5.12 contains results for (i) $(1, 0)$ (ii) $(\min(\ell, 2), 0)$, (iii) $(\min(\ell, 3), 0)$, and (iv) $(\ell, 0)$ spatial relaxations on level ℓ, the first case being a repeat of the top half of Table 5.11. Comparing the bottom row for each of the V- or F-cycle sections we see clear improvements as we move down the table, with the results for $\min(\ell, 3)$ being nearly as good as those in the most extreme expensive case of ℓ spatial relaxations on level ℓ. The exact reason for why results improve to such an extent is still unclear. Furthermore, while no post-relaxations produced the best results observed for unit wave speeds, as will be seen in § 5.4.3, this is not necessarily the case for small or variable wave speeds, for which including post-relaxation produces significant improvements.

			(N_x, N_t)						
		levels	$(2^6, 2^7)$	$(2^7, 2^8)$	$(2^8, 2^9)$	$(2^9, 2^{10})$	$(2^{10}, 2^{11})$	$(2^{11}, 2^{12})$	$(2^{12}, 2^{13})$
$(1,0)$ Spatial Relaxations On Level ℓ	V-cycles	2	23	25	26	27	28	29	29
		3	24	25	28	38	48	56	61
		4	24	25	28	39	55	75	98
		5	24	25	28	39	54	76	108
		MAX	24	25	28	39	54	77	107
	F-cycles	2	23	25	26	27	28	29	29
		3	23	25	26	29	33	37	39
		4	23	25	26	29	35	42	49
		5	23	25	26	29	35	42	52
		MAX	23	25	26	29	35	42	52
$(0,1)$ Spatial Relaxations On Level ℓ	V-cycles	2	29	37	46	54	60	63	66
		3	29	46	71	99	141	187	227
		4	29	46	72	113	172	270	400*
		5	29	46	72	114	182	290	400*
		MAX	29	46	72	114	182	296	400*
	F-cycles	2	29	37	46	54	60	63	66
		3	29	41	53	72	95	117	135
		4	29	42	56	76	108	153	213
		5	29	42	56	77	108	159	237
		MAX	29	42	56	77	109	159	238
$(1,1)$ Spatial Relaxations On Level ℓ	V-cycles	2	26	36	38	39	40	41	42
		3	26	36	47	71	84	86	89
		4	26	36	47	72	96	99	101
		5	26	36	47	72	96	99	101
		MAX	26	36	47	72	96	99	101
	F-cycles	2	26	36	38	39	40	41	42
		3	26	36	39	43	44	45	46
		4	26	36	39	44	45	47	48
		5	26	36	39	44	46	47	48
		MAX	26	36	39	44	45	47	48
$(2,0)$ Spatial Relaxations On Level ℓ	V-cycles	2	28	44	52	55	57	58	64
		3	28	44	52	55	56	58	59
		4	28	44	52	55	57	58	60
		5	28	44	52	55	57	58	72
		MAX	28	44	52	55	57	58	64
	F-cycles	2	28	44	52	54	56	58	59
		3	28	44	52	54	56	58	59
		4	28	44	52	54	56	58	59
		5	28	44	52	54	56	58	59
		MAX	28	44	52	54	56	58	59

Table 5.11: Results for explicit MGRIT combined with WRMG applied to the linear advection equation. Time-then-space coarsening. Asterisks denote tests which failed to converge within 400 iterations.

			(N_x, N_t)						
		levels	$(2^6, 2^7)$	$(2^7, 2^8)$	$(2^8, 2^9)$	$(2^9, 2^{10})$	$(2^{10}, 2^{11})$	$(2^{11}, 2^{12})$	$(2^{12}, 2^{13})$
$(1,0)$ Spatial Relaxation On Level ℓ	V-cycles	2	23	25	26	27	28	29	29
		3	24	25	28	38	48	56	61
		4	24	25	28	39	55	75	98
		5	24	25	28	39	54	76	108
		MAX	24	25	28	39	54	77	107
	F-cycles	2	23	25	26	27	28	29	29
		3	23	25	26	29	33	37	39
		4	23	25	26	29	35	42	49
		5	23	25	26	29	35	42	52
		MAX	23	25	26	29	35	42	52
$(\min(\ell,2),0)$ Spatial Relaxations On Level ℓ	V-cycles	2	23	25	26	27	28	29	29
		3	24	25	28	38	48	56	61
		4	24	25	26	32	43	60	82
		5	24	25	26	32	40	53	74
		MAX	23	25	26	29	34	47	66
	F-cycles	2	23	25	26	27	28	29	29
		3	23	25	26	29	33	37	39
		4	23	25	26	28	32	37	44
		5	23	25	26	28	32	36	42
		MAX	23	25	26	27	29	33	40
$(\min(\ell,3),0)$ Spatial Relaxations On Level ℓ	V-cycles	2	23	25	26	27	28	29	29
		3	23	25	26	30	39	49	57
		4	23	25	26	28	33	44	65
		5	23	25	26	28	34	37	53
		MAX	23	25	26	28	34	37	51
	F-cycles	2	23	25	26	27	28	29	29
		3	23	25	26	28	30	33	37
		4	23	25	26	27	29	33	39
		5	23	25	26	27	29	31	36
		MAX	23	25	26	27	29	31	35
$(\ell,0)$ Spatial Relaxations On Level ℓ	V-cycles	2	23	25	26	27	28	29	29
		3	23	25	26	30	39	49	57
		4	23	25	26	28	33	44	65
		5	23	25	26	28	34	40	48
		MAX	23	25	26	28	34	40	45
	F-cycles	2	23	25	26	27	28	29	29
		3	23	25	26	28	30	33	37
		4	23	25	26	27	29	33	39
		5	23	25	26	27	29	30	34
		MAX	23	25	26	27	29	30	33

Table 5.12: Results for explicit MGRIT combined with WRMG applied to the linear advection equation. Time-then-space coarsening. Use the indicated number of spatial CF pre-relaxations on level ℓ.

For space-then-time coarsening we have found that both pre-relaxations and post-relaxations are necessary for convergence of the method, and that iteration counts improve as the number of spatial CF relaxations increase, as illustrated in Table 5.13, which shows the results for 1 and 2 sets of spatial pre- and post-relaxations. Furthermore, varying the number of spatial or temporal relaxations with the level does not offer any additional acceleration as for time-then-space coarsening. Using ℓ temporal CF relaxations on level ℓ results in much worse convergence than normal, and doing (ℓ, ℓ) spatial relaxations on level ℓ is better than the $(1, 1)$ results but worse than the $(2, 2)$ results.

			(N_x, N_t)						
		levels	$(2^6, 2^7)$	$(2^7, 2^8)$	$(2^8, 2^9)$	$(2^9, 2^{10})$	$(2^{10}, 2^{11})$	$(2^{11}, 2^{12})$	$(2^{12}, 2^{13})$
$(1, 1)$ Spatial Relaxations On Level ℓ	V-cycles	2	27	30	32	33	34	35	36
		3	27	33	42	53	59	64	67
		4	27	33	42	56	74	97	115
		5	27	33	42	56	74	102	143
		MAX	27	33	42	56	74	102	143
	F-cycles	2	27	30	32	33	34	35	36
		3	27	30	33	37	40	42	44
		4	27	30	33	38	44	51	56
		5	27	30	33	38	44	52	63
		MAX	27	30	33	38	44	52	63
$(2, 2)$ Spatial Relaxations On Level ℓ	V-cycles	2	14	20	22	25	27	28	29
		3	14	20	24	32	42	50	57
		4	14	20	24	32	43	60	80
		5	14	20	24	32	43	59	82
		MAX	14	20	24	32	43	59	82
	F-cycles	2	14	20	22	25	27	28	29
		3	14	20	22	26	31	35	37
		4	14	20	22	26	31	37	44
		5	14	20	22	26	31	37	44
		MAX	14	20	22	26	31	37	44

Table 5.13: Results for explicit MGRIT combined with WRMG, with space-then-time coarsening.

5.4.3 For Small or Variable Wave Speeds

As illustrated by the SAMA results in Figure 5.2, MGRIT with spatial coarsening can suffer from extremely slow convergence when the wave speed is near zero. However, this can be remedied through the inclusion of spatial relaxations on the intermediate grid, as

			(N_x, N_t)					
		levels	$(2^7, 2^7)$	$(2^8, 2^8)$	$(2^9, 2^9)$	$(2^{10}, 2^{10})$	$(2^{11}, 2^{11})$	$(2^{12}, 2^{12})$
No WRMG	V-cycles	2	64	100*	100*	100*	100*	100*
		3	64	100*	100*	100*	100*	100*
		4	64	100*	100*	100*	100*	100*
		5	64	100*	100*	100*	100*	100*
	F-cycles	2	64	100*	100*	100*	100*	100*
		3	64	100*	100*	100*	100*	100*
		4	64	100*	100*	100*	100*	100*
		5	64	100*	100*	100*	100*	100*
$(1,1)$ WRMG	V-cycles	2	14	17	20	24	26	28
		3	14	17	21	27	37	48
		4	14	17	21	27	36	50
		5	14	17	21	27	36	50
	F-cycles	2	14	17	20	24	26	28
		3	14	17	20	23	28	32
		4	14	17	20	23	27	33
		5	14	17	20	23	27	33

Table 5.14: Iteration counts for implicit MGRIT with and without WRMG applied to the linear advection equation with small wave speed $a = 0.1$. Coarsening in time-then-space. Asterisks denote tests which failed to converge within 100 iterations.

illustrated by the contents of Tables 5.14 and 5.15, which contain results for (5.1) with $(a, \varepsilon) = (0.1, 0)$ solved by implicit and explicit MGRIT using spatial coarsening with or without spatial relaxations on the intermediate grid. In the implicit case we use $(1, 1)$ spatial relaxations, and for explicit we use $(\min(3, \ell), 1)$ relaxations, with the inclusion of the post-relaxation in the explicit case making a significant difference in performance. These results show that the inclusion of spatial relaxations can have a large impact on the convergence of MGRIT with spatial coarsening. The remaining growth in iteration counts observed is likely related to the choice of temporal relaxation scheme or the coarse-grid time-stepping operator, both of which remain active areas of investigation.

This strategy also works for variable wave speeds, as illustrated in Tables 5.16 and 5.17, which contain results for the linear advection equation with a wave speed $a(x, t)$ that varies both in space and time. Both implicit and explicit MGRIT exhibit typical convergence for the hyperbolic case, whereas without these spatial relaxations they would not converge within a reasonable number of iterations, similar to the top case in Tables 5.14 and 5.15.

Unfortunately, the introduction of the spatial relaxations results in MGRIT becoming a more intrusive algorithm: instead of being able to use an existing time-stepping code the

			(N_x, N_t)					
		levels	$(2^7, 2^8)$	$(2^8, 2^9)$	$(2^9, 2^{10})$	$(2^{10}, 2^{11})$	$(2^{11}, 2^{12})$	$(2^{12}, 2^{13})$
No WRMG	V-cycles	2	64	100*	100*	100*	100*	100*
		3	64	100*	100*	100*	100*	100*
		4	64	100*	100*	100*	100*	100*
		5	64	100*	100*	100*	100*	100*
	F-cycles	2	64	100*	100*	100*	100*	100*
		3	64	100*	100*	100*	100*	100*
		4	64	100*	100*	100*	100*	100*
		5	64	100*	100*	100*	100*	100*
$(\min(3, \ell), 1)$ WRMG	V-cycles	2	13	15	19	22	25	27
		3	13	15	18	22	28	37
		4	13	15	18	22	28	36
		5	13	15	18	22	28	36
	F-cycles	2	13	15	19	22	25	27
		3	13	15	19	22	25	28
		4	13	15	19	22	25	28
		5	13	15	19	22	25	28

Table 5.15: Iteration counts for explicit MGRIT with and without WRMG applied to the linear advection equation with small wave speed $a = 0.1$. Coarsening in time-then-space. Asterisks denote tests which failed to converge due to instability.

			(N_x, N_t)					
		levels	$(2^7, 2^7)$	$(2^8, 2^8)$	$(2^9, 2^9)$	$(2^{10}, 2^{10})$	$(2^{11}, 2^{11})$	$(2^{12}, 2^{12})$
$a(x,t) = -\sin^2(\pi(x-t))$ $(1,1)$ WRMG	V-cycles	2	16	20	24	27	28	30
		3	17	22	29	40	51	61
		4	17	22	29	41	58	83
		5	17	22	29	41	58	84
	F-cycles	2	16	20	24	27	28	30
		3	16	20	24	30	34	38
		4	16	20	24	30	36	44
		5	16	20	24	30	36	44
$a(x,t) = -\sin(2.5\pi t)\sin(\pi x)$ $(1,1)$ WRMG	V-cycles	2	18	19	22	24	26	27
		3	19	23	29	36	44	51
		4	19	23	29	39	52	68
		5	19	23	29	39	53	72
	F-cycles	2	18	19	22	24	26	27
		3	18	20	23	27	31	34
		4	18	20	23	28	33	38
		5	18	20	23	28	33	39

Table 5.16: Iteration counts for implicit MGRIT with WRMG applied to the linear advection equation with variable wave speed. Coarsening in time-then-space.

			(N_x, N_t)					
		levels	$(2^7, 2^8)$	$(2^8, 2^9)$	$(2^9, 2^{10})$	$(2^{10}, 2^{11})$	$(2^{11}, 2^{12})$	$(2^{12}, 2^{13})$
$a(x,t) = -\sin^2(\pi(x-t))$ $(\min(3,\ell),1)$ WRMG	V-cycles	2	19	21	23	25	32	37
		3	19	21	25	27	37	46
		4	19	21	24	25	33	45
		5	19	21	24	25	32	42
	F-cycles	2	19	21	23	25	32	37
		3	19	21	23	25	32	37
		4	19	21	23	25	32	37
		5	19	21	23	25	32	37
$a(x,t) = -\sin(2.5\pi t)\sin(\pi x)$ $(\min(3,\ell),1)$ WRMG	V-cycles	2	18	20	22	24	31	36
		3	19	23	26	30	38	47
		4	19	23	26	27	34	48
		5	19	23	26	27	33	44
	F-cycles	2	18	20	22	24	31	36
		3	18	20	22	25	31	36
		4	18	20	22	25	31	36
		5	18	20	22	25	31	36

Table 5.17: Iteration counts for explicit MGRIT with WRMG applied to the linear advection equation with variable wave speed. Coarsening in time-then-space.

user will now need to provide spatial relaxation routines, which may be extremely difficult to implement depending on the complexity of the problem being solved, whether it results in a linear or nonlinear system, and the time-stepping method being used. An alternative to using WRMG to improve convergence in these cases is adaptive spatial coarsening, which is explored in the following chapter.

Chapter 6

MGRIT with Adaptive Coarsening For Hyperbolic Problems

In this chapter we further explore the use of spatial coarsening in MGRIT and consider local wave speeds ($a(x,t)$ for linear advection, $u(x,t)$ for Burgers' equation) that are close (or equal) to zero on part of the spatial domain, which can result in severe convergence issues. The wave speed for a hyperbolic PDE is the derivative of the flux function: $\lambda(u,x,t) := \partial_u f(u,x,t)$, the characteristic speed with which small-amplitude perturbations propagate. For linear advection we have $\lambda(u,x,t) = a(x,t)$, and for the inviscid Burgers equation $\lambda(u,x,t) = u$.

In § 6.1, we provide some motivating examples that illustrate both why spatial coarsening is desirable, and why adaptive spatial coarsening is necessary in certain cases. In § 6.2, we propose a criterion for determining if spatial coarsening should occur, and provide some examples of the meshes generated by following it. In § 6.3, we describe the cell selection strategies used with explicit and implicit MGRIT and outline a method for moving vectors between grids, which is required for restriction, prolongation, and time-stepping on spatial grids that vary in time. In § 6.4, we provide serial numerical results that demonstrate improved convergence thanks to adaptive coarsening strategy, and in § 6.5, we carry out parallel scaling tests for the linear advection equation which demonstrate that speedups are possible through the use of MGRIT. To perform the numerical tests we use XBraid [2], an open-source implementation of MGRIT.

6.1 Motivating Examples

To illustrate the need for adaptive coarsening we solve both the linear advection problem and Burgers' equation for $(x,t) \in [-2,2] \times [0,4]$ using explicit and implicit MGRIT with FCF-relaxation, factor-two temporal coarsening, and either no spatial coarsening (No SC) or uniform factor-two spatial coarsening (SC-2), which employs full weighting restriction and linear interpolation. The stopping condition is based on the size of the ℓ_2 norm of the residual vector, which uses a halting tolerance of 10^{-10} scaled by the domain size: $\text{tol} = (2.5 \times 10^{-11})\sqrt{N_t N_x}$.

To solve linear advection we impose the initial condition $u_0(x) = \sin(0.5\pi x)$ and consider the constant wave speeds

A1. $a(x,t) = 1.0$, and
A2. $a(x,t) = 0.1$,

for which (4.11) and (4.12) reduce to simple upwinding. The results for these tests are presented in Table 6.1, which records iteration count, time to solution, and time per iteration (TPI). For explicit MGRIT we see the importance of maintaining stability on all levels of the grid hierarchy. For Case A1 the Courant number $\lambda\delta t/\delta x$ for SC-2 is 0.5 on all levels, with temporal coarsening terminating when further spatial coarsening is impossible. Thus, time-stepping is stable on all levels and MGRIT terminates successfully. In contrast, the Courant number for No SC is $2^{\ell-1}$ on level ℓ, where $\ell = 0$ is the finest grid, indicating that time-stepping will be unstable on all coarse levels, hence the majority of 2-level and F-cycle tests failing to converge. However, blindly applying spatial coarsening is not the answer, as illustrated by Case A2, which features a small wave speed that causes weak connections in space in (4.11) and (4.12). Here the Courant number for SC-2 remains fixed at 0.05, hence time-stepping is certainly stable on all levels, but the convergence is extremely poor due to the weak connections. The Courant number for No SC is $0.05(2^{\ell})$, hence time-stepping is stable on the first four coarse grids, and thus while the F-cycles become worse as the problem size grows, the two-level method works well.

For implicit MGRIT the No SC and SC-2 methods produce similar results for Case A1 in terms of iteration count, and there can be substantial savings of approximately 30% in terms of time to solution by using spatial coarsening. For Case A2, however, both the iteration count and time to solution for SC-2 are many times larger than the corresponding values for No SC, making uniform spatial coarsening a non-starter due to the small wave speed.

