TEL AVIV UNIVERSITY אוניברסיטת תל-אביב
Raymond and Beverly Sackler Faculty of Exact Sciences
School of Computer Science

Advanced Algorithmic Techniques in Numerical Linear Algebra: Hybridization and Randomization

Thesis submitted for the degree of Doctor of Philosophy
by

Haim Avron

This work was carried out under the supervision of
Prof. Sivan Toledo

Submitted to the Senate of Tel Aviv University
July 2010

To my dear wife and wonderful son.

Acknowledgments

First and foremost, I would like to thank my advisor, Professor Sivan Toledo. Sivan is a true mentor. I do not think one can hope for a better advisor. Working with Sivan was a truly pleasurable learning experience. I appreciate the time he spent working with me, and for teaching me how research is done in academe.

I wish to thank Professor Daniel Cohen-Or for the opportunity to work with his research group. I would also like to thank Dr. Anshul Gupta for mentoring me during my IBM summer internship in 2008. I learned a lot about research on Numerical Linear Algebra while working with Anshul. It is also a pleasure to thank my other co-authors and collaborators (both of papers included in this thesis and papers not included in this thesis): Esmond Ng, Gil Shklarski, Andrei Sharf, Chen Grief, Doron Chen, Petar Maymounkov and Prabhanjan Kambadur. I would like to especially thank Mark Tygert for very helpful discussions, ideas and insights for my two papers on randomized numerical linear algebra.

Thanks to my friends and lab partners over the years, Gil Shklarski, Aviad Zuc, Andrei Sharf, Tuli Uchitel, Michal Spivak, Nir Shasha, Avi Ben Aroya, Sharon Buckner, Guy Karlibach and Alex Druinsky. I wish to thank them for their friendship and numerous interesting discussions.

I wish to thank my dear parents for their love, encouragements and on-going support over the years. Without them pushing me from a young age, even though I did not want to be pushed, I would never have gotten to where I got.

Last but not least, I wish to thank my dearest wife Yifat, for her love and friendship, and for keeping up with me while I chase my dreams. This thesis is dedicated to her.

Abstract

The main theme of this thesis is the use of hybridization and randomization to develop new algorithms for solving important problems in numerical linear algebra. The importance of numerical linear algebra cannot be overstated. Numerical linear algebra is ubiquitous in scientific computing. For example, modeling real-world changing conditions, such as weather, air flow around a plane, automobile body deformation in a crash almost always requires solving linear systems or computing eigenvalues. In the last few decades there has been considerable progress in the development of general purpose linear solvers. Nevertheless, concurrently the the number and scale scientific applications increased dramatically, and today, more then ever, many research and industrial centers are in need of widely-applicable efficient linear solvers. This thesis explores two exciting methodologies, hybridization and randomization, which seem promising in the effort to push the boundary of numerical linear algebra.

We begin by exploring the use of hybridization. There are two methodologies for solving both sparse and dense systems of linear equations: direct and iterative. Direct solvers factor the coefficient matrix into a product of simpler matrices whose inverse is easy to apply. Iterative solvers start with an approximate solution and iteratively refine it. Each of the two methodologies has advantages and disadvantages. Direct methods tend to be generic, robust, predictable and efficient. But their scalability is limited: they may run out of memory, and be too slow for very large matrices. Iterative methods scale much better, but are usually more fragile and slower than direct methods. These difficulties are usually overcome by using a preconditioner, but this reduces generality. But preconditionning also opens the door for hybridization: a direct method can be used to form the preconditioner. By embedding a direct method inside the iterative method we are able to enhance its robustness and efficiency without sacrificing too much generality.

In chapter 2 we present an hybrid linear solver for equations arising from a finite element discretization of scalar elliptic partial differential equations. Matrices arising from finite-element discretization of scalar elliptic partial differential equations have certain algebraic properties that can be utilized for constructing an effective linear solver. We present a study of these properties and a novel solver based on them. The solver uses new purely algebraic methods for approximating the element matrices by symmetric diagonally dominant ones. The resulting matrix is then approximated using standard combinatorial preconditioners, which are factored using a direct solver. When all the element matrices are approximable, which is often the case, the solver is provably efficient. When there are inapproximable elements, the solver identifies those elements, and does not sparsify their part in the model. As this portion becomes larger, this solver automatically behaves more like a direct solver. Experimental results show that on problems in which some of the element matrices are ill conditioned, this solver is more effective than an algebraic

multigrid solver, than an incomplete-factorization solver, and than a direct solver. This is a joint work with Doron Chen, Gil Shklarski and Sivan Toledo.

In chapter 3 we propose and analyze a new tool to help solve sparse linear least-squares problems $\min_x \|Ax - b\|_2$. Our method is based on a sparse QR factorization of a low-rank perturbation $\hat{A}$ of A. More precisely, we show that the R factor of $\hat{A}$ is an effective preconditioner for the least-squares problem $\min_x \|Ax - b\|_2$, when solved using LSQR. We propose applications for the new technique. When A is rank deficient we can add rows to construct a well-conditioned preconditioner. When A is sparse except for a few dense rows we can drop these dense rows from A to obtain $\hat{A}$ to avoid fill in R . Another application is solving an updated or downdated problem. If R is a good preconditioner for the original problem A, it is a good preconditioner for the updated/downdated problem $\hat{A}$. We can also solve what-if scenarios, where we want to find the solution if a column of the original matrix is changed/removed. We present a spectral theory that analyzes the generalized spectrum of the pencil (A^*A, R^*R) and applications of this theory. This is a joint work with Esmond Ng and Sivan Toledo.

Two applications of the row perturbation techniques are explored in depth in chapters 4 and 5. Chapter 4 presents a solver for rank-deficient overdetermined least squares systems. Our solver forms a well-conditioned preconditioner using the techniques presented in chapter 3 . Numerical experiments establish the effectiveness of our solver.

Chapter 5 exploits another technique from chapter 3 to address a problem in computer graphics. The chapter presents a novel method for reconstructing a piecewise smooth surface from noisy 3D scanner data. The method forms an ℓ_1 minimization problem (which is, in our case, a second-order cone problem) which is solved using a log-barrier interior point method. This requires the solution of many least-squares problems with several dense rows. The technique of row-removal from chapter 3 is used to address the dense rows. The significance of the results lie in the realm of computer graphics, but they also demonstrate the utility of the results from chapter 3. This is a joint work with Andrei Sharf, Chen Greif and Daniel Cohen-Or.

Chapter 6 explores the use of indefinite incomplete factorization as preconditioners. Incomplete LDL^* factorizations sometimes produce an indefinite preconditioner even when the input matrix is Hermitian positive definite. MINRES and the most popular variant of CG cannot use such preconditioners; they require a positive definite preconditioner. A less known variant of CG, called ORTHOMIN, can be used with an indefinite preconditioner, but its performance has not been experimentally investigated. We experimentally evaluated ORTHOMIN, and discovered that it suffers from numerical problems. We propose a new variant of ORTHOMIN that uses selective orthogonalization to mitigate the numerical problems. We also present a new variant of MINRES which can be preconditioned using any non-singular Hermitian matrix as long as the original system is positive definite. Finally, we present extensive numerical experiments. The evaluation of these algorithms is important, as they allow the use of incomplete-factorization preconditioners for Hermitian positive definite systems, even when the preconditioner is indefinite, without resorting to a more expensive non-symmetric iterative Krylov-subspace solver. This is a joint work with Anshul Gupta and Sivan Toledo.

The second theme of this thesis is randomization. Randomization is arguably the most exciting and innovative idea to have hit linear algebra in a long time. Several such algorithms have been proposed and explored in the past decade. Some forms of

randomization have been used for decades in linear algebra. For example, the starting vectors in Lanczos algorithms are always random. But recent research led to new uses of randomization: random mixing and random sampling, which can be combined to form random projections. These ideas have been explored theoretically and have found use in some specialized applications, but they have had little influence so far on mainstream numerical linear algebra.

In chapter 7 we present an analysis of the convergence of randomized trace estimators. Starting at 1989, several algorithms have been proposed for estimating the trace of a matrix by $\frac{1}{M}\sum_{i=1}^{M} z_i^T A z_i$, where the z_i are random vectors; different estimators use different distributions for the z_is, all of which lead to $E(\frac{1}{M}\sum_{i=1}^{M} z_i^T A z_i) = \text{trace}(A)$. These algorithms are useful in applications in which there is no explicit representation of A but rather an efficient method compute $z^T A z$ given z. Existing results only analyze the variance of the different estimators. In contrast, we analyze the number of samples M required to guarantee that with probability at least $1-\delta$, the relative error in the estimate is at most ϵ. We argue that such bounds are much more useful in applications than the variance. We found that these bounds rank the estimators differently than the variance; this suggests that minimum-variance estimators may not be the best. We also make two additional contributions to this area. The first is a specialized bound for projection matrices, whose trace (rank) needs to be computed in electronic structure calculations. The second is a new estimator that uses less randomness than all the existing estimators. This is a joint work with Sivan Toledo.

Both themes, hybridization and randomization, are used in chapter 8. The limited impact of randomized algorithms in numerical linear algebra was the driving force behind the research in chapter 8, but it was the understanding that randomized algorithms should be fused with other ideas, like hybridization, that offered the solution. As already mentioned, no practical randomized linear solver has been demonstrated so far, although theoretical algorithms have been suggested. The performance of algorithms and codes on modern computers is a complex issue, one that cannot always be fully captured by complexity analysis. Performance depends on many issues and can really only be truly measured by developing and testing the software on different platforms. Furthermore, often the running time of randomized numerical linear algebra algorithms is exponential in the number of required accuracy bits. We show that by combining randomized algorithms with older ideas, and through careful engineering, randomized numerical linear algebra algorithms can be made practical.

More specifically, we describe a least-square solver for dense highly overdetermined systems that achieves residuals similar to those of direct QR factorization based solvers (LAPACK), outperforms LAPACK by large factors, and scales significantly better than any QR-based solver. Our solver uses a traditional iterative solver to solve the system, but the preconditioner is built using a randomized algorithm. This fusion of randomization and hybridization is what makes the algorithm fast and reliable. This is a joint work with Petar Maymounkov and Sivan Toledo.

Contents

CHAPTER 1

Introduction

Numerical linear algebra is ubiquitous in scientific computing. For example, modeling real-world changing conditions, such as weather, air flow around a plane, automobile body deformation in a crash, the motion of stars in a galaxy, etc. almost always requires solving linear systems or computing eigenvalues. Linear solvers that are specialized to specific application areas, say solving the pressure equation in ocean models, work well on the linear systems that arise in these applications. But as the number of scientific applications increases, it is not always practical to rely solely on special purpose solvers. Many research and industrial centers are in need of widely-applicable general-purpose solvers. During my PhD research I worked on developing new general purpose algorithms based on two ideas: hybrid linear solvers, and randomization.

There are two methodologies for solving both sparse and dense systems of linear equations: direct and iterative. Direct solvers factor the coefficient matrix into a product of simpler matrices whose inverse is easy to apply. Iterative solvers start with an approximate solution and iteratively refine it. A brief description of these techniques appears later in the introduction.

Each of these two methodologies has its advantages. Direct methods tend to be generic, robust, predictable and efficient. But their scalability is limited: they may run out of memory, and be too slow for very large matrices. Iterative methods scale much better, but are usually more fragile, less robust and slower than direct methods. These difficulties are usually overcome by using a preconditioner. Many preconditionning schemes are tailored to fit a specific problem, and the solver loses generality. But preconditionning also opens the door for hybridization: a direct method can be used to find a preconditioner. By embedding a direct method inside the iterative method we are able to enhance its robustness and efficiency without sacrificing generality.

Hybrid solvers are the first theme of the thesis. Chapter 2 describes a support-preconditionning scheme for scalar elliptic finite-element matrices. Chapters 3, 4 and 5 show how a factorization of a perturbed matrix can be used as a preconditioner for least-squares problems. Chapter 6 explores the use of incomplete indefinite factorization in short recurrence iterative methods.

Chapter 7 changes direction. It focuses on the second theme of the thesis, randomization. Randomization is arguably the most exciting and innovative idea to have hit linear algebra in a long time. Several such algorithms have been proposed and explored in the past decade (see, e.g, [**186, 89, 80, 179, 162, 88, 178, 71, 45**] and the references therein). Some forms of randomization have been used for decades in linear algebra. For example, the starting vectors in Lanczos algorithms are always random. But recent research led to new uses of randomization: random mixing and random sampling, which can be combined to form random projections. These ideas have been explored theoretically and have found use in some specialized applications (e.g., data mining [**153, 45**]), but

they have had little influence so far on mainstream numerical linear algebra. What is exciting about randomization is that it may well be the key to the next big leap in capabilities. In the last few decades there has been considerable progress in the development of general purpose linear solvers. Although, these solvers work very well, the potential for improving them using well-researched techniques is limited. To make the next big leap in performance, new techniques must be discovered, investigated, and deployed.

Chapter 8 fuses ideas from both themes, hybridization and randomization. It presents an efficient randomized dense least-squares solver. The algorithm combines the use of randomization with hybridization: a preconditioner built using a randomized algorithm is used within a deterministic iterative solver. We believe that chapter 8 may present a roadmap of how randomized algorithms should be used in numerical linear algebra. Randomized algorithms are often too weak to be used on their own, and must be combined with older ideas to yield practical solvers. We discuss this issue more later in the introduction.

Some of my research results are not included in this thesis. One is an application of the trace-estimation algorithm: a method to estimate the number of triangles in a graph, an important problem in data-mining [**15**]. I participated in the the optimization and performance evaluation of software library for high-performance computing [**137**]. During my MSc studies, I developed a parallel sparse direct solver for unsymmetric matrices [**23**]. I also co-authored a book chapter on combinatorial preconditioners [**210**].

The rest of this chapter surveys the results of this thesis. It aims to be self-contained; it starts with an brief background, and continues with descriptions of the actual results, with each section summarizing one chapter in the thesis. The summaries do not contain proofs but rather focus on the motivation, intuition and main results.

1.1. A brief background: solving linear equations

This thesis deals mainly with solving linear systems using a general purpose solver (the only exception is chapter 7). In particular, we are interested in solving the system $Ax = b$ where $A \in \mathbb{R}^{n\times n}$ and $b \in \mathbb{R}^n$, or the least-squares system $x = \arg\min_x \|Ax - b\|_2$ where $A \in \mathbb{R}^{m\times n}$ and $b \in \mathbb{R}^m$. The distinction between a *general-purpose* solvers and *specialized* solvers is not always clear-cut. In the context of this thesis we seek solvers that use only algebraic properties of A and that do not use any information about the scientific problem in which the matrix arose. For example, a solver that works for any symmetric positive definite matrix (i.e., a symmetric matrix with all positive eigenvalues) is a general purpose solver. A solver that solves a discretization of a continuous ocean model by coarsening the discretization is a specialized solver.

This section surveys existing techniques for designing general purpose solvers.

1.1.1. Direct Solvers. *Direct solvers* solve equations involving a matrix by factorizing the matrix as a product of two or more simpler matrices. For example, the most commonly used factorization for general systems is $A = LU$ where L is lower triangular and U is upper triangular. This factorization, called the *LU factorization*, is a version of Gaussian elimination. Using this factorization the equation $Ax = b$ can be solved for any b: solve $Ly = b$ and then solve $Ux = y$. This is, of course, a simplified presentation of an actual solver, since there are issues of stability that need to be addressed. Nevertheless,

the basic structure remains the same: factor the matrix into a product of simpler matrices, and solve a short sequence of simple linear systems.

There are variants of this technique for different types of linear systems. For a symmetric positive definite (SPD) matrix a Cholesky factorization, $A = LL^T$, is used (L is lower triangular). For least-squares problems a QR factorization, $A = QR$, is used (Q is orthonormal, R is upper triangular). For dense systems, the conventional implementations of these methods run in time that is cubic in the dimension of the problem, $\Theta(n^3)$ for LU and Cholesky, $\Theta(nm^2)$ for QR. So-called fast methods [**206, 80**] are not widely used.

If A is large and sparse (that is only a small number of matrix entries are nonzero) then we can avoid the cubic complexity by using methods that take advantage of sparsity. These types of solvers are not the focus of this thesis; it suffices to say that there are excellent implementations of sparse direct solvers (for example, SuiteSparse [**73**], TAUCS [**211**], MUMPS [**10**], HSL [**90**], PasTiX [**122**], SuperLU [**82**], PARADISO [**187**], WSMP [**115**] and many more). On many classes of sparse matrices, their asymptotic run time is much better than cubic, but it is very rarely near linear. The super-linear execution times limit the scalability of sparse direct solvers. Nevertheless, it is important to compare any new solver to state-of-the-art sparse direct solvers; we do so in this thesis where appropriate.

1.1.2. Kyrlov-Subspace Iterative Methods. Krylov-subspace iterative methods for solving large systems of linear equations treat the matrix as a black box. The only operation they perform is to multiply vectors by the matrix. Using this basic operation, they find approximations to the solution inside the so-called *Krylov-subspaces*,

$$\mathcal{K}_n(A, b) = \left\{b, Ab, A^2b, \ldots, A^{n-1}n\right\} .$$

Popular iterative methods include CG [**124**] (for SPD matrices), LSQR [**166**] (least-squares problems) and GMRES [**185**] (for general matrices). On certain classes of matrices, these solvers achieve better scalability than direct solvers.

1.1.3. Preconditionning. In many cases, Krylov-subspace methods are slow unless a *preconditioner* is used. The convergence rate of iterative methods depends on properties of A. For example, the convergence of CG depends on the condition number of A, the ratio between its largest and smallest singular values. If a preconditioner M is used, then the system that is solved is $M^{-1}Ax = M^{-1}b$. If the condition number of $M^{-1}A$ is small and M^{-1} is easy to apply, the preconditioner is effective. Preconditioners allow us to move between two extremes: pure direct solvers and pure iterative solvers. If $M = A$, the solver is essentially a direct solver; if M is the identity matrix, the solver is an un-preconditioned iterative method.

1.1.4. Spectral Analysis of Preconditioners. Analyzing preconditioners for symmetric positive definite and symmetric positive semi-definite (SPSD) matrices is usually done in terms of *generalized eigenvalues* and *generalized condition number*.