For Burgers' equation we consider the two different initial conditions

B1. $u_0(x) = 0.75 + 0.25\sin(0.5\pi x)$, and

	$N_x \times N_t$				$2^7 \times 2^8$	$2^8 \times 2^9$	$2^9 \times 2^{10}$	$2^{10} \times 2^{11}$	$2^{11} \times 2^{12}$
Explicit	Case A1	No SC	2-level	It	50	92	100*	100*	100*
				Time (TPI)	0.20 (.004)	0.97 (.01)	3.69 (.03)	16.57 (.16)	59.62 (.59)
			F-cycle	It	100*	100*	100*	100*	100*
				Time (TPI)	1.19 (.012)	4.39 (.04)	14.23 (.14)	51.49 (.51)	327.21 (3.27)
		SC-2	2-level	It	30	30	31	31	31
				Time (TPI)	0.13 (.004)	0.42 (.01)	1.53 (.04)	6.30 (.20)	22.18 (.71)
			F-cycle	It	34	37	41	47	54
				Time (TPI)	**0.30** (.009)	**0.96** (.02)	**3.77** (.09)	**14.73** (.31)	**63.31** (1.17)
	Case A2	No SC	2-level	It	7	7	7	7	7
				Time (TPI)	0.08 (.011)	0.25 (.03)	0.86 (.12)	3.24 (.46)	11.19 (1.59)
			F-cycle	It	8	9	34	100*	100*
				Time (TPI)	0.15 (.019)	0.49 (.05)	5.48 (.16)	53.44 (.53)	199.39 (1.99)
		SC-2	2-level	It	100*	100*	100*	100*	100*
				Time (TPI)	0.34 (.003)	1.17 (.01)	4.72 (.04)	16.15 (.16)	54.83 (.54)
			F-cycle	It	100*	100*	100*	100*	100*
				Time (TPI)	0.77 (.008)	2.27 (.02)	7.38 (.07)	25.62 (.25)	95.90 (.95)
	$N_x \times N_t$				$2^7 \times 2^7$	$2^8 \times 2^8$	$2^9 \times 2^9$	$2^{10} \times 2^{10}$	$2^{11} \times 2^{11}$
Implicit	Case A1	No SC	2-level	It	14	14	15	15	15
				Time (TPI)	0.12 (.009)	0.35 (.02)	1.31 (.08)	5.16 (.34)	26.99 (1.79)
			F-cycle	It	14	15	17	20	22
				Time (TPI)	0.23 (.016)	0.91 (.06)	4.04 (.23)	19.90 (.99)	91.15 (4.14)
		SC-2	2-level	It	15	15	15	16	16
				Time (TPI)	0.09 (.006)	0.32 (.02)	1.18 (.07)	5.01 (.31)	23.72 (1.48)
			F-cycle	It	15	17	20	24	28
				Time (TPI)	**0.15** (.010)	**0.58** (.03)	**2.40** (.12)	**10.56** (.44)	**56.84** (2.03)
	Case A2	No SC	2-level	It	8	8	8	8	8
				Time (TPI)	0.06 (.008)	0.23 (.02)	0.90 (.11)	3.44 (.43)	13.54 (1.69)
			F-cycle	It	8	8	9	9	10
				Time (TPI)	**0.15** (.019)	**0.51** (.06)	**2.14** (.23)	**8.51** (.94)	**37.73** (3.77)
		SC-2	2-level	It	64	90	92	92	92
				Time (TPI)	0.21 (.003)	1.05 (.01)	4.09 (.04)	16.26 (.17)	61.18 (.66)
			F-cycle	It	64	92	94	95	95
				Time (TPI)	0.53 (.008)	2.12 (.02)	7.88 (.08)	30.70 (.32)	114.37 (1.20)

Table 6.1: Linear advection results for Cases A1 and A2. No SC: no spatial coarsening; SC-2: factor-two uniform spatial coarsening. For each problem, the fastest F-cycle results are shown in bold. Asterisks denote tests which failed to converge due to instability (Explicit - No SC) or exceeded 100 iterations.

B2. $u_0(x) = \sin(0.5\pi x)$,

which result in moving and stationary shocks, respectively. The results for these tests are recorded in Table 6.2 (as mentioned in § 4.2, there are significant additional costs when using the Galerkin coarse-grid operator for nonlinear problems, resulting in the much higher

computation times recorded for implicit MGRIT). We see that explicit MGRIT without spatial coarsening fails for both problems, as the maximum local Courant number will be greater than or equal to one on all coarse grids, causing instability. Explicit MGRIT with spatial coarsening has a maximum local Courant number of 0.5 on all levels, ensuring stability, and it successfully terminates when the initial condition is bounded away from zero (Case B1), but fails to converge within 100 iterations when the initial condition has values near zero (Case B2). Implicit MGRIT without spatial coarsening has good iteration counts for both cases, whereas implicit MGRIT with spatial coarsening can offer significant savings in terms of time in Case B1, in spite of the increased iteration count, but fails to converge within 100 iterations in the majority of tests for Case B2.

Combined, these results indicate why spatial coarsening may be desirable for implicit MGRIT and necessary for explicit MGRIT. For explicit MGRIT, No SC will break down once the coarse-grid time step becomes sufficiently large, though it can work for grid hierarchies with few levels where the maximum wave speed is small enough to ensure stability throughout. In contrast, SC-2 ensures stability on all levels. For both explicit and implicit MGRIT, uniform spatial coarsening (SC-2) can work can work well when the wave speed is bounded away from zero, but can exhibit extremely poor convergence when the wave speed is small due to the weak spatial connections. This is analogous to the case of multigrid using Gauss-Seidel or weighted Jacobi applied to strongly anisotropic elliptic problems [16]. For implicit time stepping, SC-2 beats No SC in total time-to-solution when the wave speed is bounded away from zero due to the lower work per cycle.

6.2 Adaptive Spatial Coarsening

The 1D factor-two restriction strategy for a periodic domain is illustrated for four levels and sixteen cells in Figure 6.1. The numerical labels on each level serve as global cell indices, recording which fine-grid reference points are used on coarser levels. Rather than aggregating pairs of adjacent cells when moving from level ℓ to $\ell + 1$, we instead remove every second cell, with remaining cells expanding to cover the removed cells' portion of the domain.

Considering the discretizations (4.11-4.14) and the results of the previous section, we see that a wave speed $\lambda(u, x, t)$ near zero can result in weak couplings in the spatial direction, meaning high frequency errors are not reduced effectively by relaxation. Thus, the error after relaxation cannot be represented properly on coarse spatial grids, drastically reducing the efficiency of a multigrid iteration. Thus, if the wave speed within cell Ω_j is relatively small, we wish to retain Ω_j for the next level, as coarsening in this region will not benefit

	$N_x \times N_t$				$2^7 \times 2^8$	$2^8 \times 2^9$	$2^9 \times 2^{10}$	$2^{10} \times 2^{11}$	$2^{11} \times 2^{12}$
Explicit	Case B1	No SC	2-level	It	2*	2*	2*	2*	2*
				Time (TPI)	0.00 (.000)	0.02 (.01)	0.12 (.06)	0.46 (.23)	1.75 (.87)
			F-cycle	It	2*	2*	2*	2*	2*
				Time (TPI)	0.02 (.010)	0.09 (.04)	0.39 (.19)	1.55 (.77)	6.54 (3.27)
		SC-2	2-level	It	32	33	33	33	33
				Time (TPI)	0.11 (.003)	0.42 (.01)	1.60 (.04)	6.27 (.19)	24.44 (.74)
			F-cycle	It	35	40	45	49	58
				Time (TPI)	**0.27** (.008)	**1.04** (.02)	**4.23** (.09)	**16.30** (.33)	**75.84** (1.30)
	Case B2	No SC	2-level	It	4*	2*	2*	2*	2*
				Time (TPI)	0.01 (.003)	0.03 (.01)	0.12 (.06)	0.48 (.24)	1.90 (.95)
			F-cycle	It	2*	2*	2*	2*	2*
				Time (TPI)	0.02 (.005)	0.10 (.05)	0.42 (.21)	1.67 (.83)	6.41 (3.20)
		SC-2	2-level	It	100*	100*	100*	100*	100*
				Time (TPI)	0.35 (.004)	1.30 (.01)	4.92 (.04)	17.91 (.17)	69.07 (.69)
			F-cycle	It	100*	100*	100*	100*	100*
				Time (TPI)	0.79 (.008)	2.31 (.02)	8.31 (.08)	31.04 (.31)	128.77 (1.28)
	$N_x \times N_t$				$2^7 \times 2^7$	$2^8 \times 2^8$	$2^9 \times 2^9$	$2^{10} \times 2^{10}$	$2^{11} \times 2^{11}$
Implicit	Case B1	No SC	2-level	It	12	13	13	14	14
				Time (TPI)	1.65 (.138)	6.78 (.52)	25.44 (1.95)	106.08 (7.57)	427.33 (30.52)
			F-cycle	It	13	14	16	18	20
				Time (TPI)	5.56 (.428)	24.14 (1.72)	105.91 (6.61)	490.18 (27.23)	2118.34 (105.91)
		SC-2	2-level	It	17	18	18	18	18
				Time (TPI)	2.13 (.125)	8.31 (.46)	30.97 (1.72)	130.59 (7.25)	502.90 (27.93)
			F-cycle	It	17	18	20	23	29
				Time (TPI)	**4.56** (.268)	**17.25** (.95)	**69.04** (3.45)	**292.73** (12.72)	**1418.22** (48.90)
	Case B2	No SC	2-level	It	11	12	12	13	13
				Time (TPI)	1.42 (.129)	5.66 (.47)	20.82 (1.73)	87.56 (6.73)	350.96 (26.99)
			F-cycle	It	11	12	14	15	17
				Time (TPI)	**4.57** (.412)	**19.63** (1.63)	**84.80** (6.05)	**354.06** (23.60)	**1575.60** (92.68)
		SC-2	2-level	It	64	100*	100*	100*	100*
				Time (TPI)	7.90 (.123)	46.52 (.46)	178.77 (1.78)	713.44 (7.13)	2734.99 (27.34)
			F-cycle	It	64	100*	100*	100*	100*
				Time (TPI)	16.84 (.263)	97.58 (.97)	364.91 (3.64)	1379.71 (13.79)	5411.11 (54.11)

Table 6.2: Burgers' equation results for Cases B1 and B2. No SC: no spatial coarsening; SC-2: factor-two uniform spatial coarsening. For each problem, the fastest F-cycle results are shown in bold. Asterisks denote tests which failed to converge due to instability (Explicit - No SC) or exceeded 100 iterations.

the solution process. Experiments (not included here) suggest that it is unnecessary to fix the width of Ω_j; it is sufficient to ensure Ω_j is not removed. To determine if Ω_j is to be kept, we propose the following conditions:

$$\text{Advection: If } \min_{x \in \Omega_j} |\lambda(u,x,t)| \frac{\delta t}{\delta x_j} < \text{tol}_*\text{: keep } \Omega_j\text{; else: coarsen normally.} \tag{6.1}$$

Since $\lambda(u,x,t)\delta t/\delta x_j$ appears in the coefficients of the equations (4.11-4.14), this is an

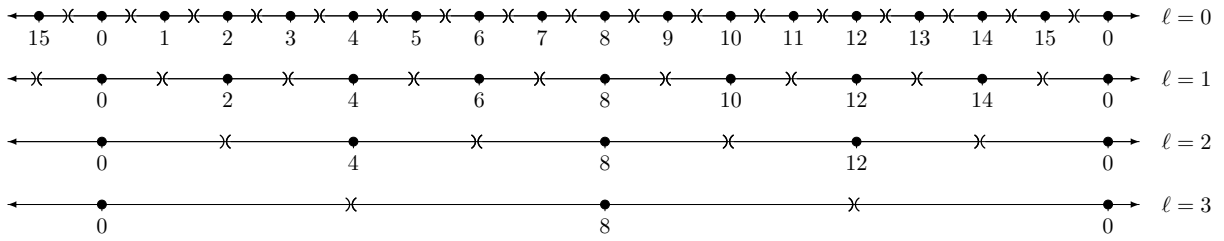

Figure 6.1: Factor-two coarsening in 1D with periodic BCs. The)(symbols represent cell boundaries.

appropriate measure to identify small matrix elements that indicate weak coupling and may lead to degraded multigrid performance if spatial coarsening is used. This approach has similarities to algebraic multigrid [82], where coarsening is operator dependent, based on the strength of different nodal connections. To implement this in XBraid, we create a `grid_info` structure that contains

1. `int *fidx`: array of global cell indices.
2. `double *xref`: array of cell reference points x_j.

The values in `fidx` are global cell indices: for example, level 2 in Figure 6.2 contains 6 cells, which have local indices $\{0, \ldots, 5\}$ and global indices $\{0, 3, 4, 8, 9, 12\}$. An array of `grid_info` structures serves as a grid hierarchy for a given time point t_i. Descriptions of the cell selection strategies employed for implicit and explicit MGRIT are described in the following section.

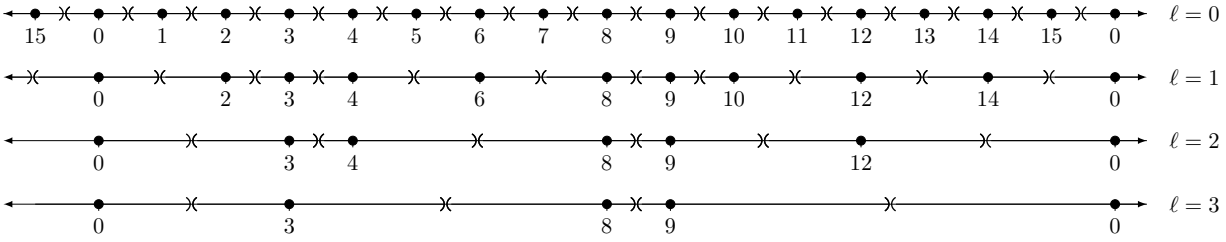

Figure 6.2: Adaptive coarsening in 1D with periodic BCs.

An example of this coarsening process is shown in Figure 6.2 for the same fine grid as in Figure 6.1 at a fixed time point, where (6.1) happened to be satisfied on all levels in cells 3 and 9. The labeled reference points are used to compute cell boundaries as per the definition of vertex-centered grids. It is worth noting that this strategy is easily adapted to non-periodic spatial domains by ensuring that the final cell is retained on all levels. An

easy way of doing so is to take $N_x = 2^k + 1$ for some $k \in \mathbb{N}$, which ensures that the final cell is always part of the uniformly coarsened grid, and hence will also always be part of the adaptively coarsened grid.

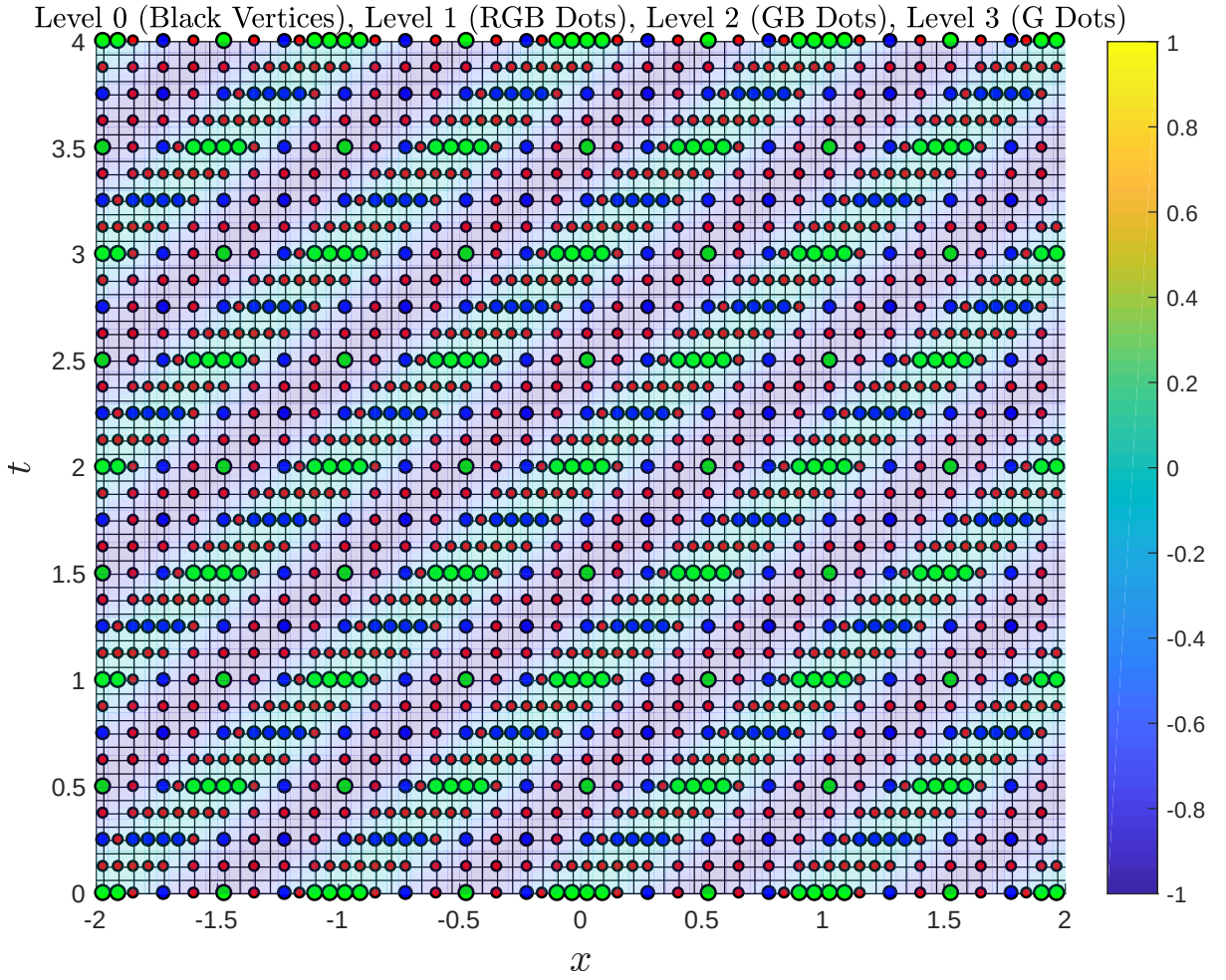

Figure 6.3: Linear advection, $a(x,t) = -\sin^2(\pi(x-t))$ (Case A4): space-time meshes obtained from adaptive spatial coarsening over 4 levels, starting with $N_x = N_t = 64$. The color map indicates the value of $a(x,t)$. Spatial coarsening is inhibited where $|a|$ is small.