Definition 1.1.1. Let S and T be n-by-n Hermitian positive semidefinite matrices. We say that a scalar λ is a *finite generalized eigenvalue* of the matrix pencil (pair) (S, T) if there is a vector $v \neq 0$ such that

$$Sv = \lambda Tv$$

and $Tv \neq 0$. We say that ∞ is a *infinite generalized eigenvalue* of (S,T) if there exists a vector $v \neq 0$ such that $Tv = 0$ but $Sv \neq 0$. Note that ∞ is an eigenvalue of (S,T) if and only if 0 is an eigenvalue of (T,S). The finite and infinite eigenvalues of a pencil are *determined eigenvalues* (the eigenvector uniquely determines the eigenvalue). If both $Sv = Tv = 0$ for a vector $v \neq 0$, we say that v is an *indeterminate eigenvector*, because $Sv = \lambda Tv$ for any scalar λ. We denote the set of determined generalized eigenvalues of (S,T) is denoted by $\Lambda(S,T)$.

Throughout this chapter eigenvalues are ordered from smallest to largest. We will denote the kth eigenvalue of S by $\lambda_k(S)$, and the kth determined generalized eigenvalue of (S,T) by $\lambda_k(S,T)$. Therefore $\lambda_1(S) \leq \cdots \leq \lambda_l(S)$ and $\lambda_1(S,T) \leq \cdots \leq \lambda_d(S,T)$, where l is the number of eigenvalues S has, and d is the number of determined eigenvalues that (S,T) has.

Definition 1.1.2. Let S and T be n-by-n Hermitian positive semidefinite matrices with the same null space. The generalized condition number $\kappa(S,T)$ is the ratio between the largest determined generalized eigenvalue and the smallest determined eigenvalue of (S,T).

The behavior of preconditioned iterative methods is determined by the clustering of the generalized eigenvalues, and number of iterations is proportional to the generalized condition number. CG will converge in $O(\sqrt{\kappa(A,M)})$ iterations. LSQR on A preconditioned by R will converge in $O(\sqrt{\kappa(A^*A, R^*R)}$ iterations.

1.1.5. General-purpose Preconditioning Methods. There are many ways to form a preconditioner M but most of them are specialized.

Arguably, the most obvious paradigm for designing general-purpose preconditionning methods is hybridization. An hybrid iterative-direct method builds a factorization of a different yet related matrix $\tilde{A}$, thus $M = \tilde{A}$. The desired properties for $\tilde{A}$ are similarity to A and ease of factorization. By embedding a direct solver into an iterative method, we sometimes gain provable and practical robustness and efficiency. This thesis shows that by using hybridization we can design provably effective general-purpose linear solvers.

One such strategy is based on incomplete factorizations **[30, 184]**. It is widely used and often works well. Direct methods tend not to scale well because of fill-in, which are positions that are zero in the matrix but are nonzero in the factors; in a sense the factors tend to "lose sparsity". Incomplete factorization schemes address this problem directly using some sparsity preserving heuristic. For example, drop-tolerance incomplete factorizations truncate values that are smaller than some prescribed drop tolerance. Support preconditioning is another strategy based on hybridization. We discuss it further below.

Two other techniques worth mentioning in this context of general purpose preconditionning are sparse approximate inverses **[31]** and algebraic multigrid **[183]**.

1.1.6. Support Preconditionning. *Support preconditionning* is a combinatorial method for constructing preconditioners.

A matrix $A \in \mathbb{R}^{n\times n}$ is *diagonally-dominant* if for $i = 1, \ldots, n$ we have

$$A_{ii} \geq \sum_{j\neq i} |A_{ij}| .$$

Symmetric diagonally-dominant (SDD) matrices can be viewed as weighted graphs and vice versa: $G(A) = (V, E)$ where $V = \{1, \ldots, n\}$ and $E = \{(i, j) \text{ s.t. } i \neq j \text{ and } A_{ij} \neq 0\}$. The weight of (i, j) is $|A_{ij}|$.

Support theory provides tools for bounding the condition number of $M^{-1}A$ based on combinatorial properties of $G(A)$ and $G(M)$. Using support theory we can move from the linear algebra domain to the graph theory domain, and build preconditioner using combinatorial graph-theoretic techniques. Both the requirement that M is easy-to-factor and that M approximates A well can be stated as graph-theoretic proprieties of $G(M)$ and of $G(M)$ relative to $G(A)$. Once $G(M)$ is found M is formed and factored.

Over the last decade many provably effective preconditioners have been developed for SDD matrices, including some nearly linear time algorithms ([**215, 113, 41, 152, 140, 203, 141**], to name a few) based on support theory. Some progress has been recently made in the attempt to extend support preconditioners to non diagonally-dominant matrices [**44, 194, 69, 221**].

1.2. Combinatorial preconditioners for scalar elliptic finite-element models

A popular method for numerically solving partial differential equations is the *finite element method*. The method constructs a matrix K and a vector b and solves the equation

$$Kx = b\,.$$

The matrix $K \in \mathbb{R}^{n \times n}$ is called a *stiffness matrix*, and it is a sum of *element matrices*, $K = \sum_{e \in E} K_e$. Matrix K_e is zero outside a small set of n_e rows and columns. In most cases n_e is uniformly bounded by a small integer (in our experiments n_e is 4 or 10). We denote the set of nonzero rows and columns of K_e by $\mathcal{N}_e$. We denote by K_e^R the restriction of matrix K_e to $\mathcal{N}_e$. The system $Kx = b$ is a *finite element linear system*.

Chapter 2 describes a solver for a subclass of finite element systems. Let $\mathbb{N}_1^n$ be a linear subspace of $\mathbb{R}^n$ that is spanned by the vector $\begin{bmatrix}1 & 1 & \cdots & 1\end{bmatrix}^T$; we drop n where the dimension is obvious. Our solver solves finite element systems where $\text{null}(K) = \mathbb{N}_1^n$ and $\text{null}(K_e^R) = \mathbb{N}_1^{n_e}$ for all $e \in E$.

Although the statement of the matrices for which our solver works on is purely algebraic, we have a specific application area in mind. Consider the following linear second-order elliptic problem: find a $u \colon \Omega \longrightarrow \mathbb{R}$ satisfying

$$\begin{aligned} -\nabla \cdot (\theta(x) \nabla u) &= f && \text{on } \Omega \\ u &= u_0 && \text{on } \Gamma_1\,, \\ \theta \partial u / \partial n &= g && \text{on } \Gamma_2\,. \end{aligned} \tag{1.2.1}$$

The domain Ω is a bounded open set of $\mathbb{R}^d$ and Γ_1 and Γ_2 form a partition of the boundary of Ω. The *conductivity* θ is a spatially varying d-by-d symmetric positive definite matrix, f is a scalar *forcing function*, u_0 is a *Dirichlet boundary condition* and g is a *Neumann boundary condition*. Equation (1.2.1) is the *Poisson Equation*, and it arises in many applications including electrostatics, mechanical engineering and theoretical physics. The variant $f = 0$ is called the *Laplace Equation*, and it arises in many applications including electromagnetism, astronomy and fluid dynamics.

Preconditionning methods for these type of equations have existed before our work. The main novelty in our work was the use of *support preconditioners*. Till our work, support preconditioners were restricted to SDD matrices, although there has been some

theoretical work on extending them to finite element methods [**43, 194, 69**]. We contributed additional results to this theory, and we have implemented and evaluated a practical solver.

This contribution is in some sense specialized and in another sense general-purpose. It is specialized in that matrices with the required structure arise mostly from specific physical models. It is general-purpose in that the algorithm itself is purely algebraic.

Chapter 2 is based on a paper on this solver that was published in the *SIAM Journal on Matrix Analysis and Applications* [**16**], co-authored by Doron Chen, Gil Shklarski, and Sivan Toledo. Both I and another PhD student in Tel-Aviv University, Gil Shklarski, made significant (but separate) contributions to this paper.

1.2.1. Overview of the Solver. Our method begins by forming a symmetric diagonally-dominant approximation L to K. The approximation is formed by building an approximation L_e for each element matrix K_e. Our approximation of the element matrices are provably good. Our solver then splits the elements into two sets. One set, denoted $E(t)$, contains all the elements in which the approximation is better than some parameter t, and the other set contains the inapproximable elements. We now scale each L_e so that the sum $L_{\leq t} = \sum_{e \in E(t)} \alpha_e L_e$ of scaled approximations is spectrally close to $K_{\leq t} = \sum_{e \in E(t)} K_e$. This is achieved by setting $\alpha_e = \min \Lambda(K_e, L_e)$.

The matrix $L_{\leq t}$ is itself an SDD matrix. Our algorithm uses one of the combinatorial algorithms discussed in 1.1.6 to construct another SDD matrix $M_{\leq t}$ that approximates $L_{\leq t}$. The difference between $M_{\leq t}$ and $L_{\leq t}$ is that $M_{\leq t}$ can be factored more quickly into sparse triangular factors than $L_{\leq t}$. We now find a scaling factor γ such that $\gamma M_{\leq t} + \sum_{e \notin E(t)} K_e$ is spectrally close to K. To find γ we first find some vector v that is orthogonal to $\mathrm{null}(M_{\leq t})$. We then set γ to its generalized Rayleigh quotient:

$$\gamma = \frac{v^T K_{\leq t} v}{v^T M_{\leq t} v} \,.$$

The null space of $M_{\leq t}$ is determined by the connected components of its graph, so it is easy to quickly find such a v (we use a random v in this subspace). Our preconditioner is $M = \gamma M_{\leq t} + \sum_{e \notin E(t)} K_e$. To solve the equation we factor M and use its factors as a factored preconditioner in a preconditioned symmetric Krylov-subspace solver such as CG and MINRES.

Our algorithm uses the *SDD approximation approach,* which is illustrated in 1.2.1: the system matrix K is approximated using an SDD matrix L, and a preconditioner M is built for L. The two crucial observations are the *approximation chain-rule* and *element-by-element approximability.* The approximation chain-rule states that if L is a good approximation of K and M is a good approximation of L then M is a good approximation of K. The formal statement is

$$\kappa(K, M) \leq \kappa(K, L) \cdot \kappa(L, M) \,.$$

The element-by-element approximability rule says that to approximate $K = \sum_{e \in E} K_e$ you can build an approximation L_e for each K_e and use $L = \sum_{e \in E} \alpha_e L_e$, where $\alpha_e = \min \Lambda(K_e, L_e)$, as an approximation of K. The formal statement is

$$\kappa(K, L) \leq \max_{e \in E} \kappa(K_e, L_e) \,.$$

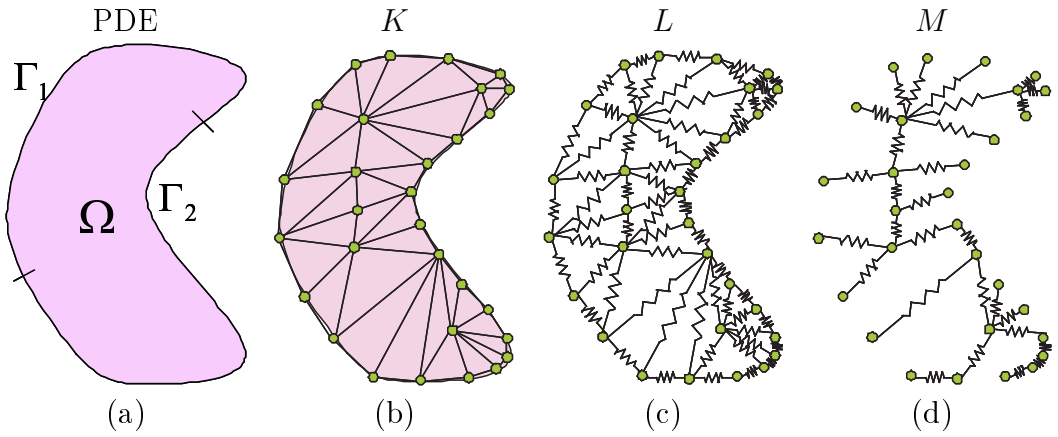


FIGURE 1.2.1. Illustration of the SDD approximation approach. Part (a) illustrates the original scalar elliptic partial differential equation. Part (b) illustrates its finite-element discretization; each triangle corresponds to a single K_e, and each node represents an unknown. The matrix $K = \sum_e K_e$. Part (c) illustrates an SDD approximation L of K. Each triangle of resistors corresponds to L_e and $L = \sum_e L_e$. Part (d) illustrates M, a preconditioner of L.

The SDD approximation approach is not new. Reitzinger et al. [**118, 174, 142**] have used complicated and rather expansive algorithms to approximate the K_e's by L_e's. Moreover, their solver was sensitive to element approximation difficulties. The second group of works by Boman, Hendrickson and Vavasis [**44**], Gupta [**117**], and Wang and Sarin [**221**] construct the matrices L_e's using geometrical, physical and discretization-specific information (not using only the $K'_e s$). Their works are therefore limited to specific finite-element formulations. This limitation is also evident in the fairly limited experiments performed with those preconditioners (i.e., results limited to specific element types).

1.2.2. Our Contributions. Our goal was to produce a high quality solver. To achieve this goal we had to make substantial theoretical and practical contributions:

(1) We designed a novel and efficient nearly optimal algorithm for approximating the element matrices (K_e's) using SDD matrices (L_e's). Our algorithm is purely algebraic and works for any element type.
(2) We study the issue of approximability.
(3) We noticed that a single bad approximation can ruin preconditioner quality. We presented a scheme to smoothly handle inapproximable elements.
(4) We engineered a practical solver and performed an extensive comparative study.

1.2.3. Implementation and Experimental Results. The last part in this effort is a practical implementation of our solver and a comparative study. We have implemented most of the code in C and used MATLAB's [**155**] CMEX API for a benchmarking interface. We have used Vaidya's combinatorial preconditioner from TAUCS, which is the implementation reported in [**63**]. We have factored our preconditioner using CHOLMOD 1.0 sparse Cholesky by Tim Davis, with METIS [**138**] fill reducing permutation.

We compared our solver against a direct solver (CHOLMOD 1.0 with METIS), against a preconditioned drop-tolerance incomplete Cholesky solver, and against an algebraic

multigrid (AMG) solver (BoomerAMG [**123**]). We tested these solvers on various three dimensional models, with various element types.

The full results appear in Chapter 2. Here we only summarize the results:

(1) Generally, our solver was robust; it produced consistent results for varying parameters and problem sizes. It was not always the fastest, but it was predictable and reliable. In no cases did it perform poorly.
(2) On small problems the direct solver was the most robust and faster than any other method. But when problem is large the problem is no longer solvable using a direct method because it does not fit into memory (we used a machine with about 8GB of RAM).
(3) On large problems, even very ill-conditioned ones, when all elements are well approximable, our solver worked well, but AMG and drop tolerance incomplete Cholesky are faster.
(4) On large problems with some inapproximable (and therefore ill-conditioned) elements, our solver was the fastest and most reliable. AMG performed poorly, and incomplete Cholesky based solvers were unreliable. Inapproximable elements can arise for many reasons. One example is the element matrix for an isosceles triangle with two tiny angles and one that almost π [**22, 193**].

A preview of the results is given in Figure 1.2.2. This graphs shows that when some elements are ill-conditioned our algorithm is superior. For the specific system the graph on the left shows the distributions of $\kappa(K_e)$ and $\kappa(K_e, L_e)$. The right graph shows the running time of the various algorithms. The vertical axis represents wall-clock time for all the phases of the solution and the horizontal axis represents the number of nonzeros in the triangular factors of the preconditioner. The rightmost (largest) horizontal coordinate in the graphs corresponds to the number of nonzeros in a complete sparse Cholesky factor of the coefficient matrix. The complete factorization runs out of space, so we estimate the running time of the complete factorization based on the assumptions that it runs at 10^9 floating-point operations per second. We see that our method, labeled "NOC+Vaidya" is faster than other methods.

1.3. Using perturbed QR factorizations to solve linear least-squares problems

Consider the following simple observation. Let A be a given matrix and let $\hat{A} = \left[\begin{smallmatrix} A \\ B \end{smallmatrix}\right]$. Then

$$\begin{aligned} \left(\hat{A}^*\hat{A}\right)^{-1} A^*A &= (A^*A + B^*B)^{-1} A^*A \\ &= (A^*A + B^*B)^{-1} (A^*A + B^*B - B^*B) \\ &= I - (A^*A + B^*B)^{-1} B^*B \,. \end{aligned} \tag{1.3.1}$$

The rank of second term on the last line is at most the rank of B, so if B has low rank, then $\left(\hat{A}^*\hat{A}\right)^{-1} A^*A$ is a low-rank perturbation of the identity. A symmetric rank-k perturbation of the identity has at most k non-unit eigenvalues, which in exact arithmetic guarantees convergence in k iterations in several Krylov-subspace methods. Therefore, the Cholesky factor of $\hat{A}^*\hat{A}$ (which is also the R factor from a QR factorization of $\hat{A}$) is a good least-squares preconditioner for A. The same analysis extends to the case where we drop rows of A. This idea has been used by practitioners [**105**].