In Figures 6.3–6.7 we show adaptive grid hierarchies generated by three rounds of coarsening, starting from a fine 64×64 space-time grid. In all three cases the black vertices indicate reference points for cells only present on level 0, red dots indicate reference points

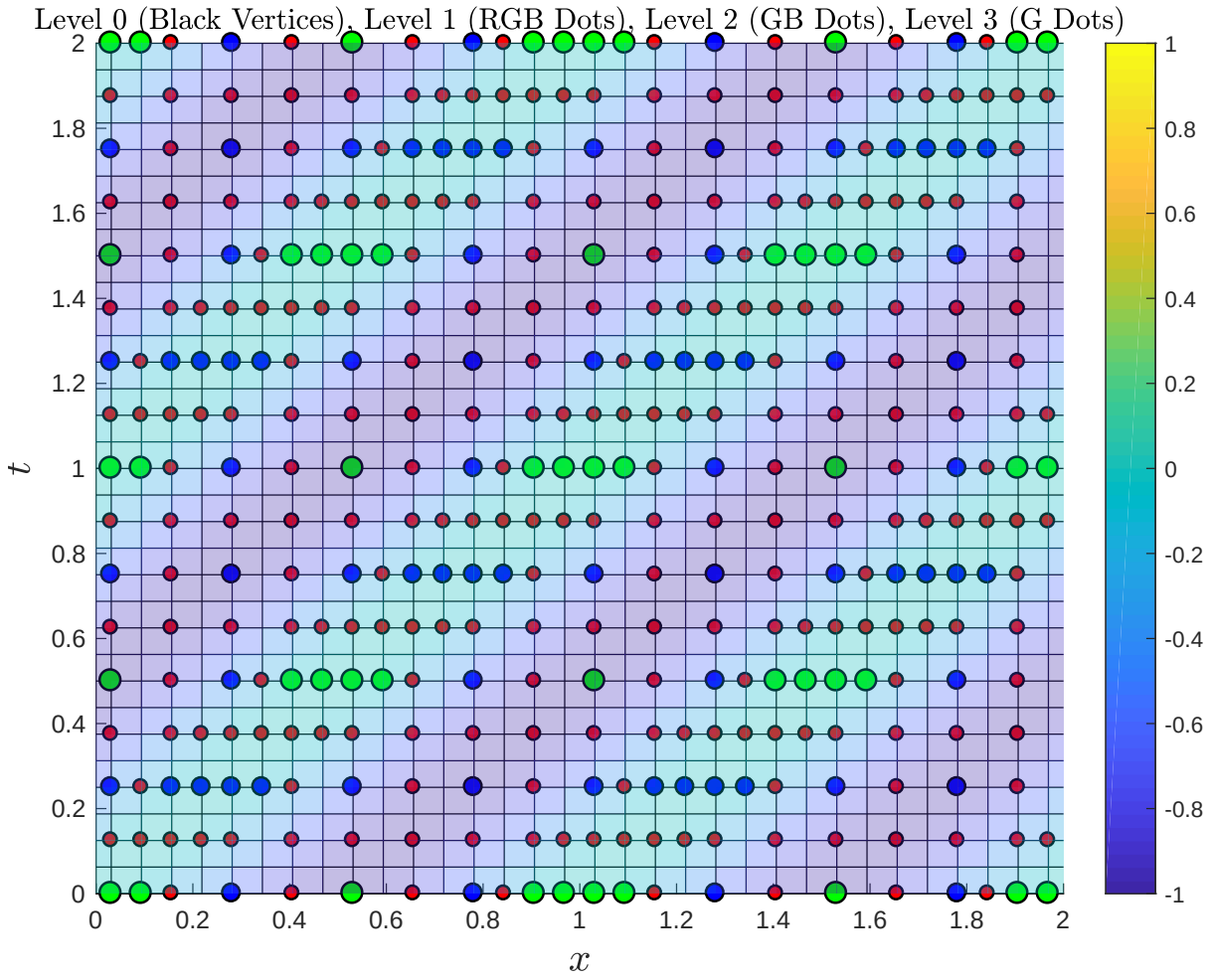

Figure 6.4: Linear advection, $a(x,t) = -\sin^2(\pi(x-t))$ (Case A4): space-time mesh, bottom-right quadrant.

for cells present on levels 0 through 1, blue dots indicate cells present on levels 0 through 2, and green dots indicate cells present on levels 0 through 3. It will be shown in § 6.4 that these grids lead to good MGRIT convergence, and thus adaptive coarsening solves the problem of small local wave speeds.

The first two grids are based on solving the linear advection equation with implicit MGRIT on $[-2,2]\times[0,4]$ for $a(x,t) = -\sin^2(\pi(x-t))$ and $a(x,t) = \frac{1}{2}(1-\sin(2\pi t))\sin(\pi x)$, respectively (these are Cases A4 and A5 defined in § 6.4.1). Due to the periodicity of $a(x,t)$

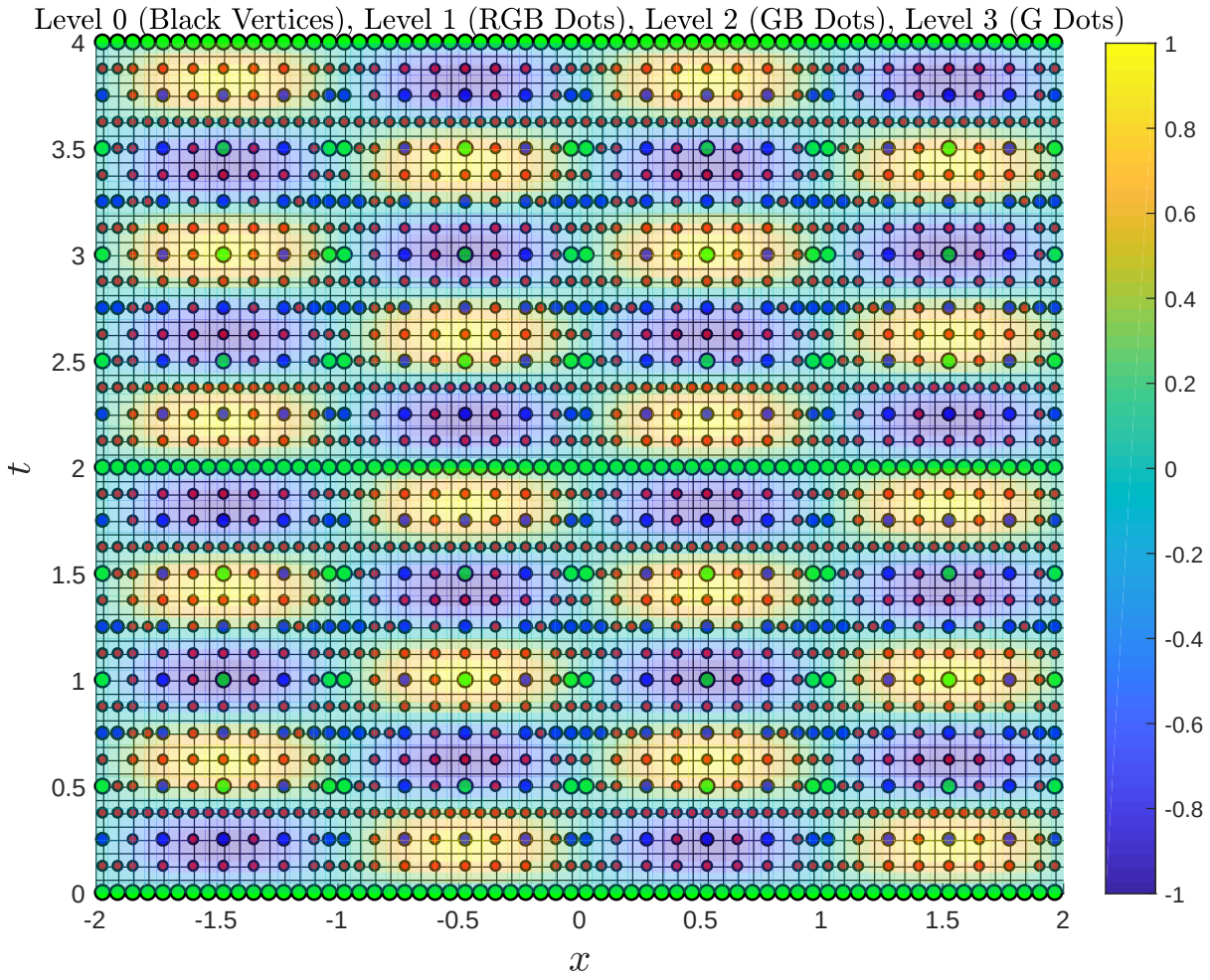

Figure 6.5: Linear advection, $a(x,t) = -\sin(2.5\pi t)\sin(\pi x)$ (Case A5): space-time meshes obtained from adaptive spatial coarsening over 4 levels, starting with $N_x = N_t = 64$. The color map indicates the value of $a(x,t)$. Spatial coarsening is inhibited where $|a|$ is small.

the grid in each quadrant is identical, so we may restrict our discussion to the bottom-right quadrant of each grid, corresponding to $(x,t) \in [0,2] \times [0,2]$ (pictured in Figures 6.4 and 6.6). For Case A4 we see that adaptation results in additional cells being kept along the lines $t = x + b$ for $b \in \mathbb{Z}$, corresponding to the solution of $a(x,t) = 0$. Similarly, for Case A5 we see adaptivity keeps cells along vertical lines defined by integer values of x and horizontal lines defined by multiples of 0.4 for t. Furthermore, by coarsening in

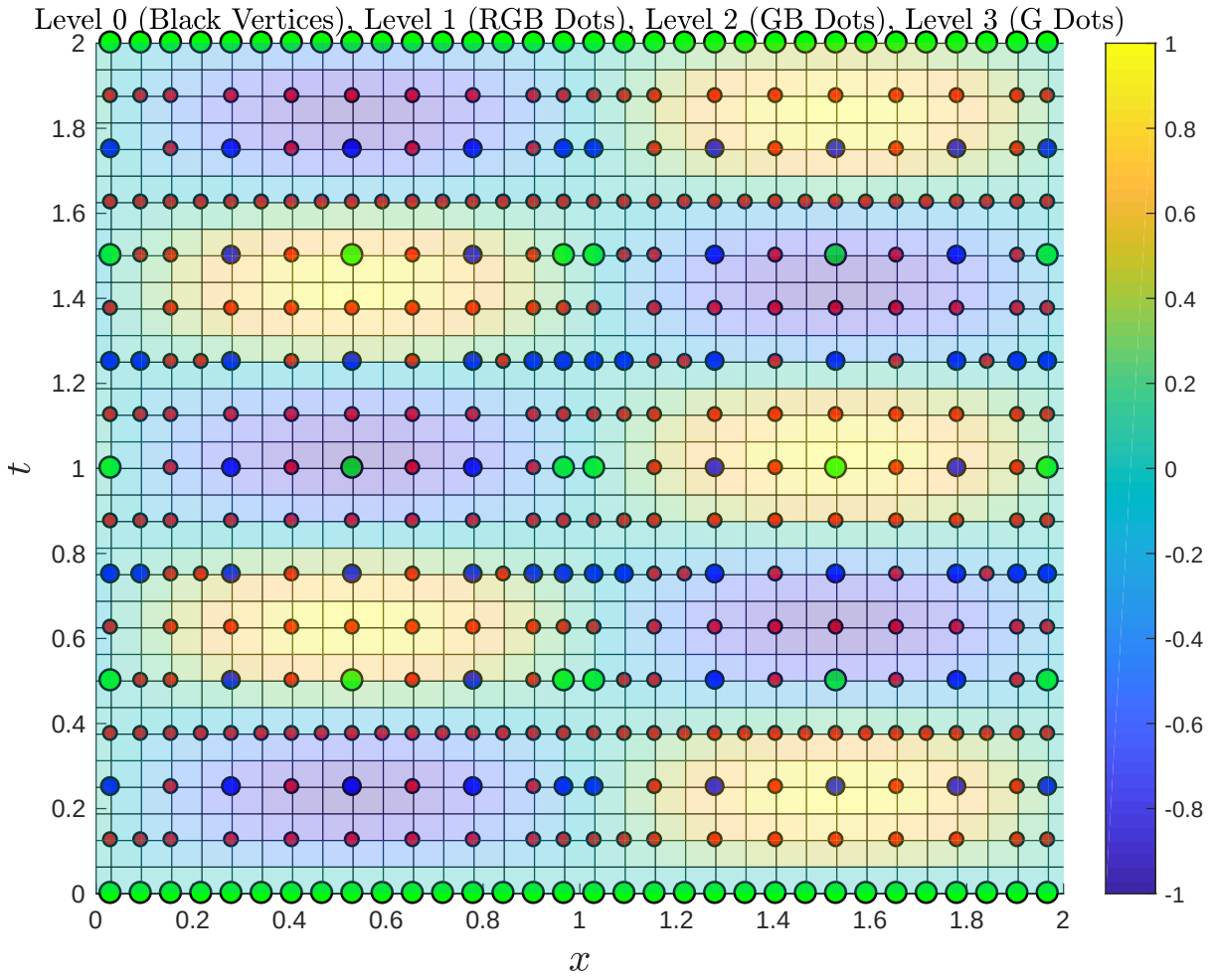

Figure 6.6: Linear advection, $a(x,t) = -\sin(2.5\pi t)\sin(\pi x)$ (Case A5): space-time mesh, bottom-right quadrant.

time we can eliminate the lines near $t = 0.4$ and $t = 1.6$ where no coarsening has taken place, resulting in cheaper coarse-grid problems with no significant deterioration in the convergence of MGRIT.

The grid in Figure 6.7 is based on the solution of Burgers' equation over the domain $(x,t) \in [-4,4] \times [0,8]$ with the initial condition

B3. $u_0(x) = 0.25 - \sin(\pi x/16)$.

These meshes were fixed once the residual norm decreased below 10^{-2} and then used for all remaining iterations. Due to the initial lack of periodicity in the local wave speed (which is the solution $u(x,t)$, pictured in Figure 6.8) we show the entire domain. Once more we see that adaptivity results in more grid cells being retained in regions where the wave speed is near zero, and the location and size of these regions change in response to the evolution of the solution.

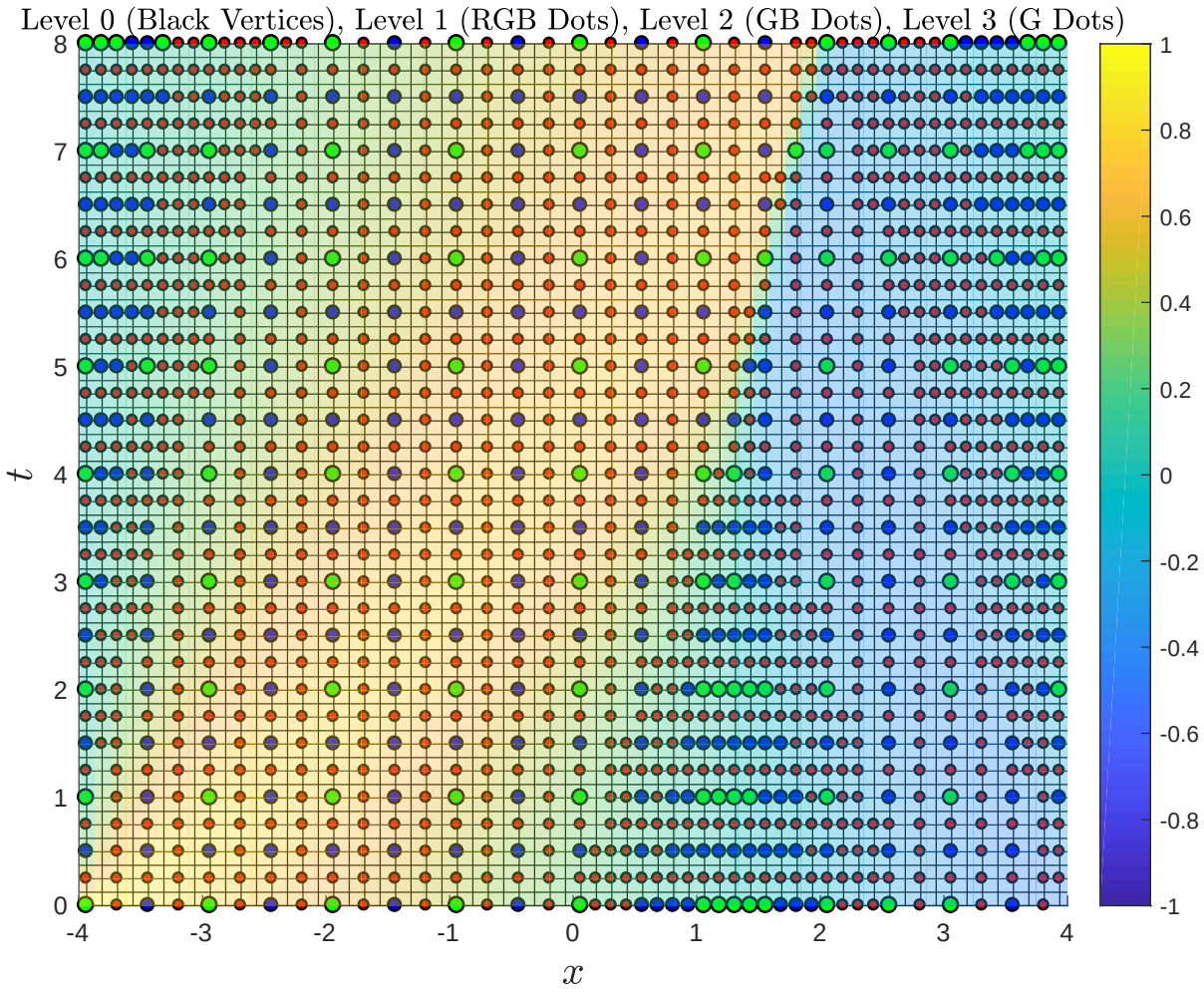

Figure 6.7: Burgers' equation, $u_0(x) = 0.25 - 0.75\sin(\pi x/16)$ (Case B3): space-time mesh obtained from adaptive spatial coarsening over 4 levels, starting with $N_x = N_t = 64$. The color map indicates the value of $u(x,t)$. Spatial coarsening is inhibited where $|u|$ is small.

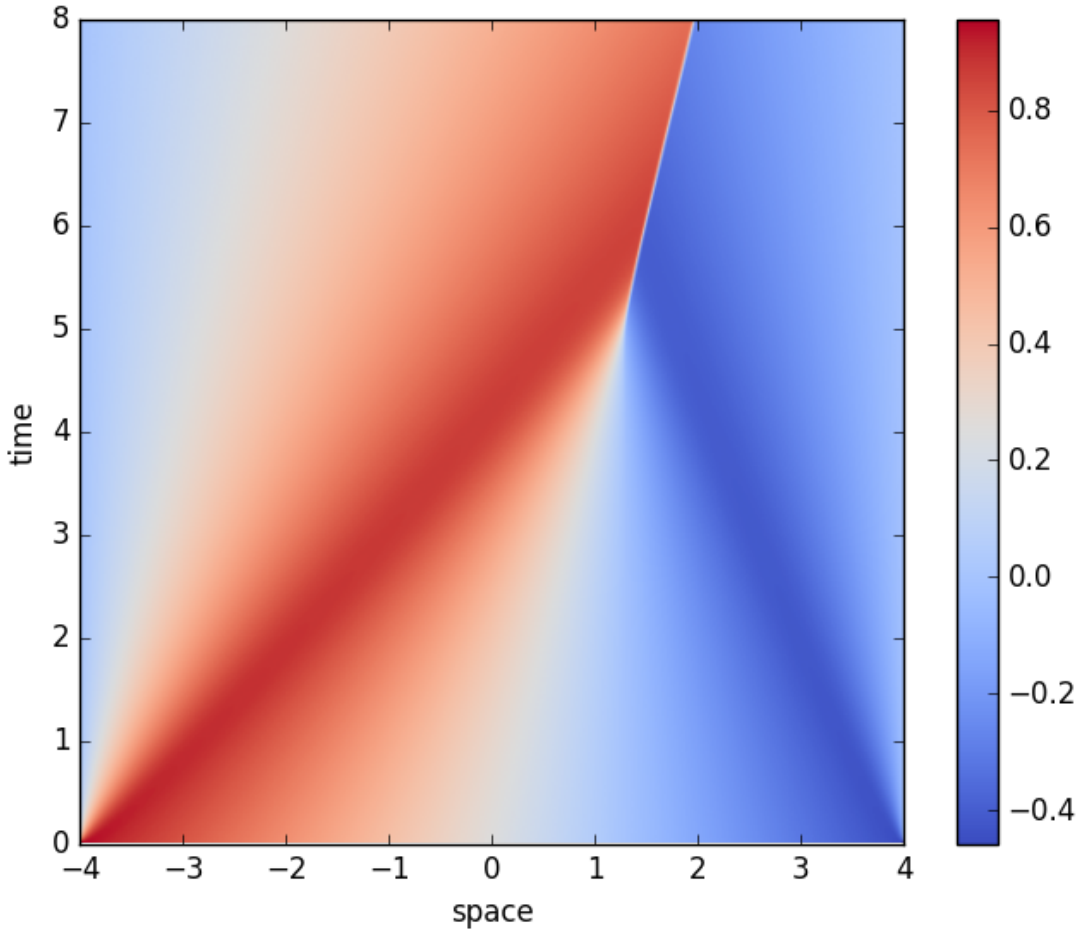

Figure 6.8: Burgers' equation, $u_0(x) = 0.25 - \sin(\pi x/16)$ (Case B3): solution on $[-4, 4] \times [0, 8]$.

6.3 Cell Selection Strategies

The following algorithms are intended as proof of concept for first-order time-stepping routines applied to the linear advection equation and Burgers' equation: further modifications may be required to handle other equations or time-stepping routines. For linear PDEs such as variable coefficient linear advection, the adaptive grid hierarchies generated will not change between MGRIT iterations, so the grids and associated transfer operators need only be computed once and then stored for reuse. In contrast, for nonlinear PDEs such as Burgers' equation the grids can change as the solution approximation is refined, and hence

the adaptive grid hierarchy and the transfer operators will need to be recomputed until a certain MGRIT residual tolerance is reached.

6.3.1 Implicit MGRIT

In our adaptive coarsening strategy we begin with the grid hierarchy generated by uniform factor-two coarsening, meaning that on level ℓ all cells with global indices that are multiples of 2^ℓ are retained. For implicit time-stepping we then use condition (6.1) to identify other cells which should be retained due to small local Courant numbers. Note that, for implicit MGRIT, we do not need to worry about violating a stability constraint when retaining spatial cells while increasing δt. Thus when restricting from level ℓ to $\ell+1$, we keep Ω_j^ℓ if

(i) $\text{fidx}[j] \bmod 2^\ell = 0$, or
(ii) (6.1) holds.

For implicit MGRIT we specify the tolerance in the second condition to be $\text{tol}_* = 0.25$. This cell selection strategy is local in scope, so it can be used in both serial and parallel implementations.

6.3.2 Explicit MGRIT - Linear Advection

To solve linear advection using explicit MGRIT we must ensure $|a(x,t)|\delta t/\delta x_j < 1$ for numerical stability, which necessitates computing the local Courant number for all cells not part of the uniform coarsening grid hierarchy on each level. We need to find the right balance between removing cells as required for stability, and keeping cells to maintain good multigrid convergence corresponding to (6.1). If we consider each cell independently, we may inadvertently end up deleting more cells than necessary for stability, leading to poorer MGRIT convergence. Instead, we collectively consider all cells between each subsequent pair of cells that belong to the uniform grid on the current level and decide which of these non-uniform grid cells must be removed for stability and which should be kept for better convergence.

If there is only one cell between two uniform grid cells, we compute

$$\text{test}_- = \frac{|a(x_{j-1/2},t)|\delta t}{\delta x_{j-1/2}} \text{ and } \text{test}_+ = \frac{|a(x_{j+1/2},t)|\delta t}{\delta x_{j+1/2}}$$

and keep the cell if doing so is beneficial for convergence and is not detrimental for stability:

$$\min(\text{test}_-, \text{test}_+) < \text{tol}_* \text{ and } \max(\text{test}_-, \text{test}_+) < \text{max}_*,$$

where we use $\max_* = 0.95$ and set tol_* to be 0.25 if $\ell = 0$, 0.4 if $\ell = 1$, and 0.49 for $\ell \geq 2$. The values for tol_* were tuned by repeated experimentation and are based on the observation that we can afford, from a computational cost perspective, to keep more spatial cells on coarser grids. Otherwise, for each of the cell interfaces we compute

$$\text{test[j]} = \frac{|a(x_{j+1/2}, t)|\delta t}{\delta x_{j+1/2}}$$

and based on the value of test[j] the interface is labeled as K (keep), N (neutral), or D (delete). Specifically, if test[j] $< \text{tol}_*$ we label this as K, if test[j] $< \max_*$ we label this as N, and otherwise we label it as D. If the sequence of labels is:

(i) X–D–D–$\cdots$–D–X: delete every second cell between D-interfaces (X = K or N)
(ii) N–D–N: delete both cells.
(iii) K–D–N: delete the right cell.
(iv) N–D–K: delete the left cell.
(v) K–D–K: further consideration is required.

In the last case we compute

$$\text{test}_- = \frac{|a(0.5(x_{j+1} + x_{j-1}), t)|\delta t}{\delta x_j} \text{ and } \text{test}_+ = \frac{|a(0.5(x_j + x_{j+2}), t)|\delta t}{\delta x_{j+1}}$$

which are the coarse-grid local Courant numbers that would result from deleting the left or right cells, respectively. We then perform a sequence of comparisons that is designed to remove both cells if the predicted coarse-grid values are both greater than $\max_*$, delete the opposite cell if only one of the test values is greater than $\max_*$, and otherwise keep the cell with the largest Courant value to maintain good MGRIT convergence.

(i) if $\min(\text{test}_-, \text{test}_+) > \max_*$
Delete both cells
(ii) else if $\text{test}_- > \max_*$
Delete right cell
(iii) else if $\text{test}_+ > \max_*$
Delete left cell
(iv) else if $\min(\text{test}_-, \text{test}_+) > \text{tol}_*$ and $\text{test}_- > \text{test}_+$
Delete right cell
(v) else if $\min(\text{test}_-, \text{test}_+) > \text{tol}_*$ and $\text{test}_- \leq \text{test}_+$
Delete left cell
(vi) else if $\text{test}_- > \text{tol}_*$ and $\text{test}_+ < \text{tol}_*$
Delete right cell

(vii) else if $\text{test}_- < \text{tol}_*$ and $\text{test}_+ > \text{tol}_*$
Delete left cell
(viii) else if $\max(\text{test}_-, \text{test}_+) < \text{tol}_*$ and $\text{test}_- > \text{test}_+$
Delete right cell
(ix) else if $\max(\text{test}_-, \text{test}_+) < \text{tol}_*$ and $\text{test}_- \leq \text{test}_+$
Delete left cell

This process is repeated until no D-labeled interfaces remain. If there are multiple adjacent N-interfaces, we next delete every second cell defined by these interfaces. At the end of this process we are left with the cells that are to be kept to ensure effective MGRIT coarse-grid corrections while maintaining stability.

To adapt this process to allow spatial parallelism we only have to make adjustments to account for how the grid is partitioned over the set of processors. If the first (respectively, last) cell on a given processor is not part of the uniform coarsening grid, then we assume that the final cell on the previous processor (respectively, first cell on the next processor) belongs to the uniform coarsening grid, and perform the previously described sequence of tests.

6.3.3 Explicit MGRIT - Burgers' Equation

For Burgers' equation we use a more stringent version of the strategy for linear advection because of the greater likelihood of stability related issues arising in the nonlinear case. As before we keep all cells that are part of the uniform grid, and make use of (6.1) to determine which of the remaining cells will be retained to improve convergence.

If there is only one cell between two uniform grid cells, we compute

$$\text{test}_- = \frac{\max(|u_{j-1}|, |u_j|)\delta t}{\delta x_{j-1/2}} \text{ and } \text{test}_+ = \frac{\max(|u_j|, |u_{j+1}|)\delta t}{\delta x_{j+1/2}}$$

and keep the cell if

$$\max(\text{test}_-, \text{test}_+) < \text{tol}_*,$$

where we set tol_* to be 0.25 if $\ell = 0$, 0.35 if $\ell = 1$, and 0.45 for $\ell \geq 2$. Otherwise, for each of the cell interfaces we compute

$$\text{test[j]} = \frac{|u_j|\delta t}{\delta x_{j+1/2}}$$

and if test[j] $<$ tol$_*$ we label this as K, otherwise labeling it as D. If there are multiple adjacent D-interfaces we delete every second cell that they define, and for isolated D-interfaces we compute

$$\text{test}_+ = \frac{|u_{j+1}|\delta t}{\delta x_{j+1}} \text{ and } \text{test}_- = \frac{|u_j|\delta t}{\delta x_j}$$

and perform the following sequence of tests.

(i) if $\min(\text{test}_-, \text{test}_+) > \text{tol}_*$
Delete both cells

(ii) else if $\text{test}_- > \text{tol}_*$ and $\text{test}_+ < \text{tol}_*$
Delete right cell

(iii) else if $\text{test}_- < \text{tol}_*$ and $\text{test}_+ > \text{tol}_*$
Delete left cell

(iv) else if $\max(\text{test}_-, \text{test}_+) < \text{tol}_*$ and $\text{test}_- > \text{test}_+$
Delete right cell

(v) else if $\max(\text{test}_-, \text{test}_+) < \text{tol}_*$ and $\text{test}_- \leq \text{test}_+$
Delete left cell

This process is repeated until no D-labeled interfaces remain, at which point the remaining cells are those to be kept to ensure effective MGRIT coarse-grid corrections.

6.3.4 Movement Between Grids

In addition to restriction and prolongation of solutions between levels, we also need to transfer solution approximations between time points on a fixed level. For adaptive grid refinement, the grid on a given level may vary with time. This means that a representation of $\mathbf{u}_i$ must be computed on the spatial grid for time t_{i+1} before $\mathbf{u}_{i+1}$ can be computed by time marching.

To map an arbitrary vector $\mathbf{v}$ from grid A to grid B we use the following strategy. For each cell Ω_j^B on grid B, we first identify the cells on grid A that contain its left boundary (Ω_α^A) and right boundary (Ω_ω^A). We compute the cell average v_j^B on Ω_j^B as a weighted average of the cell values from α to ω, scaled by the width of Ω_j^B:

$$v_j^B = \frac{1}{|\Omega_j^B|} \sum_{k=\alpha}^{\omega} |\Omega_j^B \cap \Omega_k^A| v_k^A$$

For periodic boundary conditions, the first cell on both source and target grids may appear as a pair of disconnected intervals: one at the start and one at the end of the domain. To

simplify this case, we treat the disconnected portions as separate cells before merging their results.

For factor-two coarsening, this reduces to full weighting restriction and linear interpolation prolongation, which were our initial choices; and if no spatial coarsening is carried out this reduces to $v_j^{\ell+1} = v_j^{\ell}$. In all cases this approach is conservative.

6.4 Serial Numerical Results

Numerical results within this section were generated using the XBraid parallel-in-time software package [2], and the CHOLMOD [25] and UMFPACK [29] packages from SuiteSparse for sparse matrix multiplication and factorization, respectively.

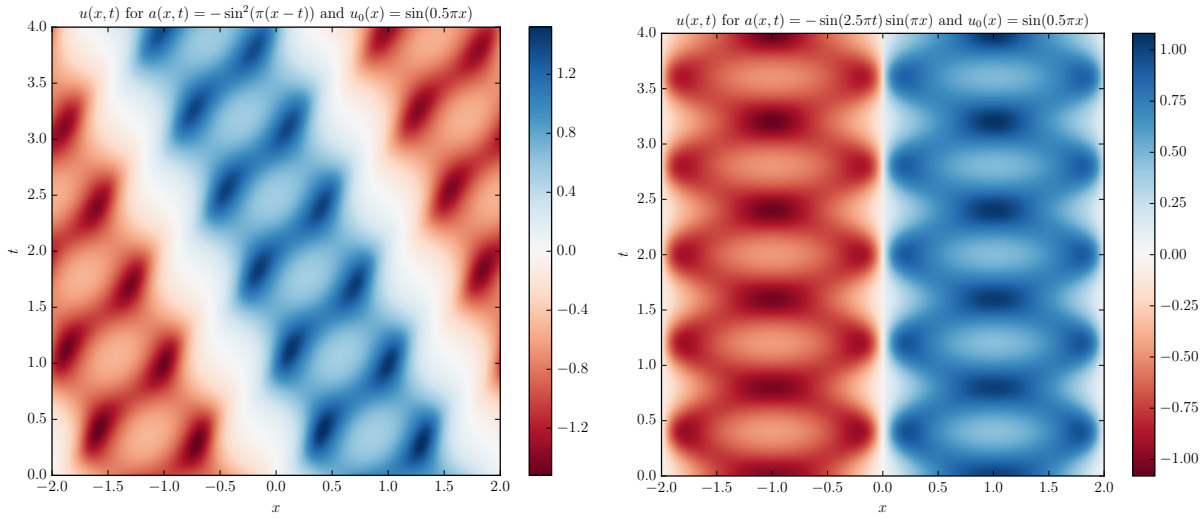

Figure 6.9: Numerical solutions for $a(x,t) = -\sin^2(\pi(x-t))$ (Case A4, left) and $a(x,t) = -\sin(2.5\pi t)\sin(\pi x)$ (Case A5, right).

6.4.1 Linear Advection

We first revisit the linear advection equation with initial condition $u_0(x) = \sin(0.5\pi x)$ solved over $[-2,2] \times [0,4]$ and consider three different variable wave speeds:

A3. $a(x) = -(0.1 + 0.9\cos^2(0.25\pi(x+2)))$ (a varies in space only),

A4. $a(x,t) = -\sin^2(\pi(x-t))$, and
A5. $a(x,t) = -\sin(2.5\pi t)\sin(\pi x)$.

We refer to these as Cases A3, A4, and A5, respectively, and note cases A4 and A5 were previously used to produce the example grids in Figures 6.3 and 6.5 and have the solutions illustrated in Figure 6.9. We solve these problems using MGRIT with factor-two temporal coarsening and one of (i) no spatial coarsening, (ii) factor-two spatial coarsening (for Case A3 only), or (iii) adaptive spatial coarsening. All tests again use a halting tolerance of $\text{tol} = (2.5 \times 10^{-11})\sqrt{N_t N_x}$. Tables 6.3 and 6.4 summarize the results for implicit and explicit MGRIT, respectively.

For implicit Case A3, we see that small $a(x)$ in part of the domain causes significant deterioration for SC-2, and that the adaptive coarsening scheme SC-A recovers good convergence, offering a 33% improvement in total time to solution on the No SC F-cycle results in spite of the increased iterations required. For explicit MGRIT applied to Case A3 we see that SC-A is the only method with convergent F-cycles, as SC-2 once again fails to converge, even for two-level methods. Note that, when comparing the entries of Tables 6.3 and 6.4, we are not concerned with the increased serial time to solution for F-cycles over 2-level cycles, because F-cycles parallelize better. We are instead looking for algorithmic scalability of the F-cycles in terms of iteration count, which we see for both implicit and explicit MGRIT in this case.

For both implicit and explicit MGRIT the additional complexity in Cases A4 and A5 of grid hierarchies that vary in time results in a more costly set-up phase and a greater per-iteration cost when compared to spatial variation only.

For Cases A4 and A5, for both types of time integration the additional complexity of having grid hierarchies that vary in time results in a more costly set-up phase and a greater per-iteration cost when compared to spatial variation only. As in Case A3, SC-2 leads to convergence degradation for implicit integration and outright failure for explicit integration (not shown). We see a benefit to using SC-A over No SC for implicit time-stepping in all test problems once the problem size is large enough. The iterations show a moderate increase as a function of problem size. The near scalability for both implicit and explicit results are promising for very large parallel machines, where gains can be expected over sequential time-stepping due to the vastly increased parallelism in MGRIT. Future work will explore eliminating the growth in iteration count for SC-A compared to No SC, while maintaining a similar time per iteration, thus bringing the iteration counts closer to those for No SC implicit timestepping. Such a result would yield significant savings for both implicit and explicit schemes.

$N_x \times N_t$				$2^7 \times 2^7$	$2^8 \times 2^8$	$2^9 \times 2^9$	$2^{10} \times 2^{10}$	$2^{11} \times 2^{11}$
Case A3	No SC	2-level	It	12	14	14	14	14
			Time (TPI)	0.03 (.003)	0.13 (.009)	0.49 (.03)	2.26 (.16)	8.18 (.58)
		F-cycle	It	12	14	16	18	20
			Time (TPI)	**0.11** (.009)	**0.49** (.035)	2.48 (.15)	12.46 (.69)	56.12 (2.80)
	SC-2	2-level	It	64	78	83	84	85
			Time (TPI)	0.15 (.002)	0.70 (.002)	2.82 (.03)	11.54 (.13)	46.31 (.54)
		F-cycle	It	64	80	85	86	87
			Time (TPI)	0.38 (.006)	1.53 (.019)	6.42 (.07)	25.94 (.30)	95.85 (1.10)
	SC-A	2-level	It	26	27	28	29	29
			Time (TPI)	0.06 (.002)	0.25 (.009)	0.99 (.03)	4.17 (.14)	16.51 (.56)
		F-cycle	It	27	27	28	29	30
			Time (TPI)	0.19 (.007)	0.60 (.022)	**2.35** (.08)	**9.45** (.32)	**37.52** (1.25)
Case A4	No SC	2-level	It	12	12	13	13	13
			Time (TPI)	0.06 (.005)	0.23 (.019)	0.94 (.07)	3.82 (.29)	15.07 (1.15)
		F-cycle	It	12	13	15	16	18
			Time (TPI)	**0.14** (.012)	**0.62** (.048)	2.94 (.19)	13.05 (.81)	60.88 (3.38)
	SC-A	2-level	It	16	15	17	19	22
			Time (TPI)	0.08 (.005)	0.26 (.017)	1.13 (.06)	4.70 (.24)	20.16 (.91)
		F-cycle	It	16	18	20	23	28
			Time (TPI)	0.16 (.010)	0.64 (.036)	**2.54** (.12)	**11.01** (.47)	**51.61** (1.84)
Case A5	No SC	2-level	It	13	12	12	12	13
			Time (TPI)	0.06 (.005)	0.23 (.019)	0.92 (.07)	3.54 (.29)	15.03 (1.15)
		F-cycle	It	13	14	14	16	17
			Time (TPI)	**0.15** (.012)	**0.64** (.046)	**2.62** (.18)	11.86 (.74)	51.02 (3.00)
	SC-A	2-level	It	19	19	20	24	26
			Time (TPI)	0.09 (.005)	0.31 (.016)	1.23 (.06)	5.31 (.22)	22.98 (.88)
		F-cycle	It	20	20	22	24	28
			Time (TPI)	0.20 (.010)	0.71 (.036)	2.77 (.12)	**11.52** (.48)	**49.69** (1.77)

Table 6.3: Linear advection: implicit time-stepping results for Cases A3, A4, and A5. No SC: no spatial coarsening; SC-2: factor-two uniform spatial coarsening, SC-A: adaptive spatial coarsening. For each test problem, the fastest F-cycle results are shown in bold.