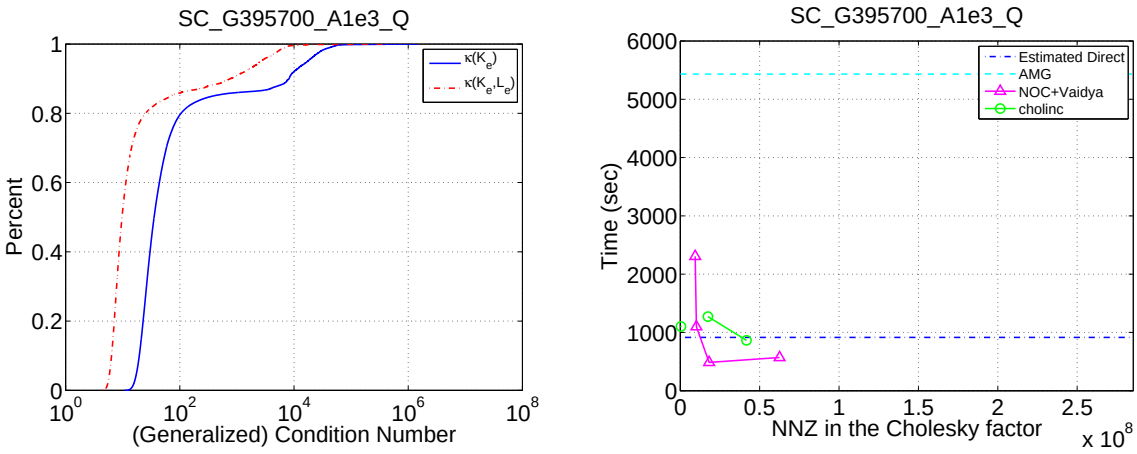


FIGURE 1.2.2. The results of the various finite-element solvers on a matrix with some very ill-conditioned elements. The left graph is the distribution of element condition number. We see that there elements that are very ill conditioned. The right graph is running time for various levels of fill-in of the Cholesky factor. Our solver is labeled "NOC+Vaidya", and incomplete Cholesky solver is labeled "cholinc".

Chapter 3 generalizes this observation to other matrix perturbations: to the case where $\hat{A}$ is singular, to the case where rows are removed instead of added, to column exchanges, and to preconditioners for $\hat{A}$ rather than its R factor. We also bound the size of the non-unit eigenvalues, which is important when A is rank deficient. Using these generalizations we propose a set of applications. Some of these applications are explored in subsequent chapters. Chapter 4 proposes a solver for rank deficient systems using the technique of row additions. In Chapter 5 we use the row removal technique to solve certain types of equations that occur in a graphics application.

Chapter 3 is based on a paper that was published in the *SIAM Journal on Matrix Analysis and Applications* [**20**], co-authored by Esmond Ng and Sivan Toledo.

1.3.1. Theoretical Results. Chapter 3 explores matrix perturbations and their applications. The main bulk of the chapter is a comprehensive spectral analysis of the generalized spectrum of matrix pencils that arise from row and column perturbations. This analysis forms the theoretical basis for the applications discussed in the second part of the chapter and in subsequent chapters. Here we review the main results of this analysis without proving them.

The first part of the analysis shows that if the number of rows/columns that are added, dropped, or replaced is small, then most of the generalized eigenvalues are 1. The number of *runaway* eigenvalues, the ones that are not 1, is bounded by the number of rows added or dropped (for row perturbations) or by twice the number of columns replaced (for column perturbations). This guarantees rapid convergence of an iterative method if the R factor is used as a preconditioner.

Theorem 1.3.1. *Let $A \in \mathbb{C}^{m\times n}$ and let $B \in \mathbb{C}^{k\times n}$ and $C \in \mathbb{C}^{r\times n}$ for some $1 \leq k+r < n$. Define*

$$\chi = \begin{bmatrix} B \\ C \end{bmatrix}.$$

The following claims hold:

(1) *In the pencil $(A^*A, A^*A + B^*B - C^*C)$, at most $\operatorname{rank}(\chi) \leq k + r$ generalized determined eigenvalues may be different from 1 (counting multiplicities).*
(2) *If 1 is not a generalized eigenvalue of the pencil (B^*B, C^*C) and $A^*A + B^*B - C^*C$ is full rank, then (a) the pencil $(A^*A, A^*A + B^*B - C^*C)$ does not have indeterminate eigenvectors, (b) the multiplicity of the eigenvalue 1 is exactly $\dim\operatorname{null}(\chi) \geq n - k - r$, and (c) the multiplicity of the zero eigenvalue is exactly $\dim\operatorname{null}(A)$.*
(3) *The sum pencil $(A^*A, A^*A + B^*B)$ cannot have an infinite eigenvalue and all its eigenvalues are in the interval $[0, 1]$.*

Theorem 1.3.2. *Let $D \in \mathbb{C}^{m\times n}$ and let $E \in \mathbb{C}^{m\times k}$ and $F \in \mathbb{C}^{m\times k}$ for some $1 \leq k < n$. Let*

$$A = \begin{bmatrix} D & E \end{bmatrix} \in \mathbb{C}^{m\times(n+k)}$$

and let

$$\tilde{A} = \begin{bmatrix} D & F \end{bmatrix} \in \mathbb{C}^{m\times(n+k)}.$$

In the pencil $(A^*A, \tilde{A}^*\tilde{A})$, at least $n - k$ generalized finite eigenvalues are 1.

The second part of the analysis concentrates on perturbations of a preconditioned system. We address the following question: if M is an effective preconditioner for $\hat{A}^*\hat{A}$, is it also an effective preconditioner for A^*A? The analysis shows that if the generalized spectrum of $\hat{A}^*\hat{A}$ is contained in a small interval, then nearly all of A^*A's spectrum is contained in the same interval. The number of runaway eigenvalues, in this case ones that are outside the interval, is bounded by the number of rows added or dropped (for row perturbations) or by twice the number of columns replaced (for column perturbations). This guarantees that if M is an effective preconditioner of $\hat{A}^*\hat{A}$ due to well-conditioning of the preconditioned matrix, it is also an effective preconditioner for A^*A.

Theorem 1.3.3. *Let $A \in \mathbb{C}^{m\times n}$ and let $B \in \mathbb{C}^{k\times n}$ and $C \in \mathbb{C}^{r\times n}$ for some $1 \leq k+r < n$. Let $M \in \mathbb{C}^{n\times n}$ be an Hermitian positive semidefinite matrix. Suppose that $\operatorname{null}(M) \subseteq \operatorname{null}(A^*A)$, $\operatorname{null}(M) \subseteq \operatorname{null}(B^*B)$ and $\operatorname{null}(M) \subseteq \operatorname{null}(C^*C)$. If*

$$\alpha \leq \lambda_1(A^*A, M) \leq \lambda_{\operatorname{rank}(M)}(A^*A, M) \leq \beta.$$

then

$$\alpha \leq \lambda_{r+1}(A^*A + B^*B - C^*C, M) \leq \lambda_{\operatorname{rank}(M)-k}(A^*A + B^*B - C^*C, M) \leq \beta.$$

Theorem 1.3.4. *Let $D \in \mathbb{C}^{m\times n}$ and let $E \in \mathbb{C}^{m\times k}$ and $F \in \mathbb{C}^{m\times k}$ for some $1 \leq k < n$. Let*

$$A = \begin{bmatrix} D & E \end{bmatrix} \in \mathbb{C}^{m\times(n+k)}$$

and let

$$\tilde{A} = \begin{bmatrix} D & F \end{bmatrix} \in \mathbb{C}^{m\times(n+k)}.$$

Let $M \in \mathbb{C}^{(n+k)\times(n+k)}$ *be an Hermitian positive semidefinite matrix, such that* $\text{null}(M) \subseteq \text{null}(A^*A)$ *and* $\text{null}(M) \subseteq \text{null}(\tilde{A}^*\tilde{A})$. *Suppose that*

$$\alpha \leq \lambda_1(A^*A, M) \leq \lambda_{\text{rank}(M)}(A^*A, M) \leq \beta\,.$$

Then we have

$$\alpha \leq \lambda_{k+1}(\tilde{A}^*\tilde{A}, M) \leq \lambda_{\text{rank}(M)-k}(\tilde{A}^*\tilde{A}, M) \leq \beta\,.$$

The ability of preconditioned LSQR to solve a regularization of an ill-conditioned least squares system is dependent on whether the numerical rank of the preconditioned system is similar to the numerical rank of the original system. The third part of the analysis shows that if a preconditioner is obtained from adding rows to A then, under some restrictions, the numerical rank is maintained up to an appropriate relaxation of the rank threshold. The restrictions are that the preconditioner is well-conditioned and the norm of the perturbation is not too large.

Theorem 1.3.5. *Let* $A \in \mathbb{C}^{m\times n}$ *and let* $B \in \mathbb{C}^{k\times n}$ *for some* $1 \leq k < n$. *Assume that* $A^*A + B^*B$ *is full rank. Denote* $\alpha = ||A^*A||_2$. *If there are* d *eigenvalues of* A^*A *that are smaller than or equal to* $\epsilon\alpha$ *for some* $1 > \epsilon > 0$, *then* d *generalized eigenvalues of* $(A^*A, A^*A + B^*B)$ *are smaller than or equal to* $\epsilon\kappa(A^*A + B^*B)$.

Theorem 1.3.6. *Let* $A \in \mathbb{C}^{m\times n}$ *and let* $B \in \mathbb{C}^{k\times n}$ *for some* $1 \leq k < n$. *Assume that* $A^*A + B^*B$ *is full rank. Denote* $\alpha = ||A^*A||_2$ *and suppose that* $||B^*B||_2 \leq \gamma\alpha$. *If there are* d *eigenvalues of* A^*A *that are larger than or equal to* $\eta\alpha$ *for some* $1 > \eta > 0$ *then* d *generalized eigenvalues of* $(A^*A, A^*A + B^*B)$ *are larger than or equal to* $\eta/(1+\gamma)$.

When $A \in \mathbb{R}^{m\times n}$ is rank deficient, there is an entire subspace of minimizers of $\|Ax - b\|_2$. When A is full rank but highly ill-conditioned, there is a single minimizer, but there are many x's that give almost the same residual norm. Of these minimizers or almost-minimizers, the user usually prefers a solution with a small norm. Formally, the set of solution we are interested in is the minimizers of $\|\bar{A}x - b\|_2$, where $\bar{A}$ is the matrix with the same singular value decomposition as A except that small singular values (below a threshold) are truncated to zero. Finding $\bar{A}$ is expensive and non-practical. We have shown that if we can find a matrix $B \in \mathbb{R}^{k\times n}$ whose number of rows is *exactly* the same as the number of singular values we wish to truncate, and $A^*A + B^*B$ is well-conditioned, then a direct method can find an approximation of a minimizer of $\|\bar{A}x - b\|_2$. This minimizer usually has a small norm. Let $\left[\begin{smallmatrix} A \\ B \end{smallmatrix}\right] = QR$ and $P = \left[\begin{smallmatrix} I_{m\times m} & 0_{m\times k} \end{smallmatrix}\right]$. Our almost minimizer is $\hat{x} = R^{-1}(PQ)^*b$.

The first lemma shows that this is an exact minimizer where$\|Ax - b\|_2$ when A is exactly rank-deficient (no small singular values but there are 0 singular values).

Lemma 1.3.7. *Let* $A \in \mathbb{C}^{m\times n}$, $B \in \mathbb{C}^{k\times n}$ *and* $b \in \mathbb{C}^{m\times r}$. *Suppose that* $\text{rank}(A) = n - k$ *and* $\left[\begin{smallmatrix} A \\ B \end{smallmatrix}\right]$ *has full rank. Let* $\left[\begin{smallmatrix} A \\ B \end{smallmatrix}\right] = QR$ *be a QR factorization of* $\left[\begin{smallmatrix} A \\ B \end{smallmatrix}\right]$. *Let* $P = \left[\begin{smallmatrix} I_{m\times m} & 0_{m\times k} \end{smallmatrix}\right]$. *The vector* $\hat{x} = R^{-1}(PQ)^*b$ *is a minimizer of* $\min_x \|Ax - b\|_2$.

We now state the second theorem and then explain what it means.

Theorem 1.3.8. *Let* $A \in \mathbb{C}^{m\times n}$, $B \in \mathbb{C}^{k\times n}$ *and* $b \in \mathbb{C}^m$. *Let* $\bar{A}$ *be the matrix with the same singular value decomposition as* A *except that the* k *smallest singular values are truncated to* 0. *Denote*

$$C = \left[\begin{smallmatrix} A \\ B \end{smallmatrix}\right] \text{ and } D = \left[\begin{smallmatrix} \bar{A} \\ B \end{smallmatrix}\right]$$

Assume that C and D are both full rank. Let $C = QR$ be a QR factorization of C and $D = \bar{Q}\bar{R}$ be the QR factorization of D. Denote

$$\delta = \frac{\sigma_{n-k+1}(A)}{\sigma_{\min}(C)}$$

where $\sigma_{n-k}(A)$ is the kth smallest singular value of A and $\sigma_{\min}(C)$ is the smallest singular value of C. Let $P = \begin{bmatrix} I_{m\times m} & 0_{m\times k} \end{bmatrix}$. Define the solutions

$$\hat{x} = R^{-1}(PQ)^* b$$

and

$$\hat{z} = \bar{R}^{-1}(P\bar{Q})^* b\,.$$

Then, provided that $\delta < 1$,

$$\frac{\|\hat{x} - \hat{z}\|_2}{\|\hat{x}\|_2} \leq \frac{\delta}{1-\delta}\left(2 + (\kappa(R)+1)\frac{\|r\|_2}{\|R\|_2\|\hat{z}\|_2}\right)$$

where

$$r = b - A\hat{x}\,.$$

According to Lemma 1.3.7 $\hat{z}$ is a minimizer of $\|\bar{A}x - b\|_2$. This theorem shows that under certain conditions $\hat{x}$ approximates $\hat{z}$ well, and is therefore an almost-minimizer. The theorem states that if δ is small (which happens when C is well conditioned and A has k tiny singular values) and R is not ill conditioned and not too large, and the norm of r is not too large, then $\hat{x}$ is a good approximation of the minimizer $\hat{z}$ that we seek. The quantity that is hard to estimate in practice is δ, which depends on the small singular values of A. Therefore, the formula is useful mainly when we know a-priori the number of small singular values of A.

1.3.2. Applications to Least-Squares Solvers. Chapter 3 suggest a few applications of the theory.

Dropping Dense Rows for Sparsity. The R factor of $A = QR$ is also the Cholesky factor of A^*A. Rows of A that are fairly dense cause A^*A to be fairly dense, which usually cause R to be dense. In the extreme case, a completely dense row in A causes A^*A and R to be completely dense. A simple solution is to drop fairly dense column before the factorization starts, and use the factor as a preconditioner in LSQR. A sophisticated algorithm to decide which rows to drop is an open research question.

Updating and Downdating. Updating a least-squares problem involves adding rows to the coefficient matrix A and to the right-hand-side b. Downdating involves dropping rows. Our analysis shows that after adding and/or dropping a small number of rows from/to A, the R factor of A is an effective preconditioner of the new system, as long as the new system is full-rank.

Adding Rows to Solve Numerically Rank-Deficient Problems. When A is full rank but highly ill-conditioned, it is desirable to solve a regularization of the least squares problem $\min_x \|Ax-b\|_2$, that is a solution of small norm. The factorization $A = QR$ is not useful for solving ill-conditioned least-squares problems. The factorization is backward stable, but the computed R is ill-conditioned. This often causes the solver to produce a solution $x = R^{-1}Q^*b$ with a huge norm. We propose to add rows to the coefficient matrix A to avoid ill-conditioning in R. The factor R is no longer the R factor of A, but the R factor of a perturbed matrix $\left[\begin{smallmatrix} A \\ B \end{smallmatrix}\right]$. Our analysis shows that if R is well-conditioned and

only a small number of rows where added then R is an effective preconditioner for solving the regularized least-squares problem using LSQR.

In Chapter 3 we present an efficient algorithm that uses a threshold $\tau \geq n+1$ to find a B such that $\|B^*B\|_2 \leq m\|A^*A\|_2$ and $\kappa(A^*A + B^*B) \leq \tau^2$. Along with finding the perturbation the R factor of the perturbed matrix is found, usually with a small amount of additional work relative to finding the R factor of the original matrix. Furthermore, B's structure guarantees that R factor will fill no more than the original factor. The algorithm attempts to add as few rows as possible.

Solving What-If Scenarios. The theory presented in this section allows us to efficiently solve what-if scenarios of the following type. We are given a least squares problem $\min \|Ax - b\|_2$. We already computed the minimizer using the R factor of A or using some preconditioner. Now we want to know how the solution would change if we fix some of its entries, without loss of generality $x_{n-k+1} = c_{n-k+1}, \ldots, x_n = c_n$, where the c_i's are some constants. Our analysis shows that for small k the factor or the preconditioner of A is an effective preconditioner for a modified least-squares system that solves the what-if scenario.

1.4. A solver for rank-deficient overdetermined least-squares problems

Chapter 4 describes a sparse solver that I implemented based on the theory in Chapter 3. The solver solves rank deficient overdetermined least-squares problems. The solver uses a QR or Cholesky factorization as a preconditioner for LSQR. The QR factorization code is new; it avoids the use of column-pivoting, relying instead on the inherent regularization features of LSQR, which is the outer solver. The use of LSQR also allows us to use the Cholesky factorization of the normal equations without sacrificing numerical stability. The results in [**61**] ensure that as long as an appropriate threshold is used, the solution is backward stable.

1.4.1. Algorithm. The first step of our algorithms is to find a matrix $B \in \mathbb{R}^{k\times n}$ such that $\kappa(\left[\begin{smallmatrix} A \\ B \end{smallmatrix}\right]) \leq \tau$, where τ is an algorithmic parameter. To goal is to find a matrix B with as few rows as possible, and with a single non-zero in each row. Our algorithm then finds a QR factorization of this matrix, $\left[\begin{smallmatrix} A \\ B \end{smallmatrix}\right] = \tilde{Q}\tilde{R}$ (in an implementation there is no need to actually form $\tilde{Q}$). Exact details on how B is found appear in Chapter 4.

Once we have found the factorization, we have two options. One is to return $x = \tilde{R}^{-1}(P\tilde{Q})^T b$ where $P = \begin{pmatrix} I_{m\times m} & 0_{m\times k} \end{pmatrix}$, and the other is to use $\tilde{R}$ as a preconditioner in an iterative solution of the problem using LSQR. If LSQR is used, then the convergence threshold must be set to reflect the desired regularization, as explained in [**166**]. In our implementation, we have used MATLAB's [**209**] built in LSQR function and set the single convergence parameter to τ^{-1}.