6.4.2 Burgers' Equation

We solve Burgers' equation for Case B3 on the spatial domain $[-4, 4]$. As $u_0'(x) < 0$ at some point in the domain, the wave will break and a shock will occur. The time at which characteristics cross and a shock forms is called the *breaking time*, T_b, and for the inviscid Burgers equation this time is given exactly as [68]

$$T_b = -\frac{1}{\min(u_0'(x))}.$$

$N_x \times N_t$				$2^7 \times 2^8$	$2^8 \times 2^9$	$2^9 \times 2^{10}$	$2^{10} \times 2^{11}$	$2^{11} \times 2^{12}$
Case A3	No SC	2-level	It	30	47	100*	100*	100*
			Time (TPI)	0.08 (.003)	0.40 (.009)	3.12 (.03)	12.36 (.12)	49.58 (.49)
		F-cycle	It	100*	100*	100*	100*	100*
			Time (TPI)	1.03 (.010)	3.40 (.034)	12.35 (.12)	48.25 (.48)	192.37 (1.92)
	SC-2	2-level	It	100*	100*	100*	100*	100*
			Time (TPI)	0.28 (.003)	0.85 (.009)	3.06 (.03)	11.79 (.11)	46.19 (.46)
		F-cycle	It	100*	100*	100*	100*	100*
			Time (TPI)	0.71 (.007)	1.98 (.020)	6.23 (.06)	22.65 (.22)	82.90 (.82)
	SC-A	2-level	It	30	30	31	31	32
			Time (TPI)	0.09 (.003)	0.30 (.010)	1.12 (.03)	4.20 (.13)	17.27 (.53)
		F-cycle	It	32	33	35	36	37
			Time (TPI)	**0.30** (.009)	**0.95** (.029)	**3.35** (.09)	**12.61** (.35)	**50.70** (1.37)
Case A4	No SC	2-level	It	20	25	31	38	48
			Time (TPI)	0.06 (.003)	0.26 (.010)	1.21 (.03)	5.78 (.15)	25.59 (.53)
		F-cycle	It	100*	100*	100*	100*	100*
			Time (TPI)	0.96 (.010)	3.28 (.033)	12.16 (.12)	46.98 (.46)	190.90 (1.90)
	SC-A	2-level	It	21	23	27	30	30
			Time (TPI)	0.09 (.004)	0.33 (.014)	1.39 (.05)	5.79 (.19)	21.01 (.70)
		F-cycle	It	21	23	28	31	33
			Time (TPI)	**0.31** (.015)	**1.09** (.047)	**4.47** (.15)	**16.76** (.54)	**67.13** (2.03)
Case A5	No SC	2-level	It	29	42	70	100*	100*
			Time (TPI)	0.08 (.003)	0.38 (.009)	2.33 (.03)	13.19 (.13)	49.91 (.49)
		F-cycle	It	100*	100*	100*	100*	100*
			Time (TPI)	1.02 (.010)	3.50 (.035)	12.76 (.12)	48.36 (.48)	186.78 (1.86)
	SC-A	2-level	It	26	26	27	29	30
			Time (TPI)	0.09 (.004)	0.32 (.012)	1.18 (.04)	4.68 (.16)	18.95 (.63)
		F-cycle	It	27	27	28	30	31
			Time (TPI)	**0.36** (.013)	**1.21** (.045)	**4.02** (.14)	**15.61** (.52)	**59.36** (1.91)

Table 6.4: Linear advection: explicit time-stepping results for Cases A3, A4, and A5. No SC: no spatial coarsening; SC-2: factor-two uniform spatial coarsening, SC-A: adaptive spatial coarsening. For each test problem, the fastest F-cycle results are shown in bold. Asterisks denote tests which failed to converge due to instability.

For this particular example we see that the breaking time is $T_b = 16/\pi \approx 5.09$, which matches the solution for the problem illustrated in Figure 6.8. Based on this observation we solve this problem on both $[-4, 4] \times [0, 4]$ and $[-4, 4] \times [0, 8]$ to consider solutions with and without shock. Test results for the half- and full-domain problems are recorded in Tables 6.5 and 6.6, respectively.

For implicit MGRIT the adaptive coarsening method fails to outperform no spatial

$N_x \times N_t$				$2^7 \times 2^7$	$2^8 \times 2^8$	$2^9 \times 2^9$	$2^{10} \times 2^{10}$	$2^{11} \times 2^{11}$
Implicit Case B3 $[-4,4] \times [0,4]$ Max fine grid CFL: $(1+2\sqrt{2})/4$	No SC	2-level	It	10	11	11	11	11
			Time (TPI)	1.30 (.13)	5.28 (.48)	19.79 (1.79)	76.80 (6.98)	296.21 (26.92)
		F-cycle	It	11	12	12	14	16
			Time (TPI)	**4.58** (.41)	**21.18** (1.76)	**78.39** (6.53)	**332.34** (23.73)	**1496.17** (93.51)
	SC-G	2-level	It	25	27	28	28	29
			Time (TPI)	3.12 (.12)	12.62 (.46)	46.62 (1.66)	180.30 (6.43)	733.46 (25.29)
		F-cycle	It	26	27	28	29	31
			Time (TPI)	7.88 (.30)	30.73 (1.13)	113.13 (4.04)	426.53 (14.70)	1714.89 (55.31)
$N_x \times N_t$				$2^7 \times 2^7$	$2^8 \times 2^8$	$2^9 \times 2^9$	$2^{10} \times 2^{10}$	$2^{11} \times 2^{11}$
Explicit Case B3 $[-4,4] \times [0,4]$ Max fine grid CFL: $(1+2\sqrt{2})/8$	No SC	2-level	It	25	2*	2*	2*	2*
			Time (TPI)	0.04 (.0001)	0.01 (.005)	0.06 (.03)	0.23 (.11)	0.90 (.45)
		F-cycle	It	2*	2*	2*	2*	2*
			Time (TPI)	0.01 (.005)	0.05 (.025)	0.20 (.10)	0.80 (.40)	3.17 (1.58)
	SC-D	2-level	It	29	31	32	32	32
			Time (TPI)	0.09 (.003)	0.33 (.011)	1.25 (.03)	4.94 (.15)	19.44 (.60)
		F-cycle	It	30	34	36	38	42
			Time (TPI)	**0.24** (.008)	**0.95** (.028)	**3.71** (.10)	**13.96** (.36)	**60.56** (1.44)
$N_x \times N_t$				$2^7 \times 2^8$	$2^8 \times 2^9$	$2^9 \times 2^{10}$	$2^{10} \times 2^{11}$	$2^{11} \times 2^{12}$
Explicit Case B3 $[-4,4] \times [0,4]$ Max fine grid CFL: $(1+2\sqrt{2})/16$	No SC	2-level	It	14	15	15	16	16
			Time (TPI)	0.05 (.004)	0.20 (.013)	0.80 (.05)	3.35 (.20)	13.56 (.84)
		F-cycle	It	2*	2*	2*	2*	2*
			Time (TPI)	0.02 (.010)	0.09 (.045)	0.39 (.19)	1.62 (.81)	6.50 (3.25)
	SC-D	2-level	It	19	21	21	22	22
			Time (TPI)	0.12 (.006)	0.45 (.021)	1.82 (.08)	7.29 (.33)	28.40 (1.29)
		F-cycle	It	19	20	21	24	27
			Time (TPI)	**0.38** (.020)	**1.33** (.067)	**5.32** (.25)	**22.62** (.94)	**97.29** (3.60)

Table 6.5: Burgers' equation results for Case B3: no shock formation. No SC: no spatial coarsening; SC-D: adaptive spatial coarsening with rediscretized coarse-grid operator; SC-G: adaptive spatial coarsening with Galerkin coarse-grid operator. The fastest F-cycle results are shown in bold. Asterisks denote tests which failed due to instability.

coarsening in the short domain results due to approximately doubling the iterations required for convergence. Better performance for large grid sizes is observed in the long domain results, due to a relative increase in the no spatial coarsening iteration count and a better time per iteration for the adaptive results (only 46% of the no spatial coarsening time per iteration for the largest test, compared to 59% in the short domain case). Furthermore, the current implementation of the Galerkin definition requires a return to the previous fine grid for each iteration, resulting in an increased time per iteration for adaptive spatial coarsening. This is generally an issue in FAS-style algorithms, which we intend to be a focus of future research.

For explicit MGRIT we note that the results for both the half and full domain tests are very similar, with the main difference being that those in Table 6.6 correspond to using twice as many time steps as those in 6.5 (to maintain the same fine-grid Δt in both cases),

$N_x \times N_t$				$2^7 \times 2^7$	$2^8 \times 2^8$	$2^9 \times 2^9$	$2^{10} \times 2^{10}$	$2^{11} \times 2^{11}$
Implicit Case B3 $[-4,4] \times [0,8]$ Max fine grid CFL: $(1+2\sqrt{2})/4$	No SC	2-level	It	12	13	13	14	14
			Time (TPI)	1.85 (.15)	7.57 (.58)	28.72 (2.20)	115.34 (8.23)	444.03 (31.71)
		F-cycle	It	12	14	15	17	20
			Time (TPI)	**6.40** (.53)	**30.87** (2.20)	135.21 (9.01)	606.01 (35.64)	2846.20 (142.31)
	SC-G	2-level	It	26	27	28	28	29
			Time (TPI)	3.86 (.14)	14.98 (.55)	57.18 (2.04)	219.74 (7.84)	905.99 (31.24)
		F-cycle	It	26	27	28	29	30
			Time (TPI)	8.54 (.32)	32.62 (1.20)	**133.85** (4.78)	**514.33** (17.73)	**1961.20** (65.37)
$N_x \times N_t$				$2^7 \times 2^8$	$2^8 \times 2^9$	$2^9 \times 2^{10}$	$2^{10} \times 2^{11}$	$2^{11} \times 2^{12}$
Explicit Case B3 $[-4,4] \times [0,8]$ Max fine grid CFL: $(1+2\sqrt{2})/8$	No SC	2-level	It	35	2*	2*	2*	2*
			Time (TPI)	0.11 (.003)	0.02 (.010)	0.11 (.05)	0.44 (.22)	1.74 (.87)
		F-cycle	It	2*	2*	2*	2*	2*
			Time (TPI)	0.02 (.010)	0.10 (.050)	0.40 (.20)	1.56 (.78)	5.99 (2.99)
	SC-D	2-level	It	31	32	33	33	33
			Time (TPI)	0.18 (.006)	0.64 (.020)	2.42 (.07)	9.13 (.27)	36.93 (1.11)
		F-cycle	It	32	35	37	42	49
			Time (TPI)	**0.50** (.016)	**2.11** (.060)	**7.21** (.19)	**31.75** (.75)	**142.96** (2.91)
$N_x \times N_t$				$2^7 \times 2^9$	$2^8 \times 2^{10}$	$2^9 \times 2^{11}$	$2^{10} \times 2^{12}$	$2^{11} \times 2^{13}$
Explicit Case B3 $[-4,4] \times [0,8]$ Max fine grid CFL: $(1+2\sqrt{2})/16$	No SC	2-level	It	14	15	16	16	17
			Time (TPI)	0.10 (.007)	0.39 (.026)	1.66 (.10)	6.41 (.40)	27.61 (1.62)
		F-cycle	It	2*	2*	2*	2*	2*
			Time (TPI)	0.05 (.025)	0.19 (.095)	0.85 (.42)	2.93 (1.46)	11.54 (5.77)
	SC-D	2-level	It	20	21	22	22	22
			Time (TPI)	0.22 (.011)	0.83 (.040)	3.57 (.16)	14.13 (.64)	52.82 (2.40)
		F-cycle	It	20	20	21	26	31
			Time (TPI)	**0.78** (.039)	**2.78** (.139)	**9.90** (.47)	**45.71** (1.75)	**217.47** (7.01)

Table 6.6: Burgers' equation results for Case B3: with shock formation. No SC: no spatial coarsening; SC-D: adaptive spatial coarsening with rediscretized coarse-grid operator; SC-G: adaptive spatial coarsening with Galerkin coarse-grid operator. The fastest F-cycle results are shown in bold. Asterisks denote tests which failed due to instability.

which results in times that are approximately doubled. Much like for linear advection, spatial coarsening is necessary for convergence. Adaptive coarsening also greatly improves convergence, but, like in the case of linear advection, we observe modest growth in iteration count with problem size and number of levels in the multigrid cycle. Yet, these results are significant, as we have a convergent method for the inviscid Burgers equation with a shock wave, a difficult problem for parallel-in-time methods, and furthermore the presence of the shock does not lead to convergence degradation compared to the smooth solution.

6.5 Parallel Scaling Results

In this section we present strong and weak parallel scaling results for MGRIT applied to the linear advection equation for $(x,t) \in [-2,2] \times [0,4]$ and $u_0(x) = \sin(0.5\pi x)$ using $a(x,t) = -\sin^2(\pi(x-t))$ (Case A4). The results for $a(x,t) = -\sin(2.5\pi t)\sin(\pi x)$ (Case A5) are similar, hence are relegated to Appendices A.1 and A.2. Results for explicit MGRIT are presented in 6.5.1, followed by results for implicit MGRIT in 6.5.2. We consider different combinations of spatial and temporal parallelism, with spatial parallelism implemented using the hypre package [1] and temporal parallelism implemented using XBraid [2]. These tests were implemented on Vulcan, an IBM Blue Gene/Q machine at Lawrence Livermore National Laboratory consisting of 24,576 nodes, with sixteen 1.6GHz PowerPC A2 cores per node and a 5D Torus interconnect, utilizing up to $2^{17} = 131072$ cores across 8192 nodes.

6.5.1 Explicit Time-stepping

Strong Scaling

For strong scaling tests we use a fine space-time mesh specified by $(N_x, N_t) = (2^n, 2^{n+1})$ for $n = 14$, 15, or 16. We compare MGRIT F-cycles with factor-two temporal coarsening, adaptive spatial coarsening (coarsening $n-1$ times) and space-time parallelism to serial time-stepping with spatial parallelism. Forward Euler time-stepping requires a matrix-vector multiplication, which is easily parallelized using hypre. For each problem size we set the minimum number of processors in each dimension to be $(p_x, p_t) = (2^a, 2^b)$ for fixed a and b. Processors are allocated to spatial and temporal dimensions in three ways:

1. $(p_x, p_t) = (2^{a+k}, 2^{b+k})$ for $k = 0, 1, 2, \ldots$
2. $(p_x, p_t) = (2^{a}, 2^{b+k})$ for $k = 0, 1, 2, \ldots$
3. $(p_x, p_t) = (2^{a+k}, 2^{b})$ for $k = 0, 1, 2, \ldots$

When tabulating results we also consider a fourth combination, where we fix the maximum number of processors at 2^P and consider $(p_x, p_t) = (2^k, 2^{P-k})$ for $k = a, a+1, \ldots, P-b$.

While algorithms for serial time-stepping with only spatial parallelism could be optimized differently from algorithms for MGRIT, we choose to use the same framework in both cases with the intent to provide fair, representative comparisons that would remain consistent for more spatial dimensions and increased problem complexity. Specifically, we use hypre to form and store the sparse matrix used in the matrix-vector product representing a time step.

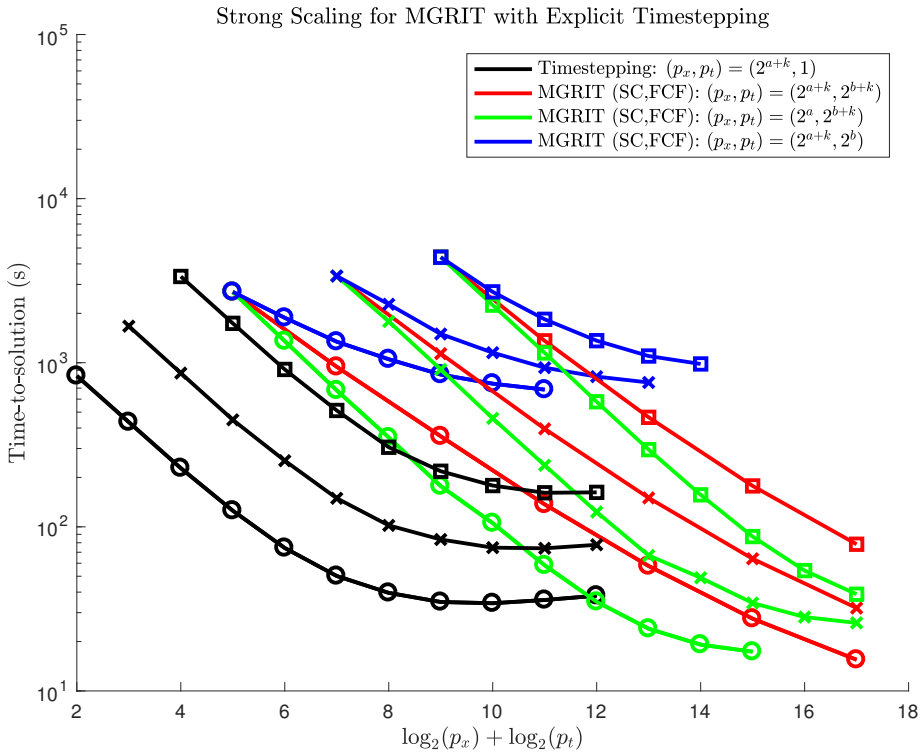

Figure 6.10: Comparison of serial time-stepping with spatial parallelism to explicit MGRIT with different combinations of space-time parallelism for three different problem sizes on up to 131072 cores, $(N_x, N_t) = (2^n, 2^{n+1})$. These results correspond to Tables 6.8–6.11. ○: $n = 14$, $a = 2$, $b = 3$. ×: $n = 15$, $a = 3$, $b = 4$. □: $n = 16$, $a = 4$, $b = 5$.

		(a, b, n)		
		$(2, 3, 14)$	$(3, 4, 15)$	$(4, 5, 16)$
(p_x, p_t)	$(2^{a+k}, 2^{b+k})$	2.21	2.31	2.06
	$(2^{a}, 2^{b+k})$	1.97	2.85	4.15

Table 6.7: Best speedup achieved for explicit MGRIT strong scaling presented in Figure 6.10, $(N_x, N_t) = (2^n, 2^{n+1})$.

In Figure 6.10 we compare serial time-stepping and MGRIT using FCF relaxations for three different problem sizes: $(N_x, N_t) = (2^n, 2^{n+1})$ for $n = 14, 15, 16$. As the basis of comparison we use strong scaling results for serial time-stepping with spatial parallelism. The results for the three different fine grids considered are recorded in Table 6.8, and are

	$(\log_2(N_x), \log_2(N_t))$		
$\log_2(p_x)$	$(14, 15)$	$(15, 16)$	$(16, 17)$
2	831.35	—	—
3	434.53	1670.53	—
4	227.82	864.64	3353.94
5	125.45	448.79	1738.12
6	74.32	253.04	910.10
7	50.15	149.25	511.78
8	39.59	102.02	305.50
9	34.83	83.72	218.12
10	34.23	74.67	178.47
11	35.82	73.98	161.25
12	38.04	77.72	162.20

Table 6.8: Strong scaling for serial explicit time-stepping with increasing amounts of spatial parallelism for fixed problem size.