Instead of finding the $\tilde{R}$ factor of $\left[\begin{smallmatrix} A \\ B \end{smallmatrix}\right]$, we can find the Cholesky factorization of $A^TA + B^TB = \tilde{R}^T\tilde{R}$, which under exact arithmetic should be the same. The factor can than be used in the iterative variant of our solver. It can also be used in the direct formula $x = \tilde{R}^{-1}\tilde{R}^{-T}A^Tb$ based on the identity $P\tilde{Q} = A\tilde{R}^{-1}$. A solver based on the direct formula is not backward stable for the same reasons that solving the normal equations is not backward stable [**126**, Section 20.4]. In the iterative solver, however, the factor is used only as a preconditioner, so the method will be as stable if an appropriate threshold

is used. The algorithm perturbs the Cholesky factorization in exactly the same way as in the QR factorization.

1.4.2. Implementation and Numerical Experiments. We have implemented a new sparse multifrontal QR factorization code. The algorithm used by the code is based on the work presented in [**151, 11**] although some of the details are different. The new multifrontal QR factorization code is now a part of TAUCS[1]. The experiments reported in Chapter 4 shows that our QR code performs well. Our algorithm was implemented on top of this new sparse QR implementation.

The Cholesky version of the algorithm was implemented by modifying Tim Davis's CHOLMOD package [**66, 77**].

We run experiments to test the behavior of the solver on rank deficient matrices. We ran on all rank deficient matrices in Tim Davis's collection that are small enough to fit in memory, and large enough to be of interest. We found that our solver is extremely fast: only a few iterations are needed, so the real bottleneck is the QR or Cholesky factorization. It also finds near-minimizer with small norm (only TSVD [**40**] finds better minimizers). On the other hand, it is not always easy to control the regularization and/or show optimality (both are easier with TSVD).

1.5. Application: ℓ_1-sparse reconstruction of sharp point set surfaces

Chapter 5 demonstrate the effectiveness of the techniques developed in Chapter 3 by demonstrating a real-life computer-graphics applications which required a customized linear solver.

The specific application is denoising of point clouds obtained from 3D laser scanners. State-of-the-art 3D laser scanners are capable of producing large amounts of raw, dense point sets. One of today's principal challenges in computer graphics is the development of robust point processing and reconstruction techniques that deal with the inherent noise of the acquired data set. In [**21**], which forms the basis for Chapter 5, we introduced an ℓ_1-sparse method for reconstructing a piecewise smooth point set surface.

Our technique is motivated by recent advances in sparse signal reconstruction. The underlying assumption in our work is that common objects, even geometrically complex ones, can typically be characterized by a rather small number of features. This, in turn, naturally lends itself to incorporating the powerful notion of sparsity into the model. Sparsity is achieved using ℓ_1 minimization instead of the more common ℓ_2 (least-squares). The use of an ℓ_1-sparse method gives rise to a reconstructed point set surface that consists mainly of smooth modes, with the residual of the objective function strongly concentrated near sharp features. Figure 1.5.1 demonstrates the effectiveness of our method on a scanned model.

The switch from ℓ_2 to ℓ_1 does not come without a price. ℓ_2 minimization requires solving least-squares equations, or simple generalizations of them. The ℓ_1 norm is not differentiable, so formulations associated with it are harder to solve. Nevertheless, the objective function is convex and as long as convexity is preserved, it is well understood how to solve the problem in hand. Usually, these method translate the problem to a series of regular least-squares equations. Thus, it is extremely important to solve these

[1]Available from `http://www.tau.ac.il/~stoledo/taucs/`.

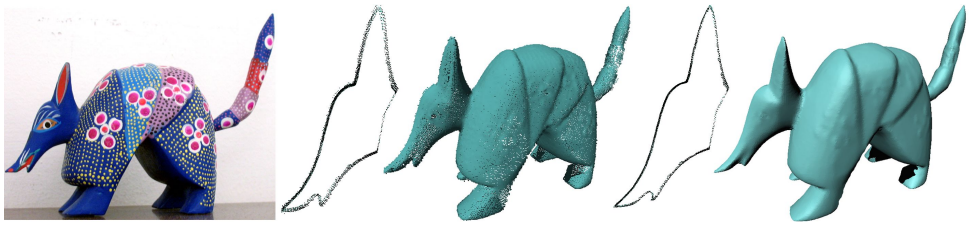


FIGURE 1.5.1. A demonstration of the effectiveness of the method presented in Chapter 5 on a scanned model. The Armadillo statue (left) is scanned generating a noisy point-cloud (middle). Using our method we are able to reconstruct it and recover its sharp features (right). Close-up view of a cross section of its head reveals the sharpness of the reconstructed surface.

equations quickly. As we shall see, the specific equations that arise by our method are particularly challenging.

Using an interior-point log-barrier solver with a customized preconditioning scheme, the solver for the corresponding convex optimization problem is competitive and the results are of high visual quality. The linear equations are solved using an iterative-direct hybrid solver that achieves running time similar to those achieved by a direct Cholesky factorization of the normal equations, but without sacrificing numerical stability. The techniques of section 1.3 are used to deal with matrix sparsity issues.

The results of this project were summarized in a paper which was submitted, and accepted subject to a minor revision, to the *ACM Transactions on Graphics* (the paper was co-authored with Andrei Sharf, Chen Greif, and Daniel Cohen-Or) [**21**].

Most aspects of this project belong to the realm of computer graphics. Nevertheless, the numerical aspect of solving the linear equations involved in the interior point method was crucial and is highly relevant to this thesis. I was involved both in the numerical aspect and in the computer-graphics aspect of the project. In this chapter I will focus on the numerical aspects of the project. For completeness, Chapter 5 follows the paper closely and its main foci is the computer graphics aspects.

1.5.1. Reconstruction Model. Our input is a *point cloud*, a set of positions $x_i \in X \subseteq \mathbb{R}^3$ with corresponding input orientation vectors $n_i \in N \subseteq \mathbb{R}^3$. The input cloud is denoted by $P(X, N)$. A point in the point cloud is the pair $p_i = (x_i, n_i)$. Our goal is to filter the point cloud so that it better describes a surface which is piecewise smooth with (a small number of) sharp edges connecting the smooth parts.

Since scanned information is generally noisy, we cannot assume that we have high quality point orientations. Hence, similarly to [**149**], we decouple orientations and positions. We solve first for the orientations, and then use them to compute consistent positions. We formulate both problems in a similar ℓ_1 nature, yielding a consistent solution.

Our orientation reconstruction model is based on the following key observation: smooth surfaces have smoothly varying normals, thus penalty functions should be defined on the surface normals. Therefore, we use pairwise normal differences as an estimator for shape smoothness. If two points p_i and p_j belong to the same smooth part, and their the distance is small enough in local feature size, then $n_i \approx n_j$. Furthermore, we assume that there is a minimum crease angle at singularities between smooth parts. Hence, at crease

angles where p_i and p_j belong to different smooth parts, the distance between n_i and n_j is above a small threshold τ. This discussion leads to an observation that the reconstructed normals should be such that only a small (i.e., sparse) number of local normal differences are large.

We use computed orientations to define consistent positions by assuming that the surface can be well approximated by local planes. Given a pair of neighbor points (p_i, p_j), we examine $n_{ij} \cdot (x_i - x_j)$, where n_{ij} is the average normal. Indeed, if both p_i and p_j belong to a smooth part of the surface then $n_{ij} \cdot (x_i - x_j) \approx 0$. At sharp features we expect $n_{ij} \cdot (x_i - x_j) \neq 0$.

We now formulate specific optimization problem based on these observations. The detailed derivations are in Chapter 5. We distinguish between the input point cloud $P(X^{in}, N^{in})$ and the output point cloud $P(X^{out}, N^{out})$.

Normals are reconstructed using the following optimization problem

$$N^{out} = \arg\min_{N} \sum_{(p_i,p_j)\in E} w_{ij} \left\| n_i - n_j \right\|_2 \quad \text{s.t.} \quad \forall_i \left\| n_i - n_i^{in} \right\|_2 \leq \gamma_n$$

where E is an adjacency set of $P(X, N)$ computed using k-nearest neighbors, w_{ij} is a set of predefined weights, and γ_n is a parameter proportional to the expected noise level.

Positions are projected along the reconstructed normals, that is

$$x_i^{out} = x_i^{in} + t_i n_i^{out} .$$

The vector t of projection distances is found by solving the following optimization problem

$$\arg\min_{t} \left\| At + f \right\|_1 \text{ s.t. } \left\| t \right\|_2 \leq \gamma_x$$

where $A \in \mathbb{R}^{|E|\times|P|}$ and $f \in \mathbb{R}^{|E|}$ ($|.|$ is the size). Each row of A correspond to a single pair $(p_i, p_j) \in E$ and is equal to

$$\left[\cdots \quad w_{ij}(n_{ij}^{out})^T n_i^{out} \quad \cdots \quad -w_{ij}(n_{ij}^{out})^T n_j^{out} \quad \cdots \right] .$$

Each index in f corresponds to a single pair $(p_i, p_j) \in E$ and is equal to

$$f_{ij} = (n_{ij}^{out})^T \cdot \left(x_i^{in} - x_j^{in} \right) .$$

γ_x is a parameter proportional to the expected noise level.

1.5.2. Solving the Minimization Problem. Our method amounts to solving a two convex optimization problems. These problems are nonlinear, but since they are convex they can be solved efficiently and reliably. Nevertheless, their non-linearity requires us to solve many linear system, which in turn must be done quickly. For the orientations phase we found that the method works well with a small value of k (number of neighbors). Thus, the use of an external package called `CVX` [**110**] was sufficient for our needs. For the positions phase the value of k must be much larger, and without a reliable solver for the positions phase only very small problems can be solved.

Our solver for the positions phase uses a log-barrier primal-only method. This involves solving a series of normal equations of the form

$$\left(A^T \Sigma_t A - g_{(\gamma)}^{-1} \mathrm{I}_{N\times N} + g_{(\gamma)}^{-2} r r^T \right) \Delta t = w_0, \tag{1.5.1}$$

where $g_{(\gamma)} < 0$ is a scalar, Σ_t is a diagonal matrix,$r \in \mathbb{R}^{|P|}$ is a dense vector and $w_0 \in \mathbb{R}^{|E|}$. Each solution is the search direction of a Newton iteration.

It is imperative to solve these equations efficiently, and this requires dealing with sparsity and conditioning issues. The matrix of the normal equations is symmetric positive definite, but for large scale problems it tends to be ill-conditioned, which in turn may result in an inaccurate search direction. Furthermore, the vector r is dense, whereas the original problem is sparse by nature. Therefore, factoring the matrix using the Cholesky decomposition may require a prohibitive amount of computational work and storage allocation. It is thus better to rewrite the 1.5.1 as a least-squares equation and to adopt an iterative solution technique.

We write

$$\tilde{A} = \begin{pmatrix} \Sigma_t^{1/2} A \\ (-g_{(\gamma)})^{-1/2} \mathrm{I}_{N\times N} \\ g_{(\gamma)}^{-1} r^T \end{pmatrix} .$$

It is easy to see that $\left(A^T \Sigma_t A - g_{(\gamma)}^{-1} \mathrm{I}_{N\times N} + g_{(\gamma)}^{-2} r r^T\right) = \tilde{A}^T \tilde{A}$. We find the search direction by solving the equivalent problem

$$\arg\min_{\Delta t} \left\| \tilde{A}\Delta t - w \right\|_2 ,$$

where $w_0 = \tilde{A}^T w$ (for brevity we omit here some details on w_0 and w).

The dense row in $\tilde{A}$ poses a problem because any direct method will overfill. To deal with this row we adopt a strategy suggested in section 1.3 (and discussed in Chapter 3). We remove the dense row from $\tilde{A}$; let us call the resulting matrix $\tilde{A}_0$. We then compute the Cholesky factorization of the *sparse* matrix associated with $\tilde{A}_0$:

$$R^T R = \tilde{A}_0^T \tilde{A}_0,$$

using CHOLMOD [**79**]. The R factor is used as a preconditioner for the augmented system associated with $\tilde{A}$, and now we apply LSQR [**166**]. The important point here is that removing r amounts to a rank-1 modification of the matrix that corresponds to the least squares operator. Therefore, only two iterations are needed for convergence in exact arithmetic. The matrix $\tilde{A}_0^T \tilde{A}_0$ may become very ill-conditioned, and sometimes this causes the Cholesky factorization to fail (it encounters a negative diagonal value because of inaccurate arithmetic). In such cases we use SuiteSparseQR [**75**] to compute a QR factorization instead and use R as a preconditioner.

Finally, we use one additional heuristic to speed up our solver. We have noticed that in some of the iterations, $\left|g_{(\gamma)}\right|$ tends to be considerably smaller than the maximum value on the diagonal of Σ_t. In those cases, LSQR on $\tilde{A}$ without a preconditioner tends to converge very quickly, because the singular values of $\tilde{A}$ are strongly clustered. We thus work on $\tilde{A}$ directly when conditioning allows for it (if $\|\Sigma_t\|_2 / \left|g_{(\gamma)}\right| \geq 10^3$), which saves the cost of a preconditioner solve.

The iterative method stops when the backward error drops below a certain threshold. This ensure backward stability relative to the desired level of accuracy. We use LSQR so the relevant condition number is $\kappa(\tilde{A})$ (and not $\kappa(\tilde{A}^T\tilde{A})$ as for the normal equations). This guarantees that we find a good search directions. This is is crucial for the log-barrier method, even if low convergence threshold are used. The threshold we used in our experiments is 10^{-8}. The accuracy of interior point method itself is based on a threshold.

TABLE 1. Solve time of the position phase on various models. Matrix size and #nnz columns refer to $\tilde{A}$.

Model	#points	k (#neighbors)	$\|E\|$	#nnz	#newton its.	Solve time
armadillo	99,416	10	545,086	1,286,528	24	46 sec
face	110,551	12	754,466	1,726,765	33	78 sec
Buddha	150,737	10	820,345	1,928,776	24	72 sec
funnel	201,655	8	860,831	2,116,119	22	66 sec
Escher	240,909	10	1,322,010	2,656,135	25	83 sec

We found that only a coarse level of accuracy was sufficient and further improvements were visually insignificant.

Table 1 shows the solve time of the positions phase on various model. Solve time was measured on a 64-bit Intel Core2 2.1 GHz using MATLAB. Other statistics are shown too. We see that although the matrix involved is large, and around 20-30 iteration are required, overall running time is not large. On average every linear equation is solved in less than 3 seconds. This shows that our scheme is highly effective.

1.5.3. Summary from a Numerical Point of View. This project is, at its core, a computer graphics project, with many insights in the computer graphics domain. Nevertheless, it has a non-trivial numerical aspect, from which we draw some informal conclusions.

- The method of row removal is useful in practice. Real-life sparse matrices sometimes have dense rows that should be accounted for.
- By using an iterative method we are sometimes able to exploit sparsity in cases where a direct method cannot. On the other hand, our matrices are small enough so that a direct method is often effective. A good hybrid gets the best of both worlds.
- The use of an iterative method also enables us to use Cholesky factorization of the normal form while not sacrificing stability. Many practitioners use the Cholesky factorization of the normal equations because it is faster on sparse matrices. They ignore the potential numerical problems involved. We can overcome these problems by using the Cholesky factorization as a preconditioner rather then a solver.
- Sometimes ill-conditioning causes the Cholesky factorization to fail (even if we only try to obtain a preconditioner). In such cases a QR factorization should be used. During the solution of the non-linear optimization problem this happened to us only a few times, but it did happen.

1.6. Experimental study of solving HPD systems using indefinite incomplete factorizations

Incomplete factorization is a popular preconditioning technique [**30, 184**]. Incomplete factorizations are attractive because they are simple to implement, generic in nature and they usually exhibit good performance. For Hermitian positive definite (HPD) systems the most natural incomplete factorization is incomplete Cholesky factorization. Unfortunately, incomplete Cholesky factorization may fail. While the Cholesky factorization LL^*

of a Hermitian positive definite matrix is guaranteed to exist, there is no such guarantee of the existence of an incomplete factorization of this form. The reason is that the errors introduced due to dropping entries from the factor may result in zero or negative diagonal values.

The traditional approach to address this problem is to force positive definiteness by modifying the factorization process. Benzi's survey [**30**] of these methods notes that the various techniques tend to fall into two categories: simple and inexpensive fixes that often result in low-quality preconditioners, or sophisticated strategies yielding high-quality preconditioners that are expensive to compute. Some techniques to circumvent possible breakdown of incomplete Cholesky factorization involve using an LDL^* factorization, where D is diagonal; this can prevent breakdown in the construction of the preconditioner, but the preconditioner might be indefinite. An indefinite preconditioner can be problematic, even when the original matrix is positive definite, because it can result in a breakdown of the symmetric Krylov-subspace solvers like CG [**124**] and MINRES [**165**] . In CG, the breakdown is caused by a division by zero if the M^{-1}-norm of the residual becomes zero; In MINRES, the breakdown is caused when trying to compute the square root of a negative value, when the algorithm computes the M^{-1}-norm of the new basis vector. Furthermore, the correctness proof of both CG and MINRES rely on the existence of a Cholesky factor of the preconditioner [**184**].

As a result, the conventional wisdom has been that alternate Krylov-subspace methods, such as symmetric QMR [**101, 102**], GMRES [**185**], or BiCGStab [**217**], etc. must be used if the preconditioner is indefinite. However, using GMRES is expensive due to the long recurrence (expensive orthogonalization steps and a high memory requirement). Algorithms like QMR or BiCGStab do not minimize a norm of the residual or a norm of the error as GMRES, CG, and MINRES do. In general, it is not possible to get both optimality and a short recurrence with a non-symmetric method [**95**].

Although not very well known, a there exists a variant of CG which allows an indefinite preconditioner [**14**]. We will refer to this variant as PCG-ODIR[2]. To the best of our knowledge this variant has not been experimentally compared to GMRES or QMR when an indefinite matrix is used to precondition an HPD system (the only implementation of PCG-ODIR that we are aware of is [**121**]). In chapter 6 we experimentally explore this case and develop a new variant of PCG-ODIR that addresses the numerical problems demonstrated in the experiments. We also propose a new Krylov-subspace variant of MINRES that guarantee convergence and allow an indefinite preconditioner to be used.

1.6.1. Conjugate Gradients as a Lanczos process. At its core the Conjugate Gradients method generates at each iteration an A-conjugate basis for the Krylov subspace. That is

$$\operatorname{span}\{q_1, q_2, \ldots, q_n\} = \mathcal{K}_n(A, b)$$

and

$$Q_n^* A Q_n = D_n$$

[2]To be more precise, this variant is called simply PCG in [**14**]. In many cases the name PCG is used for the preconditioned version of the traditional CG, so we decided to use the name PCG-ODIR because this variants uses ODIR (unlike the the traditional preconditioned CG which uses OMIN).

where

$$Q_n = \left[\begin{array}{cccc} q_1 & q_2 & \cdots & q_n \end{array} \right]$$

and D_n is a diagonal matrix. Once we have found an A-conjugate basis the Conjugate Directions method can be used to produce an optimal A-norm approximation (see §7 in Shewchuk's tutorial [**192**]). The classical CG method couples the creation of the basis with the application of the conjugate directions method in a clever way. A preconditioner can be used but it must be positive definite, otherwise the algorithm may fail (because of possible division by zero if the M^{-1}-norm of the residual becomes zero), and in any case the the correctness proof of CG relies on the existence of a Cholesky factor of the preconditioner [**184**].

It is well-known that the CG iteration can be formulated instead as a Lanczos process [**165**]. Using the Lanczos iteration we find an orthonormal basis U_n such that $U_n^* A U_n = T_n$ where T_n is tridiagonal and HPD. Let $T_n = R_n^* R_n$ be a Cholesky factorization of T_n and define

$$Q_n = U_n R_n^{-1} .$$

The columns of Q_n form an A-orthonormal basis. Unfortunately, we have not advanced towards a indefinitely-preconditioned version of CG: to use the Lanczos version of CG the preconditioner must be positive definite.

What is less known is that there is another, more straight-forward and robust, formulation of CG as a Lanczos process. Instead of using the regular Lanczos iteration we can use Lanczos with a non standard inner product. Let U be an HPD matrix such that UA is Hermitian. The U-conjugate Lanczos iteration generates basis vectors $q_1, q_2, \ldots$ such that

(1) $Q_n^* U Q_n = I_{n \times n}$,
(2) $Q_n^* U A Q_n = T_n$ (T_n is tridiagonal).

By selecting $U = A$ we find an A-conjugate basis. This version of CG can be used with an indefinite preconditioner M: We apply the A-conjugate Lanczos iteration on $M^{-1}A$.

This version of CG appears in [**14**] under the name PCG-ODIR.

1.6.2. Performance of PCG-ODIR. We have implemented PCG-ODIR and compared it to other algorithms (GMRES, QMR, BiCGStab) that work with indefinite preconditioners. The results of these comparisons appear in Chapter 6. Unfortunately, the performance is rather disappointing. PCG-ODIR is considerably faster than BiCGStab, but it is only slightly faster than QMR. It is faster than GMRES only when both use the same amount of memory, but not when both use the same preconditioner (in which case GMRES uses more memory).

A simple experiment shows that the results can be attributed to inexact arithmetic. Consider a version of PCG-ODIR where we use a long recurrence (which is more stable numerically) instead of a short recurrence (i.e., Arnoldi iteration instead of Lanczos iteration). Under exact arithmetic both the short recurrence and the long recurrence version of PCG-ODIR are equivalent. But, as Table 2 suggests, under inexact arithmetic this is not the case. We see that the long recurrence version of PCG-ODIR (labeled "ORTHODIR" the name used in [**14**]) does considerably fewer iterations then the short-recurrence version. We also see that CG sometimes performs many more iterations than ORTHODIR, but happens rarely. In particular, whenever CG performs well so does PCG-ODIR. This

TABLE 2. Numerical stability: comparing full conjugation to local conjugation. In the OILPAN (NO PRECOND) instance, the convergence threshold was set to 10^{-5}.

Matrix	Droptol	Precond Definite?	PCG-ODIR	ORTHODIR	CG
CFD1	2×10^{-3}	NO	125 its	77 its	N/A
CFD1	4.5×10^{-4}	YES	85 its	69 its	85 its
CFD1	2×10^{-4}	YES	48 its	47 its	48 its
OILPAN	NO PRECOND	N/A	783 its	747 its	783 its
OILPAN	8×10^{-3}	NO	441 its	142 its	N/A
OILPAN	1.5×10^{-3}	NO	63 its	58 its	N/A
OILPAN	8×10^{-4}	YES	39 its	42 its	39 its
PWTK	4×10^{-3}	NO	149 its	103 its	N/A
PWTK	1×10^{-3}	NO	77 its	55 its	N/A
PWTK	8×10^{-4}	YES	61 its	55 its	61 its

suggest that the numerical problems are due to short recurrence, and it is related to the quality of the preconditioner.

To summarize, our initial set of experiments showed that PCG-ODIR fails to fulfill its potential. Due to numerical stability issues more iterations are needed. This is a general problem with CG, but it is aggravated when the preconditioner is of low quality (which is usually the case if it is indefinite). We now turn to an explanation of the problem and a potential solution.

1.6.3. Selective orthogonalization. The formulation of CG as a Lanczos process was already helpful for allowing an indefinite preconditioner. We now use it to explain and deal with numerical stability issues.

The basis vectors $q_1, q_2, \ldots$ are supposed to be A-conjugate, but due to rounding errors they lose conjugacy. As long as the loss of conjugacy is bounded, that is $\|I_{n\times n} - Q_n^* A Q_n\|_2 \leq \delta$ for some small δ, we will find iterates that are close to their ideal counterparts under exact arithmetic. Loss of conjugacy is not too severe if a long recurrence is used, but using a long recurrence is wasteful in memory and computation, and usually requires a restart at some stage. It is preferable to find a more economical method.

We propose the use of *selective orthogonalization* [**167**]. Instead of orthogonalizing the current iterate with respect to all previous basis vectors, we (incrementally) form a small set of vectors, say 5 vectors, and orthogonalize with respect to them (and with respect to the last two iterates, as in the short-recurrence form). This small set of vectors should be carefully selected so that it will restore A-conjugacy to Q_n as much as possible.

A celebrated result by Paige [**164**] shows how to find such vectors for the regular Lanczos process. Let $q_1, q_2, \ldots$ be the vectors formed by the Lanczos process on a Hermitian matrix A and let T_n be the tridiagonal matrix $T_n = Q_n^* A Q_n$. Let w_j $(j = 1, \ldots, n)$ be the eigenvectors of T_n and let $z_j = Q_n w_j$ be the corresponding Ritz vectors. Paige showed that under inexact arithmetic there are constants $\gamma_{j,n+1}$ of order of the rounding unit such

that

$$z_j^* q_{n+1} = \frac{\gamma_{j,n+1}}{|\beta_{n+1} \mathrm{e}_j^T w_j|}$$

where e_j is the jth identity vector and β_{n+1} is a scalar computed during the Lanczos iteration. An iterate has a strong direction only if $|\beta_{n+1}\mathrm{e}_j^T w_j|$ is small, which also means that the Ritz vector has converged or almost converged. Selective orthogonalization consists of saving converged and nearly converged Ritz vectors and orthogonalizing the Lanczos iterates against them.

We have formulated PCG-ODIR as a Lanczos process with a non-standard inner product, which opens the door for the use of selective orthogonalization. We call PCG-ODIR with selective orthogonalization IP-CG (Indefinitely Preconditioned CG).

1.6.4. Indefinitely Preconditioned MINRES. The MINRES algorithm can be used to solve $Ax = b$ for any Hermitian matrix, and a preconditioner can be used as long as it is Hermitian positive definite. Using the A-conjugate Lanczos iteration we developed a variant of MINRES that requires the opposite: any Hermitian preconditioner can be used as long as the matrix is positive definite. Numerical behavior of this algorithm can be improved using selective orthogonalization. We call the resulting algorithm IPMINRES. We leave the exact details for chapter 6.

1.6.5. Numerical experiments. We have conducted a detailed numerical study of PCG-ODIR, IP-CG and IPMINRES and compared them to older algorithms (GMRES, QMR, BiCGStab). The results are reported in chapter 6.

Our experiments show that PCG-ODIR, IP-CG and IPMINRES converge in fewer iterations than QMR and BiCGStab. They are the fastest methods that use a short recurrence. Our experiments also show that even with selective orthogonalization numerical problems prevent IP-CG from fulfilling their full theoretical potential and GMRES usually converges in fewer iterations. Nevertheless, PCG-ODIR, IP-CG and IPMINRES often outperform GMRES by using a denser and more accurate incomplete factorization to compensate for the extra memory that GMRES requires. IP-CG is faster than PCG-ODIR, but PCG-ODIR is more economical in memory (which can be used for a denser preconditioner).

1.7. Randomized algorithms for estimating the trace of an implicit symmetric positive semi-definite matrix

In chapter 7 we move to the second theme of this thesis: randomization. Unlike previous chapters, chapter 7 does not deal with solving linear equations, but instead explores the use of randomized algorithms for finding the trace of an implicit matrix, that is, a matrix represented as a function. More specifically, we explore algorithms that compute the trace of a matrix represented as a quadratic form $x \mapsto x^T A x$. In some application areas, like lattice Quantum Chromodynamics, one often needs to compute the trace of a function of a large matrix, $\mathrm{trace}(f(A))$. Explicitly computing $f(A)$ for large matrices is not practical, but computing the bilinear form $x^T f(A) x$ for an arbitrary x is feasible [**25, 26**].

The standard approach for computing the trace of an implicit function is Monte-Carlo simulation, where the trace is estimated by $\frac{1}{M}\sum_{i=1}^{M} z_i^T A z_i$, where the z_i are random vectors. The original method is due to Hutchinson [**130**]. Although this method has been

improved over the years ([**29, 131, 224**]) , no paper to date has presented a theoretical bound on the number of samples required to achieve an ϵ-approximation of the trace. Chapter 7's most important contribution is in providing rigorous bounds on the number of Monte-Carlo samples required to achieve a maximum error ϵ with probability at least $1-\delta$ in several trace estimators. We also make a few other contribution: a specialized bounds for the case of projection matrices, which are important in certain applications, we offer a new trace estimator and we experimentally evaluate the convergence of trace estimators on a few interesting matrices.

Chapter 7 is based on a paper that was submitted for publication [**24**] in April 2010.

1.7.1. Hutchionson's method and related work. The problem of estimating the trace of an implicit matrix has been explored since 1989. Before our work, the standard Monte-Carlo method for estimating the trace of an implicit method is due to Hutchinson [**130**], who proved the following Lemma.

Lemma 1.7.1. *Let A be an $n \times n$ symmetric matrix with* $\operatorname{trace}(A) \neq 0$. *Let z be a random vector whose entries are i.i.d Rademacher random variables (*$\Pr(z_i = \pm 1) = 1/2$*). $z^T Az$ is an unbiased estimator of* $\operatorname{trace}(A)$ *i.e.,*

$$\mathrm{E}(z^T Az) = \operatorname{trace}(A)$$

and

$$\operatorname{Var}(z^T Az) = 2\left(\|A\|_F^2 - \sum_{i=1}^{n} A_{ii}^2\right).$$

Lemma 7.2.1 does not give a rigorous bound on the number of samples/matrix multiplications, and the variance term is not small enough to use Chebyshev's inequality to derive a bound. This difficulty carries over to applications of this method, such as [**25, 26**]. Hutchinson's method has been improved over the years, but the improvements do not appear to have addressed this issue. Wong et al. [**224**] suggest using test vectors z that are derived from columns of an Hadamard matrix. Iitaka and Ebisuzaki [**131**] generalized Hutchinson's estimator by using *complex* i.i.d variables with unit magnitude; they showed that the resulting estimator has lower variance than Hutchison's (but the computation cost is also higher). Silver and Röder [**196**] use Gaussian i.i.d variables, but without any analysis. Bekas et al. [**29**] focus on approximating the actual diagonal values, also using vectors derived from an Hadamard matrix.

1.7.2. Summary of results. We analyze Hutchinson's estimator, the Gaussian estimator (i.e., using Gaussian i.i.d) and an additional new estimator. We analyze the various estimators based using three different metrics. The first metric is the variance of the estimator. This is the only metric analyzed by Lemma 1.7.1 and in previous work. We believe that this metric does not reveal enough information to a practitioner. A better way to analyze an estimator is to bound the number of samples required to guarantee that the probability of the relative error exceeding ϵ is at most δ. Let A be a symmetric positive semi-definite matrix. A randomized trace estimator T is an (ϵ, δ)-*approximator* of $\operatorname{trace}(A)$ if

$$\Pr\left(|T - \operatorname{trace}(A)| \leq \epsilon \operatorname{trace}(A)\right) \geq 1 - \delta\,.$$

This is our second metric. We also analyze a third metric, the number of random bits used by the algorithm, i.e. the randomness of the algorithm. The trace estimators are

highly parallel; each Rayleigh quotient can be computed by a separate processor. If the number of random bits is small, they can be precomputed by a sequential random number generator. If the number is large (e.g., $O(n)$ per Rayleigh quotient), the implementation will need to use a parallel random-number generator. This concern is common to all Monte-Carlo methods.

We analyze five different trace estimators.

Definition 1.7.2. A *Gaussian trace estimator* for a symmetric positive-definite matrix $A \in \mathbb{R}^{n\times n}$ is

$$G_M = \frac{1}{M}\sum_{i=1}^{M} z_i^T A z_i \ ,$$

where the z_i's are M independent random vectors whose entries are i.i.d standard normal variables.

The Gaussian estimator does not constrain the 2-norm of the z_i's; it can be arbitrarily small or large. All the other estimators that we analyze normalize the quadratic forms by constraining $z^T z$ to be equal to n. This property alone allows us to prove below a general convergence bound.

Definition 1.7.3. A *normalized Rayleigh-quotient trace estimator* for a symmetric positive semi-definite matrix $A \in \mathbb{R}^{n\times n}$ is

$$R_M = \frac{1}{M}\sum_{i=1}^{M} z_i^T A z_i \ ,$$

where the z_i's are M independent random vectors such that $z_i^T z_i = n$ and $\mathrm{E}(z_i^T A z_i) = \mathrm{trace}(A)$.

The second estimator we analyze is Hutchinson's.

Definition 1.7.4. An *Hutchinson trace estimator* for a symmetric positive-definite matrix $A \in \mathbb{R}^{n\times n}$ is

$$H_M = \frac{1}{M}\sum_{i=1}^{M} z_i^T A z_i \ ,$$

where the z_i's are M independent random vectors whose entries are i.i.d Rademacher random variables.

The first two estimators use a very large sample spaces. The Gaussian estimator uses continuous random variables, and the Hutchinson estimator draws z from a set of 2^n vectors. Thus the amount of random bits required to form a sample is $\Omega(n)$. Our third estimator samples from a set of n vectors, so it only needs $O(\log n)$ random bits per sample. This estimator samples from a smaller family by estimating the trace in a more direct way: it samples the diagonal itself. The average value of a diagonal element of A is $\mathrm{trace}(A)/n$. So we can estimate the trace by sampling a diagonal element and multiplying the result by n. This corresponds to sampling a unit vector from the standard basis and computing the Rayleigh quotient.

Definition 1.7.5. A *unit vector estimator* for a symmetric positive-definite matrix $A \in \mathbb{R}^{n\times n}$ is

$$U_M = \frac{n}{M}\sum_{i=1}^{M} z_i^T A z_i \ ,$$

where the z_i's are M independent uniform random samples from $\{e_1, \ldots, e_n\}$.

In contrast to previous methods, the quadratic forms in the unit-vector estimator do not depend in any way on the off-diagonal elements of A, only on the diagonal elements. Therefore, the convergence of U_M is independent of the off-diagonal elements. The distribution of diagonal elements does influence, of course, the convergence to $\mathrm{trace}(A)/n$. For some matrices, this method must sample all the diagonal elements for U_M to be close to $\mathrm{trace}(A)$. For example, if A has one huge diagonal element, the average is useless until we sample this particular element. On the other hand, if all the diagonal elements are the same, the average converges to the exact trace after one sample.

Our last estimator is a variant of the unit vector estimator that uses randomization to address this difficulty. Instead of computing the trace of A, it computes the trace of $\mathcal{F}A\mathcal{F}^T$ where $\mathcal{F}$ is a unitary matrix. Since the *mixing matrix* $\mathcal{F}$ is a unitary, $\mathrm{trace}(A) = \mathrm{trace}(\mathcal{F}A\mathcal{F}^T)$. We construct $\mathcal{F}$ using a randomized algorithm that guarantees with high probability a relatively flat distribution of the diagonal elements of $\mathcal{F}A\mathcal{F}^T$.

Definition 1.7.6. A *random mixing matrix* is a unitary $\mathcal{F} = FD$, where F and D be n-by-n unitary matrices. The matrix F is a fixed unitary matrix called the *seed* matrix. The matrix D is a unitary random diagonal matrix with diagonal entries that are i.i.d Rademacher random variables: $\Pr(D_{ii} = \pm 1) = 1/2$.

Definition 1.7.7. A *mixed unit vector estimator* for a symmetric positive semi-definite matrix $A \in \mathbb{R}^{n\times n}$ is

$$T_M = \frac{n}{M}\sum_{i=1}^{M} z_i^T \mathcal{F}A\mathcal{F}^T z_i \ ,$$

where the z_i's are M independent uniform random samples from $\{e_1, \ldots, e_n\}$, and $\mathcal{F}$ is a random mixing matrix.

Table 3 summarizes the results of our analyses. The proofs are in sections 7.5-7.8 of chapter 7. The smallest variance is achieved by Hutchinson's estimator, but the Gaussian estimator has a better (ϵ, δ) bound. Unit vector estimators use the fewest random bits, but have an (ϵ, δ) bound that is worse than that of Gaussian and Hutchinson's estimators.

We also prove a result regarding convergence of the Gaussian trace estimator on projection matrix (i.e., a matrix with only 0 and 1 eigenvalues; the trace is equal to the ran). In this special case only $O(\mathrm{rank}(A)\log(2/\delta))$ samples (where δ is a probability of failure; there is no dependence on ϵ). Finding the rank of a projection matrix is useful for computing charge densities (in electronic structures calculations) without diagonalization [**29**].