$\log_2$(pt)	$\log_2$(px)	2	3	4	5	6	7	8
3	iter	37	37	37	37	37	37	37
	time	2698.98	1878.07	1343.52	1050.28	847.13	745.30	685.69
4	iter	37	37	—	—	—	37	—
	time	1353.76	943.93	—	—	—	378.78	—
5	iter	37	—	37	—	37	—	—
	time	678.54	—	356.30	—	219.17	—	—
6	iter	37	—	—	37	—	—	—
	time	350.12	—	—	136.98	—	—	—
7	iter	37	—	37	—	37	—	—
	time	177.02	—	93.50	—	57.69	—	—
8	iter	37	37	—	—	—	37	—
	time	105.31	88.72	—	—	—	27.56	—
9	iter	37	—	—	—	—	—	37
	time	58.28	—	—	—	—	—	15.49

Table 6.9: **Original Size:** Strong scaling for explicit MGRIT, $(N_x, N_t) = (2^{14}, 2^{15})$.

shown as the black curves in Figure 6.10. For smaller amounts of parallelism, doubling the problem size in both dimensions roughly quadruples the time to solution, and at the limit of effective parallelism the time to solution approximately doubles as the problem

$\log_2$(pt)	$\log_2$(px)	3	4	5	6	7	8	9
4	iter	39	39	39	39	39	39	39
	time	3374.07	2271.85	1496.64	1151.49	932.99	822.80	758.01
5	iter	39	39	—	—	—	39	—
	time	1784.16	1135.81	—	—	—	417.14	—
6	iter	39	—	39	—	39	—	—
	time	909.66	—	395.43	—	240.30	—	—
7	iter	39	—	—	39	—	—	—
	time	458.25	—	—	150.03	—	—	—
8	iter	39	—	39	—	39	—	—
	time	237.21	—	104.18	—	63.86	—	—
9	iter	39	39	—	—	—	39	—
	time	123.08	81.64	—	—	—	31.99	—
10	iter	39	—	—	—	—	—	—
	time	66.88	—	—	—	—	—	—

Table 6.10: **Doubled Size:** Strong scaling for explicit MGRIT, $(N_x, N_t) = (2^{15}, 2^{16})$.

$\log_2$(pt)	$\log_2$(px)	4	5	6	7	8	9
5	iter	44	44	44	44	44	44
	time	4395.27	2702.93	1838.73	1362.48	1098.43	983.13
6	iter	44	44	—	—	44	—
	time	2243.41	1359.52	—	—	569.38	—
7	iter	44	—	44	44	—	—
	time	1148.06	—	464.86	349.17	—	—
8	iter	44	—	44	44	—	—
	time	578.08	—	237.94	177.53	—	—
9	iter	44	44	—	—	44	—
	time	295.32	184.29	—	—	78.33	—
10	iter	44	—	—	—	—	—
	time	156.64	—	—	—	—	—

Table 6.11: **Quadrupled Size:** Strong scaling for explicit MGRIT, $(N_x, N_t) = (2^{16}, 2^{17})$.

size is increased. The results in Figure 6.10 are similar for each problem size, where we see that, given enough resources, we are able to improve upon the time-stepping run-times using MGRIT. For a fixed number of processors, the best use of resources is to use the majority for temporal parallelism (green curve) rather than have proportional amounts of temporal and spatial parallelism (red curve). However, when the green curves begin to flatten out there is still potential for more scalability, as indicated by the red curves,

suggesting spatial parallelism should be increased when temporal parallelism approaches the saturation point. The best speed-up observed for the cases of $(p_x, p_t) = (2^{a+k}, 2^{b+k})$ (red curve) and $(p_x, p_t) = (2^a, 2^{b+k})$ (green curve) compared to time-stepping (black curve) are recorded in Table 6.7. The results presented in Tables 6.9, 6.10 and 6.11 illustrate that the iteration count increases modestly with problem size from 37 to 39 to 44, but we do obtain the largest overall parallel speedup for the largest problem size.

Weak Scaling

For weak scaling we increase problem size and processor count while keeping the ratios $N_t : p_t$ and $N_x : p_x$ fixed at $2^{10} : 1$ and the space-time domain fixed at $[-2, 2] \times [0, 4]$. In addition to the original initial condition $u_0(x) = \sin(\pi x/2)$ we also consider the high frequency initial condition $u_0(x) = \sin(2\pi\xi x)$ where ξ is chosen so that there are 16 spatial cells per wavelength. Strong scaling tests were also considered for this initial condition, but the run-times observed were within a few percent of those for the low frequency initial condition, hence they are omitted.

					Original			Oscillatory		
Trial	$\log_2(N_x)$	$\log_2(N_t)$	$\log_2(p_x)$	$\log_2(p_t)$	ξ	Iter	Time	ξ	Iter	Time
1	10	11	0	1	1/4	31	184.83	16	31	183.20
2	11	12	1	2	1/4	33	234.81	32	33	234.19
3	12	13	2	3	1/4	34	247.17	64	34	246.70
4	13	14	3	4	1/4	36	310.55	128	36	309.92
5	14	15	4	5	1/4	39	359.84	256	39	376.04

Table 6.12: Weak scaling for explicit MGRIT with $u_0(x) = \sin(2\pi\xi x)$.

We start with a grid of size $(N_x, N_t) = (2^{10}, 2^{11})$ and either double both N_x and N_t at each step (Table 6.12) or double N_t while leaving N_x fixed (Table 6.13); we cannot increase N_x while leaving N_t fixed due to the CFL condition. If N_x and N_t are increased simultaneously, while increasing core counts from 2 to 512, and problem size from 2M to 512M degrees of freedom, we see only a factor 2 increase in solution time, indicating excellent weak parallel scaling of the MGRIT algorithm. If we increase N_t while N_x remains fixed, we observe decreases in the iteration count and time to solution due to the increasingly weak couplings in space bringing MGRIT closer to an exact solver (when $a(x, t) = 0$ MGRIT with no spatial coarsening converges in one iteration, and in this case the adaptive coarsening forces all spatial cells to be kept on all levels). It is interesting to observe that

					Original			Oscillatory		
Trial	$\log_2(N_x)$	$\log_2(N_t)$	$\log_2(p_x)$	$\log_2(p_t)$	ξ	Iter	Time	ξ	Iter	Time
1	10	11	0	1	1/4	31	184.84	16	31	183.20
2	10	12	0	2	1/4	14	110.47	16	14	109.81
3	10	13	0	3	1/4	11	98.35	16	12	104.09
4	10	14	0	4	1/4	9	88.08	16	10	94.11
5	10	15	0	5	1/4	7	76.70	16	8	82.96
6	10	16	0	6	1/4	6	71.93	16	6	71.68
7	10	17	0	7	1/4	5	68.24	16	5	68.06

Table 6.13: Weak scaling for explicit MGRIT with $u_0(x) = \sin(2\pi\xi x)$ and fixed N_x.

the results for the different initial conditions are extremely similar, suggesting that the scalability is robust for oscillatory solutions, where the solution is changing at the scale of the grid spacing.

6.5.2 Implicit Results

Strong Scaling

For implicit time-stepping we use a fine space-time mesh with equal resolution in both dimensions specified by $(N_x, N_t) = (2^{14}, 2^{14})$ and set the tolerance in our coarsening condition (6.1) to be $\text{tol}_* = 0.25$. Serial time-stepping with spatial parallelism is compared to MGRIT F-cycles with factor two temporal coarsening, either no spatial coarsening or adaptive spatial coarsening (coarsening $n - 1$ times), and space-time parallelism. Backward Euler time-stepping requires tridiagonal solves which are parallelized by using the hypre 1D cyclic reduction solver. Processors are allocated as in § 6.5.1 for the explicit case, except that we start with $a = b = 2$.

		No SC	SC
(p_x, p_t)	$(2^k, 2^k)$	6.77	3.87
	$(2^4, 2^k)$	5.43	5.08

Table 6.14: Best speedup achieved for implicit MGRIT strong scaling presented in Figure 6.11, $(N_x, N_t) = (2^{14}, 2^{14})$.

In Figure 6.11 we compare the results of serial implicit time-stepping to MGRIT with

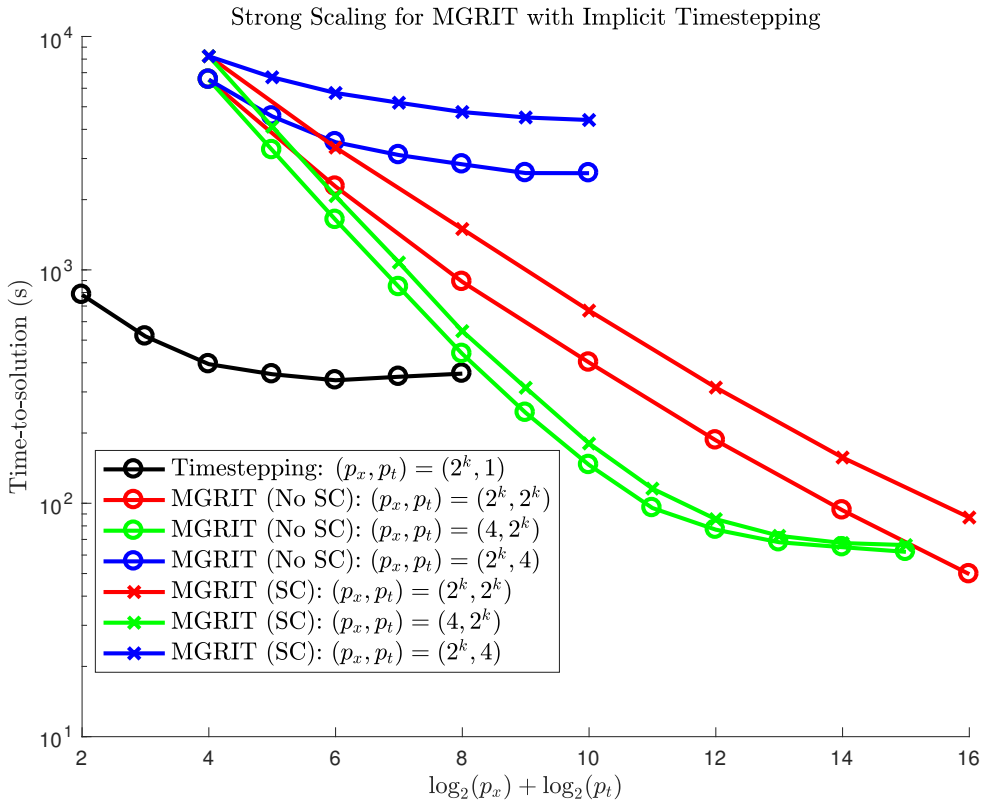

Figure 6.11: Comparison of serial time-stepping with spatial parallelism to MGRIT with or without spatial coarsening for different combinations of space-time parallelism on up to 65536 cores, $(N_x, N_t) = (2^{14}, 2^{14})$. These results correspond to Tables 6.15–6.17.

FCF temporal relaxation and either with or without spatial coarsening. As the basis of comparison we use strong scaling results for serial time-stepping with spatial parallelism (black curves), as recorded in Table 6.15. We see that the combination of communication costs and the lower amount of concurrency possible for the cyclic reduction spatial solve result in methods quickly reaching the effective limit of spatial parallelism. In contrast, significant improvements on the serial time-stepping results are possible once enough temporal parallelism has been introduced. Similar to the explicit case, we see that for up to 2^{14} processors the best results are obtained by investing the majority into temporal parallelism, though further scalability is possible if spatial parallelism is increased as temporal parallelism approaches the saturation point, which would offer improved results for 2^{12} or more processors. The difference between SC and no SC is most pronounced in the cases where $p_x \geq p_t$, with the difference between the SC and no SC curves remaining nearly

constant as the processor count increases. The best speedup for the No SC and SC cases are recorded in Table 6.14.

	$(\log_2(N_x), \log_2(N_t))$
$\log_2(p_x)$	$(14, 14)$
2	782.02
3	518.83
4	394.58
5	357.26
6	336.55
7	348.35
8	359.36

Table 6.15: Strong scaling for serial implicit time-stepping with increasing spatial parallelism, $(N_x, N_t) = (2^{14}, 2^{14})$.

In Tables 6.16 and 6.17 we tabulate the results from the previous figure. Comparing the SC and no SC results, we see that the SC iteration counts are approximately 1.5 times as large as the iteration counts for no SC (increasing from 26 to 40), indicating that if this increase can be ameliorated we could see significant improvements in the SC time to solution.

$\log_2$(pt)	$\log_2$(px)	2	3	4	5	6	7	8
2	iter	46	46	46	46	46	46	46
	time	8230.19	6686.69	5728.51	5202.09	4745.97	4498.12	4386.64
3	iter	46	46	—	—	—	46	—
	time	4106.38	3342.14	—	—	—	2260.45	—
4	iter	46	—	46	—	46	—	—
	time	2071.04	—	1496.78	—	1203.23	—	—
5	iter	46	—	—	46	—	—	—
	time	1075.52	—	—	669.24	—	—	—
6	iter	46	—	46	—	46	—	—
	time	546.51	—	398.02	—	313.18	—	—
7	iter	46	46	—	—	—	46	—
	time	312.58	277.75	—	—	—	157.05	—
8	iter	46	—	—	—	—	—	46
	time	180.42	—	—	—	—	—	86.90

Table 6.16: **Spatial Coarsening:** Strong scaling for implicit MGRIT, $N_x = N_t = 2^{14}$.

$\log_2$(pt)	$\log_2$(px)	2	3	4	5	6	7	8
2	iter	30	30	30	230	30	29	30
	time	6531.90	4547.23	3524.90	3104.28	2833.67	2599.01	2596.10
3	iter	30	30	—	—	—	30	—
	time	3261.04	2274.50	—	—	—	1348.17	—
4	iter	30	—	29	—	30	—	—
	time	1638.08	—	888.57	—	716.87	—	—
5	iter	30	—	—	30	—	—	—
	time	844.48	—	—	400.95	—	—	—
6	iter	30	—	30	—	30	—	—
	time	436.60	—	241.86	—	186.06	—	—
7	iter	30	30	—	—	—	30	—
	time	244.24	176.49	—	—	—	93.29	—
8	iter	30	—	—	—	—	—	29
	time	145.93	—	—	—	—	—	49.69

Table 6.17: **No Spatial Coarsening:** Strong scaling for implicit MGRIT, $N_x = N_t = 2^{14}$.

Weak Scaling

					Original			Oscillatory		
Trial	$\log_2(N_x)$	$\log_2(N_t)$	$\log_2(p_x)$	$\log_2(p_t)$	ξ	Iter	Time	ξ	Iter	Time
1	10	10	0	0	1/4	21	239.23	16	23	276.81
2	11	11	1	1	1/4	25	554.72	32	25	596.79
3	12	12	2	2	1/4	29	808.77	64	29	859.19
4	13	13	3	3	1/4	37	1167.96	128	37	1245.90
5	14	14	4	4	1/4	45	1401.77	256	46	1489.30

Table 6.18: Weak scaling for implicit MGRIT with $u_0(x) = \sin(2\pi\xi x)$.

For weak scaling tests we again consider the original initial condition $u_0(x) = \sin(\pi x/2)$ and the high frequency initial condition $u_0(x) = \sin(2\pi\xi x)$, keeping the ratios $N_t : p_t$ and $N_x : p_x$ fixed at $2^{10} : 1$ while solving over the fixed domain $[-2, 2] \times [0, 4]$. We start with a grid of size $(N_x, N_t) = (2^{10}, 2^{10})$ and (i) double both dimensions at each step, (ii) double N_t, leaving N_x fixed, or (iii) double N_x, leaving N_t fixed; results for these cases are recorded in Tables 6.18, 6.19, and 6.20, respectively. The results for the first two cases are similar to those for explicit MGRIT, though the results of Table 6.18 show a nearly sixfold increase in the time-to-solution from the smallest to largest test cases, compared to times approximately doubling in the explicit case. This is likely due to the

					Original			Oscillatory		
Trial	$\log_2(N_x)$	$\log_2(N_t)$	$\log_2(p_x)$	$\log_2(p_t)$	ξ	Iter	Time	ξ	Iter	Time
1	10	10	0	0	1/4	21	240.45	16	23	276.54
2	10	11	0	1	1/4	25	306.88	16	25	328.86
3	10	12	0	2	1/4	13	208.58	16	15	250.45
4	10	13	0	3	1/4	10	177.35	16	12	217.68
5	10	14	0	4	1/4	9	166.21	16	10	191.35
6	10	15	0	5	1/4	7	139.22	16	8	162.35
7	10	16	0	6	1/4	5	112.63	16	7	148.56
8	10	17	0	7	1/4	5	114.46	16	5	120.30
9	10	18	0	8	1/4	4	103.10	16	4	107.93

Table 6.19: Weak scaling for implicit MGRIT with $u_0(x) = \sin(2\pi\xi x)$ and fixed N_x.

					Original			Oscillatory		
Trial	$\log_2(N_x)$	$\log_2(N_t)$	$\log_2(p_x)$	$\log_2(p_t)$	ξ	Iter	Time	ξ	Iter	Time
1	10	10	0	0	1/4	21	258.54	16	23	276.77
2	11	10	1	0	1/4	22	450.75	32	22	451.55
3	12	10	2	0	1/4	24	592.95	64	24	591.35
4	13	10	3	0	1/4	24	640.04	128	24	636.20
5	14	10	4	0	1/4	25	713.46	256	25	709.57
6	15	10	5	0	1/4	25	769.92	512	25	766.19
7	16	10	6	0	1/4	25	827.91	1024	25	823.27
8	17	10	7	0	1/4	24	872.87	2048	24	867.65
9	18	10	8	0	1/4	24	959.04	4096	24	953.32

Table 6.20: Weak scaling for implicit MGRIT with $u_0(x) = \sin(2\pi\xi x)$ and fixed N_t.

fact that the exact cyclic reduction linear solve used in implicit MGRIT has less potential for spatial parallelism compared to the matrix-vector product required for explicit MGRIT. The third case, unique to the implicit timestepping context, shows that increasing N_x while N_t remains fixed results in a nearly constant iteration count and an increasing time to solution. Considering the results for all three cases, it appears that the growth in iteration count due to increasing problem size is primarily a result of increasing N_t while maintaining a fixed ratio for $\Delta t : \Delta x$.

Conclusion

Part I

Nonlinear preconditioning strategies are an effective way to improve the convergence of iterative solvers for nonlinear systems and nonlinear optimization problems. In particular, when the problem formulation naturally suggests a fixed-point iteration that is more effective than steepest descent, such iterations can be greatly accelerated through use as nonlinear preconditioners. Nonlinearly preconditioned methods can be based on nonlinear left-preconditioning, nonlinear acceleration strategies, or by drawing comparisons to linear preconditioning for iterations solving linear systems or quadratic optimization problems.

In part I of this thesis we developed NPQN methods based on the L-Broyden and L-BFGS update equations and extended the NPNGMRES, NPNCG and NPQN nonlinearly preconditioned optimization methods to the matrix manifold setting, which can be useful for solving problems where the unknowns have some constraints, such as underlying symmetry or orthogonal structure. These iterations were applied to the problems of computing two popular tensor decompositions: the CP decomposition and the Tucker HOSVD. These decompositions remain commonly used tools in data compression and multilinear statistical analysis, hence improved computational algorithms will continue to be in demand.

Numerical results provide evidence that: (i) manifold NPNGMRES approximating the Hessian by a difference of gradients and NPNCG using the Polak–Ribière or Hestenes-Stiefel β formulas significantly outperformed HOOI, manifold NCG, manifold L-BFGS, and a tCG based manifold trust region algorithm for the Tucker HOSVD; and (ii) NPQN methods are effectively combined with CP-ALS iteration for the CP decomposition, being much faster than the individual QN method or fixed point iteration for difficult problems or when high accuracy is required. Furthermore, our results show that the proposed NPQN methods may significantly outperform NPNCG and NPNGMRES, being up to 50% faster is some cases. These results strongly suggest that nonlinearly preconditioned methods are leading contenders for efficient tensor decomposition algorithms.

There are a number of directions to carry out future work based on these results. At this point we have established the effectiveness of nonlinearly preconditioned versions of the popular NCG, NGMRES, L-BFGS and L-Broyden methods in Euclidean space and on Grassmann matrix manifolds. These methods can be applied to other systems of equations or optimization problems for matrix or tensor problems that have associated fixed point iterations that are more effective than steepest descent. We can also consider the development of preconditioned versions of other algorithms, such as those based on trust region strategies.