Lemma 1.7.8. *Let $A \in \mathbb{R}^{n\times n}$ be a projection matrix, and let $\delta > 0$ be a failure probability. For $M \geq 24\,\mathrm{rank}(A)\ln(2/\delta)$, the Gaussian trace estimator G_M of A satisfies*

$$\Pr(\mathrm{round}(G_M) \neq \mathrm{rank}(A)) \leq \delta \, .$$

Estimator	Variance of one sample	Bound on # samples for an (ϵ, δ)-approx	Random bits per sample
Gaussian	$2\|A\|_F$	$20\epsilon^{-2}\ln(2/\delta)$	infinite; $\Theta(n)$ in floating point
Normalized Rayleigh-quotient	-	$\frac{1}{2}\epsilon^{-2}n^{-2}\operatorname{rank}^2(A)\ln(2/\delta)\kappa_f^2(A)$	-
Hutchinson's	$2\left(\|A\|_F^2 - \sum_{i=1}^n A_{ii}^2\right)$	$6\epsilon^{-2}\ln(2\operatorname{rank}(A)/\delta)$	$\Theta(n)$
Unit Vector	$n\sum_{i=1}^n A_{ii}^2 - \operatorname{trace}^2(A)$	$\frac{1}{2}\epsilon^{-2}\ln(2/\delta)r_D^2(A)$ $r_D(A) = \frac{n\cdot\max_i A_{ii}}{\operatorname{trace}(A)}$	$\Theta(\log n)$
Mixed Unit Vector	-	$8\epsilon^{-2}\ln\left(4n^2/\delta\right)\ln(4/\delta)$	$\Theta(\log n)$

TABLE 3. Summary of results: quality of the estimators under different metrics. The proofs appear in chapter 7.

1.7.3. Consequences. From a theoretical point of view, the (ϵ, δ) bound for the Gaussian estimator seems good: for fixed ϵ and δ, only $O(1)$ samples are needed. However, the ϵ^{-2} factor in the bound implies that the number of samples may need to scale exponentially with the number of bits of accuracy (the number of samples in the bound scales exponentially with $\log_{10}\epsilon^{-1}$). Therefore, in applications that require only a modest ϵ, say $\epsilon = 0.1$, the Gaussian estimator is good. But in applications that require a small ϵ, even $\epsilon = 10^{-3}$, the number of samples required may be too high.

We also conducted numerical experiments on a few interesting matrices. Convergence to a small error is slow, and close to the bound for the Gaussian estimator, so it appears that this bound is tight or almost-tight. For example, see Figure 1.7.1; more experiments are reported in chapter 7. Our experiments also did not show a considerable difference in practice between the Gaussian, Hutchinson and mixed unit vector estimators.

Randomized trace estimators quickly give a crude estimate of the trace (correct to within 10% or 1%, say), but they require a huge number of samples to obtain a very accurate estimate. The ϵ^{-2} factor in the bound is common to many randomized algorithms in numerical linear algebra, and is, unfortunately, often unsatisfactory for applications. In the next chapter we present a linear solver that uses a randomized algorithm as an inexact approximator within the context of a deterministic iterative solver. The overall strategy yields a solver that is both fast and accurate. We believe that the most promising strategy for using randomized algorithm in numerical linear algebra is by leveraging traditional algorithms using (inexact) randomized algorithms. Unfortunately, we are not aware of a suitable iterative algorithm for trace computations.

1.8. Engineering a random-sampling numerical linear algebra algorithm

The two themes of this thesis merge in chapter 8. Chapter 8 describes an high-performance robust solver for dense overdetermined least-squares systems that is dramatically faster than LAPACK's solver. These results also appear in a paper published in the *SIAM Journal on Scientific Computing* [**18**].

The project started from the understanding that so far, randomized algorithm have had only a limited impact in numerical linear algebra. No practical linear solver has been demonstrated so far, although theoretical algorithms have been suggested. The

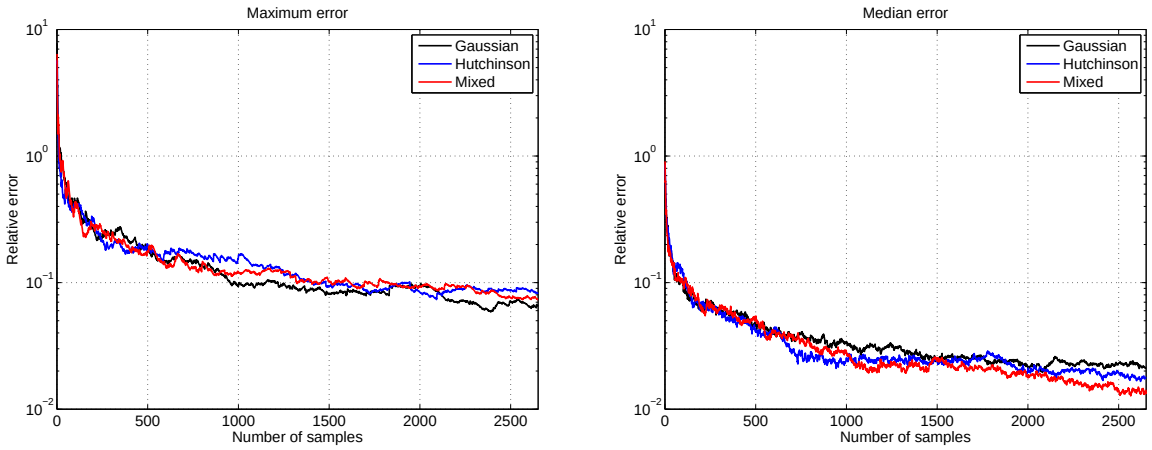


FIGURE 1.7.1. Convergence of the estimators on a matrix of order 100,000 whose elements are all 1. The graph on the left shows the maximum error during 100 runs of the algorithm, and the graph on the right the median of the 100 runs.

performance of algorithms and codes on modern computers is a complex issue, one that cannot always be fully captured by theoretical complexity analysis. Performance depends on many issues and can really only be truly measured by developing and testing the software on different platforms. Furthermore, as we have seen in the previous section, running time of randomized algorithm usually depend on the required accuracy, and often the running time is exponential in the number of required accuracy bits.

Our goal is to show that thorough careful engineering of a new least-squares solver, which we call Blendenpik, and through extensive analysis and experimentation, randomized algorithm can beat state-of-the-art numerical linear algebra libraries *in practice*. To beat traditional numerical linear algebra algorithms, randomized algorithms must be combined with older, well-proved deterministic techniques, mainly iterative and preconditioning techniques. For example, Blendenpik uses a classical preconditioned iterative linear solver, but the preconditioner is built using a randomized algorithm.

Blendenpik outperforms LAPACK by large factors on realistic problem sizes (i.e., not huge) while achieving similar accuracy. Our solver scales better than LAPACK's, so the performance difference grows with problem size. We believe that the results reported in Chapter 8 show the potential of random-sampling algorithms, and suggest that randomized algorithms should be considered for use in state-of-the-art numerical linear algebra libraries.

1.8.1. Overview of the Algorithm. Our solver minimizes large highly overdetermined systems $x = \arg\min_x \|Ax - b\|_2$ where $A \in \mathbb{R}^{m \times n}$ and $b \in \mathbb{R}^m$. Traditionally, A is factored using, say, a QR factorization, at a cost of $\Theta(mn^2)$. A simple random-sampling approach is to factor only a randomly-selected subset of A's rows. That is, we randomly form an $r \times m$ sampling matrix $\mathcal{S}$ and factor $\mathcal{S}A = QR$. This factorization cannot be used to accurately minimize the sum of squares, but R can be used as a preconditioner for an iterative solver like LSQR [**166**]. Obviously, if $r = m$ then $\mathcal{S}A = A$ and LSQR

will converge in one iteration, so with enough samples, R is a good preconditioner. How many samples do we really need?

For matrices with random independent uniform entries, $r = 4n$ usually yields a good preconditioner, but this is not true for all matrices (e.g., one needs $r = \Omega(m)$ for the m-by-n identity). It turns out that the number of samples needed is related to the *coherence* of the matrix [**54**].

Definition 1.8.1. Let A be an $m \times n$ full rank matrix and let U be an $m \times n$ matrix whose columns form an orthonormal basis for the column space of A. The *coherence* of A is defined as

$$\mu(A) = \max \|U_{i,*}\|_2^2 .$$

The coherence of a matrix is always smaller than 1 and bigger than n/m. Note that it does not depend on the condition number of A. Random sampling yields a good preconditioner on incoherent matrices (matrices with small coherence). For example, if $\mu(A) = n/m$, then only $\Theta(n \log n)$ rows need to be sampled to obtain a good preconditioner. Unfortunately, we cannot always guarantee a bound on $\mu(A)$ in advance.

Drineas et al. [**89**] suggest to address this difficulty with a row-mixing strategy: they multiply A from the left by $\mathcal{F} = FD$ where D is a random diagonal matrix with ± 1 on it's diagonal and F is an Hadamard matrix. Multiplying $\mathcal{F}$ by A can be done in $O(mn \log m)$ operations using the fast Walsh-Hadamard transform. It can be shown that with high probability, $\mu(\mathcal{F}A) = O((n/m) \log m)$. At this point, random sampling can be used to form a preconditioner. Nguyen et al. [**162**] show that F can be replaced by the normalized matrix of Fourier-type transforms (DFT[3], DCT[4], DHT[5] and others).

To summarize, the algorithm proceeds as follows. A random diagonal matrix D with ± 1 on its diagonal with equal probability is formed. Either DCT, DHT or the fast Walsh-Hadamard transform are applied to DA. We then sample γn (γ is a parameter) rows from FDA to form a new matrix. A reduced QR factorization of the matrix is found and R is used as a preconditioner for LSQR. Assuming a constant number of LSQR iterations, the total cost is $\Theta(mn \log m + n^3)$ operations.

1.8.2. Numerical experiments. We have implemented the least-squares solver and conducted extensive numerical experiments. Here we preview the results, while most of the results appear in Chapter 8. The benchmark code is LAPACK's function DGELS. The code is implemented in C and uses BLAS routines for basic matrix operations. We set LSQR's convergence threshold to 10^{-14}, which is close to $\epsilon_{\text{machine}}$. We did not set it lower to avoid stagnation of the iterative method close to convergence. We measured the running times on a machine with two AMD Opteron 242 processors (we only used one) running at 1.6 GHz with 8 GB of memory. Matrices were generated using MATLAB's RAND function (random independent uniform entries).

Figure 1.8.1 compares the running times of the new solver and LAPACK for increasingly larger matrices. The y-axis is the ratio of LAPACK's running time to the new solver's running time. All the matrices are well conditioned. For tiny matrices LAPACK is faster,

[3]Discrete Fourier Transform
[4]Discrete Cosine Transform
[5]Discrete Hartley Transform

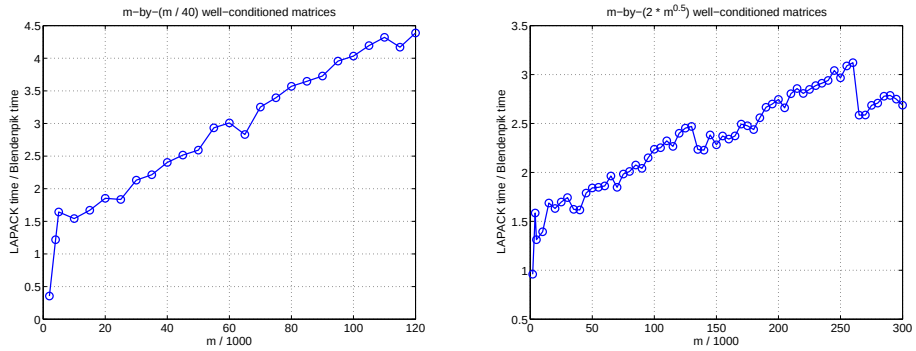


FIGURE 1.8.1. Comparison between LAPACK and the new solver for increasingly larger matrices. Graphs show the ratio of LAPACK's running time to the new solver's running time on random matrices with two kinds of aspect ratios

but the new algorithm is faster even on fairly small matrices, and the ratio grows with matrix size. On the largest matrices we tested, the new solver is about four times faster than LAPACK. In other experiments (reported in Chapter 8) we have found that even on highly incoherent matrices the solver is reliable and faster than LAPACK, although by smaller factors than the results reported in Figure 1.8.1.

1.8.3. Conclusions. Blendenpik is competitive in the usual metrics of numerical linear algebra, and it demonstrates that randomized algorithms can be effective in general-purpose numerical linear algebra software. We have not encountered cases of large dense matrices where the solver fails to beat LAPACK, even in hard test cases, and we have not encountered large variance in running time of the algorithm on a given matrix. Even the convergence rate in the iterative phase is stable and predictable (unlike many algorithms that use an iterative method). These results indicate the potential of randomized algorithms, and that the potential can be realized by combining them with older, more traditional techniques.

CHAPTER 2

Combinatorial Preconditioners for Scalar Elliptic Finite-Element Problems

2.1. Introduction

This chapter[1] presents novel combinatorial preconditioners for scalar elliptic finite-element problems based on element-by-element symmetric diagonally dominant approximations. Symmetric diagonally dominant matrices are relatively easy to precondition. This observation has led two groups of researchers to propose linear solvers that are based on element-by-element approximation of a given coefficient matrix by a diagonally dominant matrix. The diagonally dominant approximation is then used to construct a preconditioner, which is applied to the original problem. Before we describe their proposals and our new contributions, however, we describe the formulation of the problem.

These existing techniques, as well as our new algorithm, use a given finite-element discretization of the following problem: Find $u\colon \Omega \longrightarrow \mathbb{R}$ satisfying

$$\begin{array}{rcll} \nabla \cdot (\theta(z)\nabla u) & = & -f & \text{on } \Omega \\ u & = & u_0 & \text{on } \Gamma_1 \ , \\ \theta(z)\partial u/\partial n & = & g & \text{on } \Gamma_2 \ . \end{array} \tag{2.1.1}$$

The domain Ω is a bounded open set of $\mathbb{R}^d$ and Γ_1 and Γ_2 form a partition of the boundary of Ω. The *conductivity* θ is a spatially varying d-by-d symmetric positive definite matrix, f is a scalar *forcing function*, u_0 is a *Dirichlet boundary condition* and g is a *Neumann boundary condition*.

We assume that the discretization of (2.1.1) leads to an algebraic system of equations

$$Kx = b \ .$$

The matrix $K \in \mathbb{R}^{n\times n}$ is called a *stiffness matrix*, and it is a sum of *element matrices*, $K = \sum_{e\in E} K_e$. Each element matrix K_e corresponds to a subset of Ω called a *finite element*. The elements are disjoint except perhaps for their boundaries and their union is Ω. We assume that each element matrix K_e is symmetric, positive semidefinite, and zero outside a small set of n_e rows and columns. In most cases n_e is uniformly bounded by a small integer (in our experiments n_e is 4 or 10). We denote the set of nonzero rows and columns of K_e by $\mathcal{N}_e$.

For the problem (2.1.1), all the element matrices are singular with rank $n_e - 1$ and null vector $\begin{bmatrix}1 & 1 & \cdots & 1\end{bmatrix}^T$. This is a key aspect of techniques for approximating element

[1]The results in this chapter also appear in a paper co-authored with Doron Chen, Gil Shklarski and Sivan Toledo which was published in the *SIAM Journal on Matrix Analysis and Applications* [**16**]; several examples and minor proofs appears in a technical report [**22**].
Most of the research was carried out by myself and another PhD student from Tel-Aviv University, Gil Shklarski, in a collaborative effort. These results also appear in Gil's thesis [**193**].

matrices by diagonally dominant ones: the methods only works when element matrices are either nonsingular or are singular with rank $n_e - 1$ and null vector $\begin{bmatrix}1 & 1 & \cdots & 1\end{bmatrix}^T$.

While the general approach of element-by-element (EBE) preconditioning was proposed as early as [**129**], Reitzinger and his collaborators were the first to propose element-by-element approximation by symmetric and diagonally dominant (SDD) matrices [**118, 174, 142**]. They proposed two element-approximation techniques. In one technique, the approximation problem is formulated as an optimization problem, in which one tries to minimize the generalized condition number of a given element matrix K_e and an SDD matrix L_e; a generic optimization algorithm is then used to find L_e. The second technique uses symbolic algebra to formulate the approximation problem, and uses symbolic-algebra algorithms to find L_e. Both methods are quite expensive and slow. A row or a column that is zero in K_e is also zero in L_e, so the L_e's are also very sparse. Once the L_e's are found, they are assembled. When all the L_e's are SDD, their sum L is also SDD. The matrix L is then used to construct an algebraic multigrid solver, which is used as a preconditioner for K.

Boman, Hendrickson, and Vavasis proposed a different element-by-element approximation technique [**44**]. They used geometric information about the elements and the values of θ to directly construct L_e. The approximation algorithm is inexpensive. They show that under certain conditions on the continuous problem and on the finite-element mesh, the approximations are all good. They proposed to use L to construct a combinatorial preconditioner rather then apply multigrid to L. This proposal was based on the observation that over the last decade, several provably good combinatorial graph algorithms for constructing preconditioners for SDD matrices have been developed [**215, 112, 113, 41, 200, 201, 93, 202, 152**]. Some of them, as well as some combinatorial heuristics, have been shown to be effective in practice [**63, 135, 173, 175, 100, 172**].

In this chapter, we extend this paradigm in two ways. First, we propose a novel, effective, and purely algebraic method for approximating an element matrix K_e by an SDD matrix L_e. Our approximation algorithm is relatively inexpensive and provably good: the spectral distance between K_e and L_e is within a $n_e^2/2$ factor of the best possible for an SDD approximation of K_e, where n_e is the number of nonzero rows and columns in K_e. In particular, this means that our algorithm produces good approximations whenever the algorithm of Boman et al. does. Furthermore, there exist element matrices that are ill conditioned and that are far from diagonal dominance (in the sense that a small perturbation of their entries cannot make them diagonally dominant), which our algorithm approximates well.