Part II

Parallel-in-time integration methods are a novel, if somewhat counter-intuitive, way to increase the amount of concurrency possible when solving a system of equations that evolve in time, and in the future such methods may be necessary to obtain further clock time speedups and to make use of the most recent massively parallel computing systems, especially with the continuing push towards exascale computing. However, in order for this to become a reality substantial improvements in the performance of parallel-in-time methods for hyperbolic and advection dominated problems are necessary.

In part II of this thesis we explored how the addition of spatial coarsening can affect the convergence of the MGRIT parallel-in-time method applied to the linear advection and inviscid Burgers equations in one spatial dimension. When using explicit time-stepping methods spatial coarsening is necessary to ensure the the CFL stability condition is satisfied on all levels of the grid hierarchy, and it is beneficial for both explicit and implicit methods as it produces smaller coarse-grid problems, and hence cheaper multigrid cycles. However, the introduction of spatial coarsening can result in a deteriorated convergence rate, and small local wave speeds can result in extremely poor convergence due to weak spatial connections.

We identified two potential ways of improving convergence when spatial coarsening is used: one intrusive and one non-intrusive. The intrusive method involves the introduction of spatial relaxations on the temporally coarsened intermediate grids, which brings MGRIT closer to a full space-time multigrid method with alternating temporal and spatial relaxation and coarsening. While effective in improving the convergence rate and handling near zero wave speeds, this method requires algorithms to handle the spatial relaxations, which may be difficult to implement (it was nontrivial even for the variable coefficient linear advection case), and hence it may not be easily combined with existing time-steppers, which is one of the key strengths of MGRIT. The non-intrusive approach is an adaptive spatial

coarsening strategy that prevents coarsening in regions where the local Courant number is small. This method, while not addressing the deterioration generally observed when spatial coarsening is included, is successful in ensuring reasonable convergence rates in cases where the wave speed is near or equal to zero at different points in the spatial domain. Iteration counts are obtained that show reasonable scalability as a function of problem size. To our best knowledge, we obtain the first convergent parallel-in-time method for the inviscid Burgers equation, and solutions with shocks do not exhibit convergence deterioration. Parallel results on up to 131072 cores illustrate robustness and scalability of the approach for very large problem sizes, and its potential to achieve run-time speedups when spatial parallelism alone saturates. Weak scaling results show that the scalability is robust for solutions with oscillations on the scale of the grid spacing.

As parallel-in-time integration is a developing field, there are many different directions to be explored moving forward. Perhaps the most pressing issue related to the work presented here is the lack of multigrid optimality: there is growth in iteration counts both with problem size and with the number of levels in the multigrid hierarchy. Reducing or preventing this growth would make MGRIT substantially more competitive and even greater parallel speedups would be possible. A second area is improving the Galerkin type coarse-grid operators for Burgers' equation, or for problems handled via a FAS approach, so that revisiting the fine grid is avoided. As the adaptive coarsening strategy is extensible, in principle, to both 2D and 3D, a third possibility would be to implement adaptive spatial coarsening for problems in two or more spatial dimensions and/or for systems of hyperbolic equations.

References

[1] hypre: Scalable linear solvers and multigrid methods. http://llnl.gov/casc/hypre.

[2] XBraid: Parallel multigrid in time. http://llnl.gov/casc/xbraid.

[3] Pierre-Antoine Absil, Christopher G. Baker, and Kyle A. Gallivan. Trust-region methods on Riemannian manifolds. *Foundations of Computational Mathematics*, 7(3):303–330, 2007.

[4] Pierre-Antoine Absil, Robert Mahony, and Rodolphe Sepulchre. *Optimization Algorithms on Matrix Manifolds*. Princeton University Press, 2009.

[5] Evrim Acar, Daniel M. Dunlavy, and Tamara G. Kolda. A scalable optimization approach for fitting canonical tensor decompositions. *Journal of Chemometrics*, 25(2):67–86, 2011.

[6] Donald G. Anderson. Iterative procedures for nonlinear integral equations. *Journal of the ACM*, 12(4):547–560, 1965.

[7] Brett W. Bader and Tamara G. Kolda. Algorithm 862: MATLAB tensor classes for fast algorithm prototyping. *ACM Transactions on Mathematical Software*, 32(4):635–653, December 2006.

[8] Brett W. Bader, Tamara G. Kolda, et al. MATLAB tensor toolbox version 2.5. Available online, January 2012.

[9] Jonas Ballani and Lars Grasedyck. A projection method to solve linear systems in tensor format. *Numerical Linear Algebra with Applications*, 20(1):27–43, 2013.

[10] Richard Bartels and James W. Daniel. A conjugate gradient approach to nonlinear elliptic boundary value problems in irregular regions. In G.A. Watson, editor, *Conference on the Numerical Solution of Differential Equations*, volume 363 of *Lecture Notes in Mathematics*, pages 1–11. Springer Berlin Heidelberg, 1974.

[11] Maurice S. Bartlett. An inverse matrix adjustment arising in discriminant analysis. *The Annals of Mathematical Statistics*, 22(1):107–111, 1951.

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

[13] Nicolas Boumal. Riemannian trust regions with finite-difference Hessian approximations are globally convergent. In *Geometric Science of Information*, pages 467–475. Springer, 2015.

[14] Nicolas Boumal, Bamdev Mishra, Pierre-Antoine Absil, and Rodolphe Sepulchre. Manopt, a Matlab toolbox for optimization on manifolds. *Journal of Machine Learning Research*, 15:1455–1459, 2014.

[15] Achi Brandt. Multi-level adaptive solutions to boundary-value problems. *Mathematics of Computation*, 31(138):333–390, 1977.

[16] William L. Briggs, Steve F. McCormick, et al. *A Multigrid Tutorial*. Society for Industrial and Applied Mathematics, 2000.

[17] Rasmus Bro. *Multi-way Analysis in the Food Industry: Models, Algorithms, and Applications*. PhD thesis, Københavns Universitet, 1998.

[18] Peter R. Brune, Matthew G. Knepley, Barry F. Smith, and Xuemin Tu. Composing scalable nonlinear algebraic solvers. *SIAM Review*, 57(4):535–565, 2015.

[19] Albert G. Buckley. Extending the relationship between the conjugate gradient and BFGS algorithms. *Mathematical Programming*, 15(1):343–348, 1978.

[20] Richard H. Byrd, Jorge Nocedal, and Robert B. Schnabel. Representations of quasi-Newton matrices and their use in limited memory methods. *Mathematical Programming*, 63(1):129–156, 1994.

[21] J. Douglas Carroll and Jih-Jie Chang. Analysis of individual differences in multidimensional scaling via an N-way generalization of "Eckart-Young" decomposition. *Psychometrika*, 35(3):283–319, 1970.

[22] Philippe Chartier and Bernard Philippe. A parallel shooting technique for solving dissipative ODE's. *Computing*, 51(3-4):209–236, 1993.

[23] Bilian Chen, Zhening Li, and Shuzhong Zhang. On optimal low rank Tucker approximation for tensors: the case for an adjustable core size. *Journal of Global Optimization*, pages 1–22, 2014.

[24] Feng Chen, Jan S. Hesthaven, and Xueyu Zhu. On the use of reduced basis methods to accelerate and stabilize the parareal method. In *Reduced Order Methods for Modeling and Computational Reduction*, pages 187–214. Springer, 2014.

[25] Yanqing Chen, Timothy A. Davis, William W. Hager, and Sivasankaran Rajamanickam. Algorithm 887: CHOLMOD, supernodal sparse Cholesky factorization and update/downdate. *ACM Transactions on Mathematical Software*, 35(3):22:1–22:14, October 2008.

[26] Andrzej Cichocki, Danilo Mandic, Lieven De Lathauwer, Guoxu Zhou, Qibin Zhao, Cesar Caiafa, and Anh-Huy Phan. Tensor decompositions for signal processing applications: From two-way to multiway component analysis. *Signal Processing Magazine, IEEE*, 32(2):145–163, March 2015.

[27] Paul Concus, Gene H. Golub, and Dianne P. O'Leary. Numerical solution of nonlinear elliptic partial differential equations by a generalized conjugate gradient method. *Computing*, 19(4):321–339, 1978.

[28] Xiaoying Dai and Yvon Maday. Stable parareal in time method for first-and second-order hyperbolic systems. *SIAM Journal on Scientific Computing*, 35(1):A52–A78, 2013.

[29] Timothy A. Davis. Algorithm 832: UMFPACK V4.3—an unsymmetric-pattern multifrontal method. *ACM Transactions on Mathematical Software*, 30(2):196–199, June 2004.

[30] Lieven De Lathauwer, Bart De Moor, and Joos Vandewalle. A multilinear singular value decomposition. *SIAM Journal on Matrix Analysis and Applications*, 21(4):1253–1278, 2000.

[31] Lieven De Lathauwer, Bart De Moor, and Joos Vandewalle. On the best rank-1 and rank-$(r_1, r_2, \ldots, r_n)$ approximation of higher-order tensors. *SIAM Journal on Matrix Analysis and Applications*, 21(4):1324–1342, 2000.

[32] Hans De Sterck. A nonlinear GMRES optimization algorithm for canonical tensor decomposition. *SIAM Journal on Scientific Computing*, 34(3):A1351–A1379, 2012.

[33] Hans De Sterck. Steepest descent preconditioning for nonlinear GMRES optimization. *Numerical Linear Algebra with Applications*, 20(3):453–471, 2013.

[34] Hans De Sterck and Manda Winlaw. A nonlinearly preconditioned conjugate gradient algorithm for rank-r canonical tensor approximation. *Numerical Linear Algebra with Applications*, 2014.

[35] John E. Dennis, Jr. and Robert B. Schnabel. *Numerical Methods for Unconstrained Optimization and Nonlinear Equations*, volume 16 of *Classics in Applied Mathematics*. Society for Industrial & Applied Mathematics, 1996.

[36] Veselin Dobrev, Tzanio Kolev, N. Anders Petersson, and Jacob Schroder. Two-level convergence theory for parallel time integration with multigrid. *SIAM Journal on Scientific Computing, LLNL-JRNL*, 692418, 2016.

[37] Daniel M. Dunlavy, Tamara G. Kolda, and Evrim Acar. Poblano v1.0: A MATLAB toolbox for gradient-based optimization. Technical Report SAND2010-1422, Sandia National Laboratories, Albuquerque, NM and Livermore, CA, March 2010.

[38] Carl Eckart and Gale Young. The approximation of one matrix by another of lower rank. *Psychometrika*, 1(3):211–218, 1936.

[39] Alan Edelman, Tomás A. Arias, and Steven T. Smith. The geometry of algorithms with orthogonality constraints. *SIAM Journal on Matrix Analysis and Applications*, 20(2):303–353, 1998.

[40] Lars Eldén and Berkant Savas. A Newton-Grassmann method for computing the best multilinear rank-(r_1, r_2, r_3) approximation of a tensor. *SIAM Journal on Matrix Analysis and Applications*, 31(2):248–271, 2009.

[41] Matthew Emmett and Michael Minion. Toward an efficient parallel in time method for partial differential equations. *Communications in Applied Mathematics and Computational Science*, 7(1):105–132, 2012.

[42] Robert D. Falgout, Stephanie Friedhoff, Tzanio V. Kolev, Scott P. MacLachlan, and Jacob B. Schroder. Parallel time integration with multigrid. *SIAM Journal on Scientific Computing*, 36(6):C635–C661, 2014.

[43] Robert D. Falgout, Aaron Katz, Tzanio V. Kolev, Jacob B. Schroder, Andrew Wissink, and Ulrike M. Yang. Parallel time integration with multigrid reduction for a compressible fluid dynamics application. 2015. LLNL-JRNL-663416.

[44] Robert D. Falgout, Thomas A. Manteuffel, Benjamin O'Neill, and Jacob B. Schroder. Multigrid reduction in time for nonlinear parabolic problems: A case study. *SIAM Journal on Scientific Computing (to appear)*, 2016. LLNL-JRNL-692258.

[45] Haw-ren Fang and Yousef Saad. Two classes of multisecant methods for nonlinear acceleration. *Numerical Linear Algebra with Applications*, 16(3):197–221, 2009.

[46] Stephanie Friedhoff and Scott P. MacLachlan. A generalized predictive analysis tool for multigrid methods. *Numerical Linear Algebra with Applications*, 22(4):618–647, 2015.

[47] Stephanie Friedhoff, Scott P. MacLachlan, and Christoph Börgers. Local Fourier analysis of space-time relaxation and multigrid schemes. *SIAM Journal on Scientific Computing*, 35(5):S250–S276, 2013.

[48] Martin J. Gander. 50 years of time parallel time integration. In *Multiple Shooting and Time Domain Decomposition Methods*, pages 69–113. Springer, 2015.

[49] Martin J. Gander and Madalina Petcu. Analysis of a Krylov subspace enhanced parareal algorithm for linear problems. In *ESAIM: Proceedings*, volume 25, pages 114–129. EDP Sciences, 2008.

[50] Martin J. Gander and Stefan Vandewalle. Analysis of the parareal time-parallel time-integration method. *SIAM Journal on Scientific Computing*, 29(2):556–578, 2007.

[51] Stefan Güttel. A parallel overlapping time-domain decomposition method for ODEs. In *Domain Decomposition Methods in Science and Engineering XX*, pages 459–466. Springer, 2013.

[52] Wolfgang Hackbusch. *Tensor Spaces and Numerical Tensor Calculus*, volume 42 of *Springer Series in Computational Mathematics*. Springer, 2012.

[53] William W. Hager and Hongchao Zhang. A new conjugate gradient method with guaranteed descent and an efficient line search. *SIAM Journal on Optimization*, 16(1):170–192, 2005.

[54] William W. Hager and Hongchao Zhang. A survey of nonlinear conjugate gradient methods. *Pacific journal of Optimization*, 2(1):35–58, 2006.

[55] Richard A. Harshman. Foundations of the PARAFAC procedure: Models and conditions for an "explanatory" multi-modal factor analysis. *UCLA Working Papers in Phonetics*, 16:1–84, 1970.

[56] Magnus R. Hestenes and Eduard Stiefel. *Methods of Conjugate Gradients for Solving Linear Systems*, volume 49. National Bureau of Standards Washington, DC, 1952.

[57] L. Hogben. *Handbook of Linear Algebra, Second Edition*. Discrete Mathematics and Its Applications. CRC Press, 2016.

[58] Graham Horton. The time-parallel multigrid method. *Communications in Applied Numerical Methods*, 8(9):585–595, 1992.

[59] Willem Hundsdorfer and Jan G. Verwer. *Numerical Solution of Time-Dependent Advection-Diffusion-Reaction Equations*, volume 33 of *Springer Series in Computational Mathematics*. Springer Science & Business Media, 2013.

[60] Mariya Ishteva. *Numerical methods for the best low multilinear rank approximation of higher-order tensors*. PhD thesis, Department of Electrical Engineering, Katholieke Universiteit Leuven, 2009.

[61] Mariya Ishteva, Pierre-Antoine Absil, Sabine Van Huffel, and Lieven De Lathauwer. Best low multilinear rank approximation of higher-order tensors, based on the Riemannian trust-region scheme. *SIAM Journal on Matrix Analysis and Applications*, 32(1):115–135, 2011.

[62] Mariya Ishteva, Lieven De Lathauwer, Pierre-Antoine Absil, and Sabine Van Huffel. Differential-geometric Newton method for the best rank-(r_1, r_2, r_3) approximation of tensors. *Numerical Algorithms*, 51(2):179–194, 2009.

[63] Carl T. Kelley. *Iterative Methods for Optimization*. Society for Industrial and Applied Mathematics, 1999.

[64] Tamara G. Kolda and Brett W. Bader. Tensor decompositions and applications. *SIAM Review*, 51(3):455–500, 2009.

[65] Tamara G. Kolda and Jimeng Sun. Scalable tensor decompositions for multi-aspect data mining. In *Eighth IEEE International Conference on Data Mining*, pages 363–372. IEEE, 2008.

[66] Matthieu Lecouvez, Robert D Falgout, Carol S Woodward, and Philip Top. A parallel multigrid reduction in time method for power systems. In *Power and Energy Society General Meeting (PESGM), 2016*, pages 1–5. IEEE, 2016.

[67] Yann Lecun and Corinna Cortes. The MNIST database of handwritten digits.

[68] Randall J. LeVeque. *Numerical Methods for Conservation Laws*. Springer, 1992.

[69] Randall J. LeVeque. *Finite Volume Methods for Hyperbolic Problems*, volume 31 of *Cambridge Texts in Applied Mathematics*. Cambridge University Press, 2002.

[70] Lek-Heng Lim and Pierre Comon. Nonnegative approximations of nonnegative tensors. *Journal of Chemometrics*, 23(7-8):432–441, 2009.

[71] Jacques-Louis Lions, Yvon Maday, and Gabriel Turinici. Résolution d'EDP par un schéma en temps «pararéel». *Comptes Rendus de l'Académie des Sciences-Series I-Mathematics*, 332(7):661–668, 2001.

[72] David G. Luenberger and Yinyu Ye. *Linear and Nonlinear Programming*, volume 228 of *International Series in Operations Research and Management Science*. Springer, 2015.

[73] Hans D. Mittelmann. On the efficient solution of nonlinear finite element equations I. *Numerische Mathematik*, 35(3):277–291, 1980.

[74] E. Hastings Moore. On the reciprocal of the general algebraic matrix. *Bulletin of the American Mathematical Society*, 26:394–395, 1920.

[75] Jorge J. Moré and David J. Thuente. Line search algorithms with guaranteed sufficient decrease. *ACM Transactions on Mathematical Software*, 20(3):286–307, 1994.

[76] J. Larry Nazareth. A relationship between the BFGS and conjugate gradient algorithms and its implications for new algorithms. *SIAM Journal on Numerical Analysis*, 16(5):794–800, 1979.

[77] Jorge Nocedal and Stephen Wright. *Numerical Optimization*. Springer Series in Operations Research and Financial Engineering. Springer New York, 2006.

[78] Roger Penrose. A generalized inverse for matrices. *Mathematical Proceedings of the Cambridge Philosophical Society*, 51(3):406–413, 1955.

[79] Elijah Polak and Gerard Ribière. Note sur la convergence de méthodes de directions conjuguées. *ESAIM: Mathematical Modelling and Numerical Analysis-Modélisation Mathématique Et Analyse Numérique*, 3(R1):35–43, 1969.

[80] Agostino Prastaro. *Geometry of PDEs and Mechanics*. World Scientific, 1996.

[81] Thorsten Rohwedder and Reinhold Schneider. An analysis for the DIIS acceleration method used in quantum chemistry calculations. *Journal of Mathematical Chemistry*, 49(9):1889–1914, 2011.

[82] John W. Ruge and Klaus Stüben. Algebraic multigrid. *Multigrid Methods*, 3(13):73–130, 1987.

[83] Daniel Ruprecht. Wave propagation characteristics of parareal. *arXiv preprint arXiv:1701.01359*, 2017.

[84] Daniel Ruprecht and Rolf Krause. Explicit parallel-in-time integration of a linear acoustic-advection system. *Computers & Fluids*, 59:72–83, 2012.

[85] Yousef Saad. *Iterative Methods for Sparse Linear Systems*. Society for Industrial and Applied Mathematics, Philadelphia, PA, USA, 2nd edition, 2003.

[86] Yousef Saad and Martin 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.

[87] Ferdinando S. Samaria and Andy C. Harter. Parameterisation of a stochastic model for human face identification. In *Proceedings of the Second IEEE Workshop on Applications of Computer Vision*, pages 138–142. IEEE, 1994.

[88] Berkant Savas and Lars Eldén. Handwritten digit classification using higher order singular value decomposition. *Pattern Recognition*, 40(3):993–1003, 2007.

[89] Berkant Savas and Lek-Heng Lim. Quasi-Newton methods on Grassmannians and multilinear approximations of tensors. *SIAM Journal on Scientific Computing*, 32(6):3352–3393, 2010.

[90] Jack Sherman and Winifred J. Morrison. Adjustment of an inverse matrix corresponding to a change in one element of a given matrix. *The Annals of Mathematical Statistics*, 21(1):124–127, 1950.

[91] Jonathan R. Shewchuk et al. An introduction to the conjugate gradient method without the agonizing pain, 1994.

[92] Age Smilde, Rasmus Bro, and Paul Geladi. *Multi-way Analysis: Applications in the Chemical Sciences*. John Wiley & Sons, 2005.

[93] Nguyen Thanh Son. A real time procedure for affinely dependent parametric model order reduction using interpolation on Grassmann manifolds. *International Journal for Numerical Methods in Engineering*, 93(8):818–833, 2013.

[94] Ben Southworth, Tom Manteuffel, Steve McCormick, Steffen Munzenmaier, and John Ruge. Reduction-based algebraic multigrid for upwind discretizations. *arXiv preprint arXiv:1704.05001*, 2017.

[95] Shlomo Ta'asan and Hong Zhang. On the multigrid waveform relaxation method. *SIAM Journal on Scientific Computing*, 16(5):1092–1104, 1995.

[96] Shlomo Ta'asan and Hong Zhang. Fourier-Laplace analysis of the multigrid waveform relaxation method for hyperbolic equations. *BIT Numerical Mathematics*, 36(4):831–841, 1996.

[97] Giorgio Tomasi and Rasmus Bro. A comparison of algorithms for fitting the PARAFAC model. *Computational Statistics & Data Analysis*, 50(7):1700–1734, 2006.

[98] Ulrich Trottenberg, Cornelius W. Oosterlee, and Anton Schüller. *Multigrid*. Academic Press, 2000.

[99] Ledyard R. Tucker. Implications of factor analysis of three-way matrices for measurement of change. *Problems in Measuring Change*, pages 122–137, 1963.

[100] Ledyard R. Tucker. Some mathematical notes on three-mode factor analysis. *Psychometrika*, 31(3):279–311, 1966.

[101] Stefan Vandewalle and Eric Van de Velde. Space-time concurrent multigrid waveform relaxation. *Annals of Numerical Mathematics*, 1:347–363, 1994.

[102] Nick Vannieuwenhoven, Raf Vandebril, and Karl Meerbergen. On the truncated multilinear singular value decomposition. *TW Reports*, TW589:1–34, 2011.

[103] Nick Vannieuwenhoven, Raf Vandebril, and Karl Meerbergen. A new truncation strategy for the higher-order singular value decomposition. *SIAM Journal on Scientific Computing*, 34(2):A1027–A1052, 2012.

[104] Homer F. Walker and Peng Ni. Anderson acceleration for fixed-point iterations. *SIAM Journal on Numerical Analysis*, 49(4):1715–1735, 2011.

[105] Takumi Washio and Cornelis W. Oosterlee. Krylov subspace acceleration for nonlinear multigrid schemes. *Electronic Transactions on Numerical Analysis*, 6(271-290):3–1, 1997.

[106] Max A. Woodbury. Inverting modified matrices. *Memorandum Report*, 42:106, 1950.

[107] Yuto Yokota and Takashi Nodera. The L-BFGS method for nonlinear GMRES acceleration. Technical report, Keio University, Department of Mathematics, 2015.

[108] David M. Young. Iterative methods for solving partial difference equations of elliptic type. *Transactions of the American Mathematical Society*, 76(1):92–111, 1954.

[109] David M. Young. *Iterative Solution of Large Linear Systems*. Computer Science and Applied Mathematics. Academic Press, 1971.

[110] Miao Zhang and Chris Ding. Robust Tucker tensor decomposition for effective image representation. In *2013 IEEE International Conference on Computer Vision*, pages 2448–2455, Dec 2013.

Appendices

Appendix A

Supplementary Parallel Scaling Results

A.1 Explicit Time-stepping for Case A5

	$(\log_2(N_x), \log_2(N_t))$		
$\log_2(p_x)$	$(14, 15)$	$(15, 16)$	$(16, 17)$
2	846.33	—	—
3	441.81	1699.65	—
4	232.14	880.60	3419.02
5	127.19	456.45	1769.27
6	75.44	257.11	926.55
7	50.81	151.31	519.89
8	40.13	103.31	309.69
9	35.19	84.62	220.80
10	34.63	75.39	180.07
11	36.10	74.48	162.61
12	38.36	78.33	163.74

Table A.1: Strong scaling for serial explicit time-stepping with increasing amounts of spatial parallelism and no temporal parallelism for fixed problem size.

$\log_2$(pt)	$\log_2$(px)	2	3	4	5	6	7	8
3	iter	36	36	36	36	36	36	36
	time	2836.37	1691.60	1207.62	986.34	809.76	714.02	647.39
4	iter	36	36	—	—	—	36	—
	time	1484.32	880.73	—	—	—	365.42	—
5	iter	36	—	36	—	36	—	—
	time	818.17	—	336.45	—	210.60	—	—
6	iter	36	—	—	36	—	—	—
	time	440.94	—	—	132.10	—	—	—
7	iter	36	—	36	—	36	—	—
	time	231.56	—	91.40	—	55.70	—	—
8	iter	36	36	—	—	—	36	—
	time	137.10	90.40	—	—	—	26.92	—
9	iter	36	—	—	—	—	—	36
	time	78.75	—	—	—	—	—	15.08

Table A.2: **Original Size:** Strong scaling for explicit MGRIT, $(N_x, N_t) = (2^{14}, 2^{15})$.

$\log_2$(pt)	$\log_2$(px)	3	4	5	6	7	8	9
4	iter	42	41	41	42	41	42	41
	time	3455.59	2216.87	1493.09	1209.49	973.17	875.18	784.65
5	iter	41	41	—	—	—	41	—
	time	1860.44	1150.40	—	—	—	438.65	—
6	iter	41	—	42	—	42	—	—
	time	976.22	—	417.92	—	258.22	—	—
7	iter	42	—	—	41	—	—	—
	time	515.44	—	—	159.37	—	—	—
8	iter	41	—	41	—	41	—	—
	time	272.32	—	110.88	—	67.64	—	—
9	iter	42	42	—	—	—	42	—
	time	151.43	91.31	—	—	—	34.59	—
10	iter	41	—	—	—	—	—	—
	time	86.61	—	—	—	—	—	—

Table A.3: **Doubled Size:** Strong scaling for explicit MGRIT, $(N_x, N_t) = (2^{15}, 2^{16})$.

$\log_2$(pt)	$\log_2$(px)	4	5	6	7	8	9
5	iter	51	52	50	51	50	50
	time	4846.38	3999.70	3592.11	3410.69	3301.74	3500.35
6	iter	50	51	—	—	51	—
	time	2811.06	1526.86	—	—	1735.72	—
7	iter	50	—	50	51	—	—
	time	1271.31	—	526.61	409.13	—	—
8	iter	51	—	51	51	—	—
	time	666.09	—	278.37	211.18	—	—
9	iter	51	51	—	—	50	—
	time	350.51	215.74	—	—	90.34	—
10	iter	50	—	—	—	—	—
	time	191.07	—	—	—	—	—

Table A.4: **Quadrupled Size:** Strong scaling for explicit MGRIT, $(N_x, N_t) = (2^{16}, 2^{17})$.

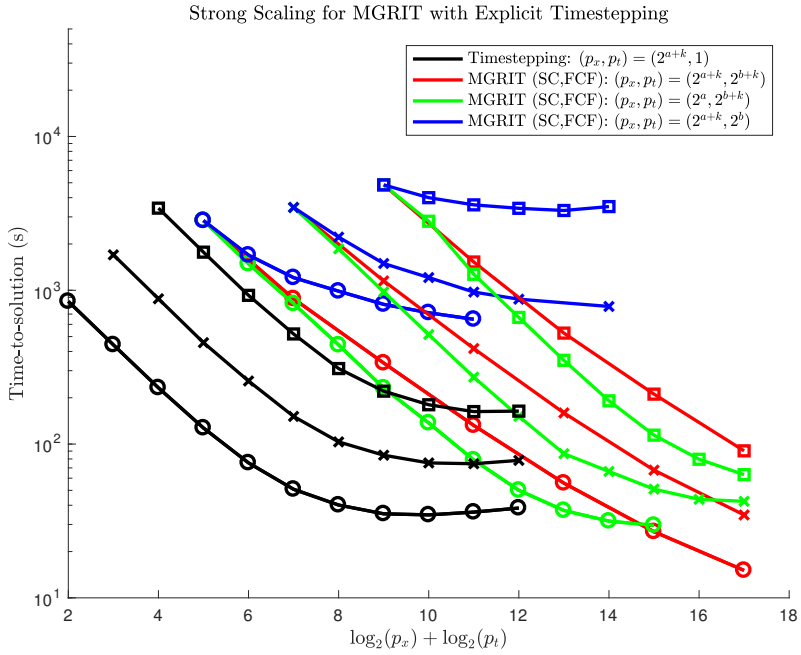

Figure A.1: Comparison of serial time-stepping with spatial parallelism to explicit MGRIT with different combinations of space-time parallelism for three different problem sizes on up to 131072 cores, $(N_x, N_t) = (2^n, 2^{n+1})$. These results correspond to Tables A.1–A.4. ○: $n = 14$, $a = 2$, $b = 3$. ×: $n = 15$, $a = 3$, $b = 4$. □: $n = 16$, $a = 4$, $b = 5$.

		(a,b,n)		
		$(2,3,14)$	$(3,4,15)$	$(4,5,16)$
(p_x,p_t)	$(2^{a+k},2^{b+k})$	2.30	2.15	1.78
	$(2^{a},2^{b+k})$	1.17	1.75	2.55

Table A.5: Best speedup achieved for explicit MGRIT strong scaling presented in Figure A.1, $(N_x, N_t) = (2^n, 2^{n+1})$.

					Original			Oscillatory		
Trial	$\log_2(N_x)$	$\log_2(N_t)$	$\log_2(p_x)$	$\log_2(p_t)$	ξ	Iter	Time	ξ	Iter	Time
1	10	11	0	1	1/4	30	184.86	16	30	184.98
2	11	12	1	2	1/4	31	223.12	32	31	222.95
3	12	13	2	3	1/4	33	245.39	64	33	245.68
4	13	14	3	4	1/4	35	291.70	128	35	291.61
5	14	15	4	5	1/4	36	339.98	256	36	339.53

Table A.6: Weak scaling for explicit MGRIT with $u_0(x) = \sin(2\pi\xi x)$.

					Original			Oscillatory		
Trial	$\log_2(N_x)$	$\log_2(N_t)$	$\log_2(p_x)$	$\log_2(p_t)$	ξ	Iter	Time	ξ	Iter	Time
1	10	11	0	1	1/4	30	184.63	16	30	185.09
2	10	12	0	2	1/4	14	111.16	16	15	117.20
3	10	13	0	3	1/4	13	110.60	16	14	116.89
4	10	14	0	4	1/4	10	94.61	16	11	101.02
5	10	15	0	5	1/4	8	83.49	16	9	90.12
6	10	16	0	6	1/4	6	72.14	16	6	72.20
7	10	17	0	7	1/4	7	82.17	16	7	82.09

Table A.7: Weak scaling for explicit MGRIT with $u_0(x) = \sin(2\pi\xi x)$ and fixed N_x.

A.2 Implicit Time-stepping for Case A5

	$(\log_2(N_x), \log_2(N_t))$
$\log_2(p_x)$	$(14, 14)$
2	801.46
3	536.29
4	415.51
5	371.04
6	350.20
7	353.65
8	369.76

Table A.8: Strong scaling results for serial implicit time-stepping with increasing amounts of spatial parallelism, $(N_x, N_t) = (2^{14}, 2^{14})$.

$\log_2$(pt)	$\log_2$(px)	2	3	4	5	6	7	8
2	iter	27	26	27	27	27	26	27
	time	5937.06	3991.70	3210.82	2825.08	2578.00	2357.61	2364.47
3	iter	27	26	—	—	—	26	—
	time	2964.59	1997.37	—	—	—	1184.73	—
4	iter	27	—	27	—	26	—	—
	time	1488.67	—	832.33	—	629.40	—	—
5	iter	27	—	—	27	—	—	—
	time	767.72	—	—	364.10	—	—	—
6	iter	26	—	26	—	27	—	—
	time	382.61	—	211.93	—	169.23	—	—
7	iter	27	27	—	—	—	27	—
	time	221.78	160.31	—	—	—	84.93	—
8	iter	27	—	—	—	—	—	27
	time	132.43	—	—	—	—	—	46.83

Table A.9: **No Spatial Coarsening:** Strong scaling for implicit MGRIT, $N_x = N_t = 2^{14}$.

$\log_2$(pt)	$\log_2$(px)	2	3	4	5	6	7	8
2	iter	40	40	41	40	40	40	40
	time	7329.11	5463.71	4666.86	4374.57	4068.72	3883.04	3824.27
3	iter	40	40	—	—	—	40	—
	time	3740.71	2786.03	—	—	—	1964.83	—
4	iter	41	—	40	—	40	—	—
	time	1996.01	—	1297.06	—	1054.42	—	—
5	iter	40	—	—	40	—	—	—
	time	1129.15	—	—	591.32	—	—	—
6	iter	40	—	40	—	40	—	—
	time	634.29	—	354.25	—	276.62	—	—
7	iter	40	40	—	—	—	40	—
	time	361.17	264.02	—	—	—	138.59	—
8	iter	40	—	—	—	—	—	40
	time	206.53	—	—	—	—	—	75.87

Table A.10: **Spatial Coarsening:** Strong scaling for implicit MGRIT, $N_x = N_t = 2^{14}$.

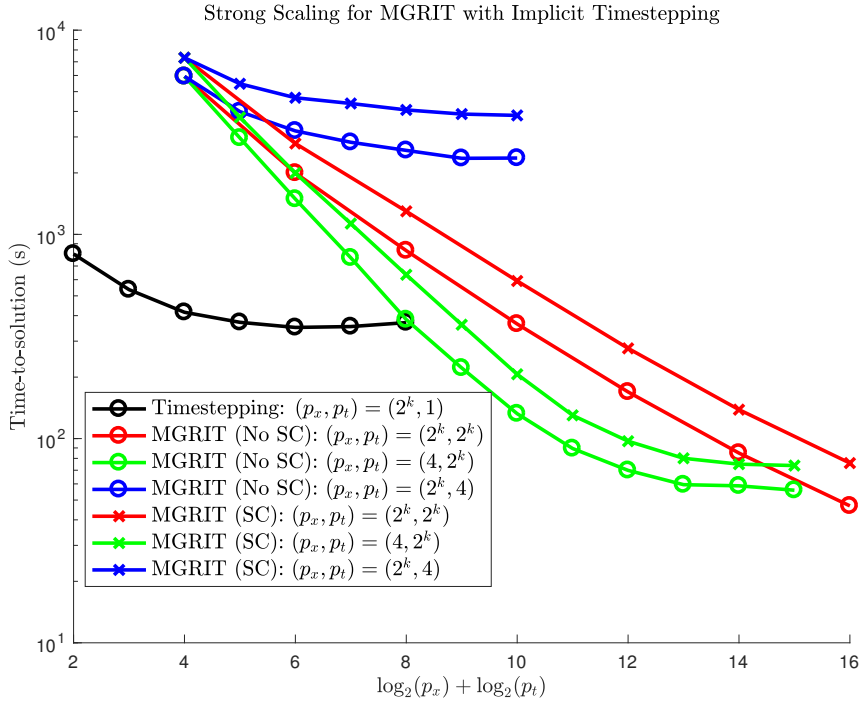

Figure A.2: Comparison of serial time-stepping with spatial parallelism to MGRIT with or without spatial coarsening for different combinations of space-time parallelism on up to 65536 cores, $(N_x, N_t) = (2^{14}, 2^{14})$. These results correspond to Tables A.8–A.10.

(p_x, p_t)		No SC	SC
	$(2^k, 2^k)$	7.48	4.62
	$(2^4, 2^k)$	6.27	4.75

Table A.11: Best speedup achieved for implicit time-stepping strong scaling presented in Figure A.2, $(N_x, N_t) = (2^{14}, 2^{14})$.

					Original			Oscillatory		
Trial	$\log_2(N_x)$	$\log_2(N_t)$	$\log_2(p_x)$	$\log_2(p_t)$	ξ	Iter	Time	ξ	Iter	Time
1	10	10	0	0	1/4	24	295.32	16	24	296.61
2	11	11	1	1	1/4	26	626.30	32	26	624.78
3	12	12	2	2	1/4	29	871.47	64	29	869.50
4	13	13	3	3	1/4	34	1124.93	128	33	1091.52
5	14	14	4	4	1/4	41	1309.98	256	41	1295.42

Table A.12: Weak scaling for implicit MGRIT with $u_0(x) = \sin(2\pi\xi x)$.

					Original			Oscillatory		
Trial	$\log_2(N_x)$	$\log_2(N_t)$	$\log_2(p_x)$	$\log_2(p_t)$	ξ	Iter	Time	ξ	Iter	Time
1	10	10	0	0	1/4	24	297.03	16	24	296.59
2	10	11	0	1	1/4	27	367.88	16	27	367.27
3	10	12	0	2	1/4	13	224.39	16	14	238.11
4	10	13	0	3	1/4	11	203.74	16	11	203.43
5	10	14	0	4	1/4	9	176.65	16	9	176.33
6	10	15	0	5	1/4	8	162.44	16	8	162.16
7	10	16	0	6	1/4	6	133.52	16	7	148.48
8	10	17	0	7	1/4	5	120.45	16	5	120.26
9	10	18	0	8	1/4	4	107.93	16	4	107.84

Table A.13: Weak scaling for implicit MGRIT with $u_0(x) = \sin(2\pi\xi x)$ and fixed N_x.

					Original			Oscillatory		
Trial	$\log_2(N_x)$	$\log_2(N_t)$	$\log_2(p_x)$	$\log_2(p_t)$	ξ	Iter	Time	ξ	Iter	Time
1	10	10	0	0	1/4	24	297.05	16	24	294.81
2	11	10	1	0	1/4	23	465.06	32	23	465.20
3	12	10	2	0	1/4	23	559.13	64	23	558.23
4	13	10	3	0	1/4	23	594.61	128	23	593.32
5	14	10	4	0	1/4	23	642.39	256	23	641.37
6	15	10	5	0	1/4	23	682.46	512	23	679.69
7	16	10	6	0	1/4	22	696.81	1024	23	721.51
8	17	10	7	0	1/4	22	750.16	2048	23	775.58
9	18	10	8	0	1/4	22	816.76	4096	22	812.83

Table A.14: Weak scaling for implicit MGRIT with $u_0(x) = \sin(2\pi\xi x)$ and fixed N_t.