Vavasis has shown [**219**] that some of the results in this chapter can be used to find an optimal approximation of an element matrix K_e by an SDD matrix L_e. Since his method is computationally expensive it is not clear if it is effective in a practical solver. Our code uses our provably good, cheap to compute, but suboptimal approximation instead.

Our second contribution to this paradigm is a technique to handle problems in which some of the element matrices cannot be well approximated by SDD matrices. This may arise because of anisotropy in θ, or because of ill-shaped elements, for example. Our algorithm splits the elements into two sets: the set $E(t)$ in which L_e is a good approximation of K_e, and the rest (where t is a parameter that determines how good we require approximations to be.) We then scale and assemble the good element-by-element approximations to form $L = \sum_{e \in E(t)} \alpha_e L_e$. Next, we use a combinatorial graph algorithm to

construct an easy-to-factor approximation M of L. Finally, we scale M and add γM to the inapproximable elements, to form $\gamma M + \sum_{e \notin E(t)} K_e$. We factor this matrix and use it as a factored preconditioner for $K = \sum_e K_e$.

The splitting idea is simple, but there is no easy way to exploit it in most preconditioning paradigms. In Reitzinger's method, for example, one could build an algebraic multigrid solver for $L = \sum_{e \in E(t)} \alpha_e L_e$, but how would one incorporate the inapproximable elements into this solver? There is no obvious way to do this. An algebraic multigrid solver for $\gamma M + \sum_{e \notin E(t)} K_e$ is unlikely to be much better as a preconditioner than an algebraic multigrid solver for K, because $\gamma M + \sum_{e \notin E(t)} K_e$ is far from diagonal dominance unless all the elements are approximable.

The reason that the splitting idea works well with combinatorial preconditioners is that combinatorial preconditioning algorithms sparsify the SDD matrix L that is given to them as input. The sparsification makes the Cholesky factorization of the sparsified M much cheaper and sparser than the factorization of L. If most of the elements are approximable, adding $\sum_{e \notin E(t)} K_e$ to γM is likely to yield a preconditioner that is still cheap to factor.

Once the Cholesky factor of the preconditioners are computed, we use it in a preconditioned symmetric Krylov-subspace solver such as Conjugate Gradients [**68, 124**], SYMMLQ, or MINRES [**165**]. For most of the combinatorial algorithms that we can use to construct M, it is possible to show that the preconditioner is spectrally close to K. The spectral bounds give a bound on the number of iterations that the Krylov-subspace algorithm performs.

Experimental results that explore the performance and behavior of our solver show that the solver is highly reliable. In particular, on some problems other solvers, including an algebraic-multigrid solver and an incomplete-Cholesky solver, either fail or are very slow; our solver handles these problems without difficulty.

The rest of the chapter is organized as follows. Section 2.2 presents our element-approximation method. The scaling of the element-by-element approximation is presented in Section 2.3. The combinatorial sparsification phase is described in Section 2.4 and the handling of inapproximable elements in Section 2.5. The costs associated with the different phases of the solver are described in Section 2.6. Experimental results are presented in Section 2.7. We mention some open problems that this research raises in Section 2.8.

2.2. Nearly-Optimal Element-by-Element Approximations

In this section we show how to compute a nearly optimal SDD approximation L_e to a given symmetric positive semidefinite matrix K_e that is either nonsingular or has a null space consisting only of the constant vector.

2.2.1. Defining the Problem. Let $\mathbb{S}$ be a linear subspace of $\mathbb{R}^n$ (the results of this section also apply to $\mathbb{C}^n$, but we use $\mathbb{R}^n$ in order to keep the discussion concrete). We denote by $\mathbb{R}^{\mathbb{S}} \subseteq \mathbb{R}^{n \times n}$ the set of symmetric positive (semi)definite matrices whose null space is exactly $\mathbb{S}$.

Definition 2.2.1. Given two matrices A and B in $\mathbb{R}^{\mathbb{S}}$, a *finite generalized eigenvalue* λ of (A, B) is a scalar satisfying $Ax = \lambda Bx$ for some $x \notin \mathbb{S}$. The *generalized finite spectrum* $\Lambda(A, B)$ is the set of finite generalized eigenvalues of (A, B), and the *generalized condition*

number $\kappa(A, B)$ is

$$\kappa(A, B) = \frac{\max \Lambda(A, B)}{\min \Lambda(A, B)} .$$

(This definition can be generalized to the case of different null spaces, but this is irrelevant for this chapter.) We informally refer to $\kappa(A, B)$ as the spectral distance between A and B.

We refer to the following optimization problem as the *optimal symmetric semidefinite approximation problem.*

Problem 2.2.2. Let A be a symmetric positive (semi)definite matrix with null space $\mathbb{S}$ and let $B_1, B_2, \ldots, B_m$ be rank-1 symmetric positive semidefinite matrices. Find coefficients $d_1, d_2, \ldots, d_m$ that minimize the generalized condition number of A and

$$B_{\text{opt}} = \sum_{j=1}^{m} d_j^2 B_j$$

under the constraint $\text{null}(B_{\text{opt}}) = \text{null}(A)$, or decide that the null spaces cannot match under any d_j's. We can assume without loss of generality that all the B_j's have unit norm.

A slightly different representation of the problem is useful for characterizing the optimal solution. Let $B_j = Z_j Z_j^T$, where Z_j is a column vector. Let Z be an n-by-m matrix whose columns are the Z_j's. Then

$$\sum_{j=1}^{m} d_j^2 B_j = \sum_{j=1}^{m} d_j^2 Z_j Z_j^T = Z D D^T Z^T$$

where D is the m-by-m diagonal matrix with d_j as its jth diagonal element. This yields an equivalent formulation of the optimal symmetric semidefinite approximation problem.

Problem 2.2.3. Given a symmetric positive (semi)definite n-by-n matrix A with null space $\mathbb{S}$ and an n-by-m matrix Z, find an m-by-m diagonal matrix D such that $\text{null}(ZDD^TZ^T) = \mathbb{S}$ and that minimizes the generalized condition number $\kappa(A, ZDD^TZ^T)$, or report that no such D exists.

We are interested in cases where $\text{range}(Z) = \text{range}(A)$, where the problem is clearly feasible.

Vavasis has pointed out [**219**] that minimizing $\kappa(Z^TD^TDZ, A)$ is equivalent to finding a diagonal non-negative matrix S that minimizes $\max \Lambda(Z^TSZ, A)$ such that $\min \Lambda(Z^TSZ, A) \geq 1$, and taking $D = S^{1/2}$. He showed that this optimization problem can be solved as a convex semidefinite programming problem: find the minimum t and the corresponding S for which the constraints $\min \Lambda(tA, Z^TSZ) \geq 1$ and $\min \Lambda(Z^TSZ, A) \geq 1$ can be satisfied simultaneously. Convex semidefinite programming problems can be solved in polynomial time. An algorithm based on this observation can find an optimal SDD approximation, but applying it might be costly. We have decided to pursue a different direction by finding a suboptimal but provably good approximation using a fast and simple algorithm.

2.2.2. From Generalized Condition Numbers to Condition Numbers. The main tool that we use to find nearly optimal solutions to Problem (2.2.2) is a reduction of the problem to the well studied problem of scaling the columns of a matrix to minimize its condition number.

Definition 2.2.4. Given a matrix A, let $\sigma_{\max}$ be the largest singular value of A and $\sigma_{\min}$ be the smallest nonzero singular value of A. The *condition number* of A is $\kappa(A) = \sigma_{\max}/\sigma_{\min}$. If $A \in \mathbb{R}^{\mathbb{S}}$ then $\kappa(A) = \kappa(A, P_{\perp\mathbb{S}})$, where $P_{\perp\mathbb{S}}$ is the orthogonal projector onto the subspace orthogonal to $\mathbb{S}$.

The following lemma, which is a generalization of [**43**, Theorem 4.5], is the key to the characterization of B_{opt}.

Lemma 2.2.5. *Let $A = UU^T$ and $B = VV^T$, where U and V are real valued matrices of order $n \times m$. Assume that A and B are symmetric, positive semidefinite and null(A) = null(B). We have*

$$\Lambda(A, B) = \Sigma^2\left(V^+U\right)$$

and

$$\Lambda(A, B) = \Sigma^{-2}\left(U^+V\right) \ .$$

In these expressions, $\Sigma(\cdot)$ is the set of nonzero singular values of the matrix within the parentheses, Σ^ℓ denotes the same singular values to the ℓth power, and V^+ denotes the Moore-Penrose pseudoinverse of V.

Proof. Both U and V have n rows, so U^+ and V^+ have n columns. Therefore, the products V^+U and U^+V exist. Therefore,

$$\begin{aligned}
\Sigma^2\left(V^+U\right) &= \Lambda\left(V^+UU^T\left(V^+\right)^T\right) \\
&= \Lambda\left(\left(V^+\right)^T V^+UU^T\right) \\
&= \Lambda\left(\left(VV^T\right)^+ UU^T\right) \\
&= \Lambda\left(B^+A\right) \ .
\end{aligned}$$

($\Lambda(\cdot)$ denotes the set of eigenvalues of the argument.) For the second line we use the following observation. If X is n-by-k and Y is k-by-n, then the nonzero eigenvalues of XY and YX are the same. The second line follows from this observation for $X = \left(V^+\right)^T$ and $Y = V^+UU^T$. The third line follows from the equality $\left(XX^T\right)^+ = \left(X^+\right)^T X^+$ for an order n-by-k matrix X [**34**, Proposition 6.1.6].

It is sufficient to show that $\Lambda\left(B^+A\right) = \Lambda(A, B)$ in order to prove the first part of the lemma. Let $\lambda \in \Lambda(A, B)$, then there exists a vector $x \perp \text{null}(B) = \text{null}(A)$, such that $Ax = \lambda Bx$. Since B is symmetric, $x \in \text{range}(B) = \text{range}(B^T)$. Therefore, $B^+Ax = \lambda B^+Bx = \lambda x$. The last equality follows from the fact that B^+B is a projector onto range(B^T) [**34**, Proposition 6.1.6]. Therefore, $\Lambda\left(B^+A\right) \supseteq \Lambda(A, B)$. Let $\lambda \in \Lambda(B^+A)$, then there exists a vector x, such that $B^+Ax = \lambda x$. Therefore, $Ax = BB^+Ax = \lambda Bx$. The first equality follows from the fact that range(A) = range(B) and [**34**, Proposition 6.1.7]. Therefore, $\Lambda\left(B^+A\right) \subseteq \Lambda(A, B)$ which shows the equality.

The second result $\Lambda(A, B) = \Sigma^{-2}\left(U^+V\right)$ follows from replacing the roles of A and B in the analysis above and from the equality $\Lambda(A, B) = \Lambda^{-1}(B, A)$. The reversal yields

$$\Lambda(A, B) = \Lambda^{-1}(B, A) = \left(\Sigma^2\left(U^+V\right)\right)^{-1} = \Sigma^{-2}\left(U^+V\right) \ .$$

□

This lemma shows that Problems 2.2.2 and 2.2.3 can be reduced to the problem of scaling the columns of a single matrix to minimize its condition number. Let $A = UU^T$ and let Z satisfy range(Z) = range(A). (If A is symmetric positive semidefinite but U is not given, we can compute such a U from the Cholesky factorization of A or from its eigendecomposition.) According to the lemma,

$$\Lambda(A, ZDD^TZ^T) = \Sigma^{-2}\left(U^+ZD\right) .$$

Therefore, minimizing $\kappa(A, ZDD^TZ^T)$ is equivalent to minimizing the condition number $\kappa(U^+ZD)$ under the constraint range(ZD) = range(Z).

The other equality in Lemma 2.2.5 does not appear to be useful for such a reduction. According to the equality

$$\Lambda(A, ZDD^TZ^T) = \Sigma^{2}\left((ZD)^+U\right) ,$$

but unfortunately, there does not appear to be a way to simplify $(ZD)^+U$ in a way that makes D appear as a row or column scaling. (Note that in general, $(ZD)^+ \neq D^+Z^+$.)

The problem of scaling the columns of a matrix to minimize its condition number has been investigated extensively. Although efficient algorithms for minimizing the condition number of a rectangular matrix using diagonal scaling do not exist, there is a simple approximation algorithm. It may be possible to use Vavasis's algorithm [**219**] to solve this problem too, but this is not the concern of this work.

2.2.3. Computing the Pseudoinverse of a Factor of an Element Matrix. Before we can find a scaling matrix D, we need to compute U^+ from a given element matrix $K_e = UU^T$ and to form U^+Z.

We compute U^+ in one of two ways. If the input to our solver is an element matrix K_e with a known null space, we can compute U^+ from an eigendecomposition of K_e. Let $K_e = Q_e\Lambda_eQ_e^T$ be the *reduced* eigendecomposition of K_e (that is, Q_e is n-by-rank(K_e) and Λ_e is a rank(K_e)-by-rank(K_e) nonsingular diagonal matrix). We have $K_e = Q_e\Lambda_e^{1/2}\left(Q_e\Lambda_e^{1/2}\right)^T$ so we can set $U = Q_e\Lambda_e^{1/2}$, so $U^+ = \Lambda_e^{-1/2}Q_e^T$.

Many finite-element discretization techniques actually generate the element matrices in a factored form. If that is the case, then some symmetric factor F of $K_e = FF^T$ is given to our solver as input. In that case, we compute a reduced singular-value decomposition SVD of F, $F = Q_e\Sigma_eR_e^T$, where Σ_e is square, diagonal, and invertible, and R_e is square and unitary, both of order rank(F). Since

$$K_e = FF^T = Q_e\Sigma_eR_e^TR_e\Sigma_e^TQ_e^T = Q_e\Sigma_e^2Q_e^T$$

is an eigendecomposition of K_e, we can set $U = Q_e\Sigma_e$ and we have $K_e = UU^T$. In this case $U^+ = \Sigma_e^{-1}Q_e^T$. This method is more accurate when K_e is ill conditioned.

Note that in both cases we do not need to explicitly form U^+; both methods provide a factorization of U^+ that we can use to apply it to Z.

Once we form U^+Z, our solver searches for a diagonal matrix D that brings the condition number of U^+ZD close to the minimal condition number possible. This problem is better understood when U^+Z is full rank. Fortunately, in our case it always is.

Lemma 2.2.6. *Let U be a full rank m-by-n matrix, $m \geq n$, and let Z be an m-by-ℓ matrix with range(Z) = range(U). Then U^+Z has full row rank.*

PROOF. Since range(Z) = range(U), there exists an n-by-l matrix C such that $Z = UC$. By definition, rank(Z) $\leq$ rank(C) $\leq n$. Moreover, since range(Z) = range(U) and U is full rank, we have that n = rank(Z). Therefore, rank(C) = n.

It is sufficient to show that $C = U^+Z$ to conclude the proof of the lemma. Since U is full rank and $m \geq n$, the product U^+U is the n-by-n identity matrix. Therefore, $U^+Z = U^+UC = C$.

□

Without the assumption range(Z) = range(U), the matrix U^+Z can be rank deficient even if both $U^+$and Z are full rank.

Example 2.2.7. Let

$$U = \begin{bmatrix} 2 & 0 \\ -1 & -1 \\ -1 & 1 \end{bmatrix}, \quad Z = \begin{bmatrix} 1 & 1 \\ 1 & -1 \\ 1 & 0 \end{bmatrix}.$$

The columns of U are orthogonal, range(Z) $\neq$ range(U). This gives

$$U^+ = \begin{bmatrix} 1/3 & -1/6 & -1/6 \\ 0 & -1/2 & 1/2 \end{bmatrix}$$

and

$$U^+Z = \begin{bmatrix} 0 & 1/2 \\ 0 & 1/2 \end{bmatrix},$$

which is clearly not full rank.

2.2.4. Nearly-Optimal Column Scaling. Given a matrix U^+Z, we wish to find a diagonal matrix D that minimizes the condition number of U^+ZD, under the assumption range(Z) = range(U), and under the constraint that range(ZD) = range(Z).

To keep the notation simple and consistent with the literature, in this section we use A to denote U^+Z and we use m and n to denote the number of rows and columns in $A = U^+Z$.

The key result that we need is due to van der Sluis [**216**], who showed that choosing $\tilde{D}$ such that all the columns of $A\tilde{D}$ have unit 2-norm brings $A\tilde{D}$ to within a factor of $\sqrt{n}$ of the optimal scaling. Van der Sluis, extending an earlier result of Bauer for square invertible matrices [**28**], analyzed the full-rank case.

Given an m-by-n matrix A, $m \geq n$, and a nonsingular diagonal D van der Sluis defined

$$\kappa_{\text{vdS}}(AD) = \frac{\|AD\|_2}{\min_{x \neq 0} \|ADx\|_2 / \|x\|_2} \tag{2.2.1}$$

(his original definition is not specific to the 2-norm, but this is irrelevant for us). If A is non-singular then $\kappa_{\text{vdS}}(AD) = \kappa(AD)$. He, like Bauer, was interested in finding the diagonal matrix D that minimizes (2.2.1). This definition of the problem implicitly assumes that A is full rank, otherwise $\kappa_{\text{vdS}}(AD) = \infty$ for any nonsingular diagonal D. Also, if A is full rank then the minimizing D must give a full-rank AD. If A has more columns than rows, we can use a complementary result by van der Sluis, one that uses row scaling on a matrix with more rows than columns. Van der Sluis result show that if the rows of $\tilde{D}A$ have unit 2-norm, then $\kappa_{\text{vdS}}(\tilde{D}A)$ is within a factor of $\sqrt{m}$ of the minimum possible. This gives us the result that we need, because $\kappa_{\text{vdS}}(AD) = \kappa_{\text{vdS}}(D^TA^T)$. Shapiro showed that, in the general case, van der Sluis' estimate on $\kappa_{\text{vdS}}(A\tilde{D})$ cannot be

improved by more than a $\sqrt{2}$ factor [**188**]. The following are formal statements of the last two cited results.

Lemma 2.2.8. *(Part of Theorem 3.5 in* [**216**]*) Let A be an m-by-n full-rank matrix and let $\tilde{D}$ be a diagonal matrix such that in $\tilde{D}A$ all rows have equal 2-norm. Then for every diagonal matrix D we have*

$$\kappa_{\mathrm{vdS}}(\tilde{D}A) \leq \sqrt{m}\kappa_{\mathrm{vdS}}(DA)\,.$$

Lemma 2.2.9. *(Theorem in* [**188**]*) For every $\epsilon > 0$ there exists an n-by-n nonsingular, real valued matrix A such that if:*

(1) *$\tilde{D}$ is a diagonal matrix such that all the diagonal elements of $\tilde{D}A^TA\tilde{D}$ are equal to one, and*
(2) *D is a diagonal matrix such that $\kappa(AD)$ is minimized,*

then

$$\kappa(A\tilde{D}) > \left(\sqrt{n/2} - \epsilon\right)\kappa(AD)\,.$$

As we have shown in Lemma 2.2.6, that matrix $A = U^+Z$ whose columns we need to scale is full rank, so van der Sluis's results apply to it.

We note that further progress has been made in this area for square invertible A's, but it appears that this progress is not applicable to our application when $A = U^+Z$ is rectangular (which is usually the case). Braatz and Morari showed that for a square invertible A, the minimum of $\kappa(AD)$ over all positive diagonal matrices D can be found by solving a related optimization problem, which is convex [**49**]. Their paper states that this result also applies to the rectangular case [**49**, Remark 2.5]; what they mean by that comment is that the related optimization problem minimizes $\|AD\|_2\|D^{-1}A^+\|_2$ [**48**], whereas we need to minimize $\kappa(AD) = \|AD\|_2\|(AD)^+\|_2$.

2.2.5. Nearly-Optimal Symmetric diagonally dominant Approximations. We now turn our attention to the matrices that arise as element matrices in finite-element discretizations of (1.2.1). Such a matrix K_e has null space that is spanned by the constant vector and by unit vectors e_j for every zero column j of K_e. The part of the null space that is spanned by the unit vectors is irrelevant, so we assume that we are dealing with a matrix A whose null space is spanned by constant vector $\begin{bmatrix}1 & 1 & \cdots & 1\end{bmatrix}^T$.

We wish to approximate a symmetric semidefinite matrix A with this null space (or possibly a nonsingular matrix) by a symmetric diagonally dominant matrix B,

$$B_{ii} \geq \sum_{\substack{j=1\\ j\neq i}}^{n} |B_{ij}|\ .$$

To define a matrix Z such that the expression ZDD^TZ^T can generate any symmetric diagonally dominant matrix, we define the following vectors.

Definition 2.2.10. Let $1 \leq i, j \leq n$, $i \neq j$. A length-n *positive edge vector*, denoted $\langle i, -j\rangle$, is the vector

$$\langle i, -j\rangle_k = \begin{cases} +1 & k = i \\ -1 & k = j \\ 0 & \text{otherwise.} \end{cases}$$

A *negative edge vector* $\langle i, j \rangle$ is the vector

$$\langle i, j \rangle_k = \begin{cases} +1 & k = i \\ +1 & k = j \\ 0 & \text{otherwise.} \end{cases}$$

A *vertex vector* $\langle i \rangle$ is the unit vector

$$\langle i \rangle_k = \begin{cases} +1 & k = i \\ 0 & \text{otherwise.} \end{cases}$$

A symmetric diagonally dominant matrix can always be expressed as a sum of outer products of scaled edge and vertex vectors. Therefore, we can conservatively define Z to be the matrix whose columns are all the positive edge vectors, all the negative edge vectors, and all the vertex vectors.

If A is singular and its null space is the constant vector, we can do better. Chen and Toledo provided a combinatorial characterization of the null space of SDD matrices [**64**].

Lemma 2.2.11. *([**64**]) Let A be a symmetric diagonally dominant matrix whose null space is the constant vector. Then A is a sum of outer products of scaled positive edge vectors. Furthermore, the null space of a symmetric diagonally dominant matrix with a positive off-diagonal element (corresponding to an outer product of a scaled negative edge vector) cannot be span* $\begin{bmatrix} 1 & 1 & \cdots & 1 \end{bmatrix}^T$.

Therefore, if A is singular with this null space, we only need to include in the column set Z the set of positive edge vectors. If A is nonsingular, we also include in Z negative edge vectors and vertex vectors.

We can also create even sparser Z's; they will not allow us to express every SDD B as $B = ZDD^TZ^T$, but they will have the same null space as A. To define these sparser Z's, we need to view the edge vector $\langle i, -j \rangle$ as an edge that connects vertex i to vertex j in a graph whose vertices are the integers 1 through n. The null space of ZZ^T is the constant vector if an only if the columns of Z, viewed as edges of a graph, represent a connected graph. Therefore, we can build an approximation $B = ZDD^TZ^T$ by selecting an arbitrary connected graph on the vertices $\{1, \ldots, n\}$. By [**64**, Lemma 4.2], if A is nonsingular, we can include in Z the positive edge vectors of a connected graph plus one arbitrary vertex vector.

If A is well conditioned (apart perhaps from one zero eigenvalue), we can build a good approximation $B = ZDD^TZ^T$ even without the column-scaling technique of Lemma 2.2.5. In particular, this avoids the computation of the pseudo-inverse of a factor U of $A = UU^T$. Clearly, if A is nonsingular and well conditioned, then we can use I as an approximation: the generalized condition number $\kappa(A, I)$ is $\kappa(A)$. If A has rank $n - 1$ and the constant vector is its null vector, then

$$B_{C(n_e)} = \frac{1}{n_e} \begin{bmatrix} n_e - 1 & -1 & -1 & \cdots & -1 \\ -1 & n_e - 1 & -1 & \cdots & -1 \\ -1 & -1 & n_e - 1 & \cdots & -1 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ -1 & -1 & \cdots & \cdots & n_e - 1 \end{bmatrix}$$

yields

$$\kappa(A, B_{C(n_e)}) = \sigma_{\max}(A)/\sigma_{\min}(A) \ , \tag{2.2.2}$$

which we assumed is low ($\sigma_{\max}(A)$ is the largest singular value of A, and $\sigma_{\min}(A)$ is the smallest nonzero singular value of A). The matrix $B_{C(n_e)}$ is the Laplacian of the complete graph, and it is clearly SDD. The identity (2.2.2) follows from the fact that $B_{C(n_e)}$ is an orthogonal projection onto range(A). The following lemma summarizes this discussion.

Lemma 2.2.12. *Let A be a symmetric positive (semi)definite matrix. If A is nonsingular, or if the null space of A is span $\begin{bmatrix}1 & 1 & \cdots & 1\end{bmatrix}^T$, then there is an SDD matrix B such that $\kappa(A, B) \leq \kappa(A)$.*

This result may seem trivial (it is), but it is nonetheless important. The global stiffness matrix $K = \sum_e K_e$ is often ill conditioned, but the individual element matrices K_e are usually well conditioned. Since we build the approximation $L = \sum_e L_e$ element by element, this lemma is often applicable: When K_e is well conditioned, we can set L_e to be an extension of $B_{C(n_e)}$ into an n-by-n matrix. The rows and columns of the scaled $B_{C(n_e)}$ are mapped to rows and columns $\mathcal{N}_e$ of L_e and the rest of L_e is zero.

For well-conditioned A's, we can also trade the approximation quality for a sparser approximation than $B_{C(n_e)}$. The matrix

$$B_{S(n_e)} = \frac{1}{n_e} \begin{bmatrix} n_e - 1 & -1 & -1 & \cdots & -1 \\ -1 & 1 & 0 & \cdots & 0 \\ -1 & & 1 & \cdots & 0 \\ \vdots & & \vdots & \ddots & \vdots \\ -1 & & 0 & \cdots & 1 \end{bmatrix}$$

gives

$$\kappa(A, B_{S(n_e)}) \leq \kappa(A, B_{C(n_e)})\kappa(B_{C(n_e)}, B_{S(n_e)}) = \kappa(A)\kappa(B_{S(n_e)}) = \frac{n_e \sigma_{\max}(A)}{\sigma_{\min}(A)} \ . \tag{2.2.3}$$

For small n_e, this may be a reasonable tradeoff. The bound (2.2.3) follows from the observation that the eigenvalues of $B_{S(n_e)}$ are exactly 0, 1, and n_e. We note that besides generating a sparser matrix this kind of approximation preserves structural properties such as planarity. This may be important for certain sparsification algorithms **[32, 140]** and subsequent factorization algorithms.

When A is ill conditioned, there may or may not be an SDD matrix B that approximates it well. The following two examples demonstrate both cases.

Example 2.2.13. Let

$$A = \frac{1}{2\epsilon} \begin{bmatrix} 1 + \epsilon^2 & -\epsilon^2 & -1 \\ -\epsilon^2 & \epsilon^2 & 0 \\ -1 & 0 & 1 \end{bmatrix}$$

for some small $\epsilon > 0$. This matrix has rank 2 and null vector $\begin{bmatrix}1 & 1 & 1\end{bmatrix}^T$, and it its condition number is proportional to $1/\epsilon^2$. Since A is diagonally dominant, there is clearly an SDD matrix B (namely, A itself) such that $\kappa(A, B) = 1$. This matrix is the element matrix for a linear triangular element with nodes at $(0, 0)$, $(0, \epsilon)$, and $(1, 0)$ and a constant $\theta = 1$. The ill conditioning is caused by the high aspect ratio of the triangle, but this

ill conditioned matrix is still diagonally dominant. Small perturbations of the triangle will yield element matrices that are not diagonally dominant but are close to a diagonally dominant one.

Example 2.2.14. The matrix in Example 2.2.13 is a SDD matrix that is ill conditioned. This is an obvious example of an ill-conditioned but well-approximable matrix. We now show an example of a non-SDD (and not close to SDD) ill-conditioned matrix which is still well-approximable. Let

$$A = \frac{1}{6\epsilon}\begin{bmatrix} 3(1+\epsilon^2) & \epsilon^2 & 1 & -4\epsilon^2 & 0 & -4 \\ \epsilon^2 & 3\epsilon^2 & 0 & -4\epsilon^2 & 0 & 0 \\ 1 & 0 & 3 & 0 & 0 & -4 \\ -4\epsilon^2 & -4\epsilon^2 & 0 & 8(1+\epsilon^2) & -8 & 0 \\ 0 & 0 & 0 & -8 & 8(1+\epsilon^2) & -8\epsilon^2 \\ -4 & 0 & -4 & 0 & -8\epsilon^2 & 8(1+\epsilon^2) \end{bmatrix}$$

for some small $\epsilon > 0$. This matrix is the element matrix for a quadratic triangular element with nodes $(0,0)$, $(0,\epsilon)$ and $(1,0)$, quadrature points are midpoints of the edges with equal weights, and material constant $\theta = 1$.

This matrix is clearly ill conditioned since the maximum ratio between its diagonal elements is proportional to $1/\epsilon^2$.

To show that this matrix is approximable consider the following SDD matrix:

$$A_+ = \frac{1}{6\epsilon}\begin{bmatrix} 4(1+\epsilon^2) & 0 & 0 & -4\epsilon^2 & 0 & -4 \\ 0 & 4\epsilon^2 & 0 & -4\epsilon^2 & 0 & 0 \\ 0 & 0 & 4 & 0 & 0 & -4 \\ -4\epsilon^2 & -4\epsilon^2 & 0 & 8(1+\epsilon^2) & -8 & 0 \\ 0 & 0 & 0 & -8 & 8(1+\epsilon^2) & -8\epsilon^2 \\ -4 & 0 & -4 & 0 & -8\epsilon^2 & 8(1+\epsilon^2) \end{bmatrix}.$$

We will show that $\kappa(A, A_+) \leq 2$. Define the matrix

$$A_- = \frac{1}{6\epsilon}\begin{bmatrix} (1+\epsilon^2) & -\epsilon^2 & -1 & 0 & 0 & 0 \\ -\epsilon^2 & \epsilon^2 & 0 & 0 & 0 & 0 \\ -1 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}.$$

Notice that $A = A_+ - A_-$. Since A_- is symmetric positive definite this implies that $\lambda_{\max}(A, A_+) \leq 1$. We will show that $\lambda_{\min}(A, A_+) \geq 1/2$. This condition is equivalent to the condition that $\lambda_{\max}(A_+, A) \leq 2$, which is the equivalent to the condition that $2A - A_+$ is positive semidefinite. According to Lemma 3.3 in [**32**] it is enough to prove that $\bar{\sigma}(A_-, A_+) \leq \frac{1}{2}$. This can easily be achieved by path embedding where we embed the $(1,2)$ edge in A_- with the path$(1,4) \rightarrow (4,2)$ and the edge $(1,3)$ with the path $(1,6) \rightarrow (6,3)$. Congestion is 1 because no edge is reused and for both paths the dilation is $\frac{1}{2}$.

Example 2.2.15. Our example of a matrix that cannot be approximated well by an SDD matrix is the element matrix for an isosceles triangle with two tiny angles and one that

is almost π, with nodes at $(0,0)$, $(1,0)$, and $(1/2,\epsilon)$ for some small $\epsilon > 0$. The element matrix is

$$A = \frac{1}{2\epsilon}\begin{bmatrix} \frac{1}{4}+\epsilon^2 & \frac{1}{4}-\epsilon^2 & -\frac{1}{2} \\ \frac{1}{4}-\epsilon^2 & \frac{1}{4}+\epsilon^2 & -\frac{1}{2} \\ -\frac{1}{2} & -\frac{1}{2} & 1 \end{bmatrix}$$

This matrix has rank 2 and null vector $\begin{bmatrix}1 & 1 & 1\end{bmatrix}^T$. For any SDD matrix B with the same null space, $\kappa(A,B) \geq \epsilon^{-2}/4$ **[22, 193]** (this example is due to Shklarski).

2.2.6. A Heuristic for Symmetric diagonally dominant Approximations. In Section 2.2.5 we have shown how to find a nearly-optimal SDD approximation B to a symmetric positive (semi)definite matrix A whose null space is spanned by $\begin{bmatrix}1 & \cdots & 1\end{bmatrix}^T$. In this section we show a simple heuristic. We have found experimentally that it often works well. On the other hand, we can show that in some cases it produces approximations that are arbitrarily far from the optimal one. Thus, this section has two related goals: to describe a simple heuristic that often works well, but to point out that it cannot always replace the method of Section 2.2.5.

Definition 2.2.16. Let A be a symmetric positive (semi)definite matrix. We define A_+ to be the SDD matrix defined by

$$(A_+)_{ij} = \begin{cases} a_{ij} & i \neq j \text{ and } a_{ij} < 0 \\ 0 & i \neq j \text{ and } a_{ij} \geq 0 \\ \sum_{k \neq j} -(A_+)_{ik} & i = j\ . \end{cases}$$

Clearly, A_+ is SDD.

It turns out that in many cases, $\kappa(A, A_+)$ is small, making A_+ a good approximation for A. The following lemma gives a theoretical bound for $\kappa(A, A_+)$. The proof for this Lemma is due to Shklarski, and can be found in **[22, 193]**.

Lemma 2.2.17. *Let A be an SPSD n_e-by-n_e matrix with null(A) = span$[1 \ldots 1]^T$. Then null(A_+) = span$[1 \ldots 1]^T$, and $\kappa(A, A_+) \leq \sqrt{n_e}\kappa(A)$. Moreover, if there exist a constant c and an index i such that $\|A\|_{\text{inf}} \leq cA_{ii}$, then $\kappa(A, A_+) \leq c\kappa(A)$.*

This means that if A is well conditioned and small, then A_+ is always a fairly good approximation. The matrix A_+ is also sparser than A, and similarly to $B_{S(n)}$ its mesh will be planar if the mesh of A was planar.

But when A is ill conditioned, A_+ can be an arbitrarily bad approximation even though A is approximable by some other SDD matrix.

Example 2.2.18. Let $0 < \epsilon \ll 1$, and let $M \geq \frac{4}{\epsilon}$,

$$A = \begin{bmatrix} 1+M & -1 & 0 & -M \\ -1 & 1+M & -M & 0 \\ 0 & -M & M & 0 \\ -M & 0 & 0 & M \end{bmatrix} - \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 1-\epsilon & -1+\epsilon \\ 0 & 0 & -1+\epsilon & 1-\epsilon \end{bmatrix}.$$

This matrix is symmetric semidefinite with rank 3 and null vector $\begin{bmatrix}1 & 1 & 1 & 1\end{bmatrix}^T$. For small ϵ, A is ill conditioned, with condition number larger than $8\epsilon^{-2}$. Let

$$q_1 = \frac{1}{2}\begin{bmatrix}1\\1\\1\\1\end{bmatrix}\ , \quad q_2 = \frac{1}{2}\begin{bmatrix}1\\1\\-1\\-1\end{bmatrix}\ , \quad q_3 = \frac{1}{2}\begin{bmatrix}1\\-1\\1\\-1\end{bmatrix}\ , \text{ and } \quad q_4 = \frac{1}{2}\begin{bmatrix}1\\-1\\-1\\1\end{bmatrix}$$

be an orthonormal basis for $\mathbb{R}^4$. We have

$$\begin{aligned} q_1^T A q_1 &= 0 \\ q_2^T A q_2 &= 2M \\ q_3^T A q_3 &= 2M + \epsilon \\ q_4^T A q_4 &= \epsilon\ . \end{aligned}$$

Therefore, $\kappa(A) \geq 2M/\epsilon \geq 8\epsilon^{-2}$. We show that the matrix A_+ is a poor approximation of A.

$$\begin{aligned} q_1^T A_+ q_1 &= 0 \\ q_2^T A_+ q_2 &= 2M \\ q_3^T A_+ q_3 &= 2M + 1 \\ q_4^T A_+ q_4 &= 1\ . \end{aligned}$$

Therefore,

$$\kappa(A, A_+) > \left(1 - \frac{1-\epsilon}{2M+1}\right)\epsilon^{-1} \approx \epsilon^{-1}\ .$$

Nevertheless, the SDD matrix

$$B = \begin{bmatrix} \epsilon + M & -\epsilon & 0 & -M \\ -\epsilon & \epsilon + M & -M & 0 \\ 0 & -M & \epsilon + M & -\epsilon \\ -M & 0 & -\epsilon & \epsilon + M \end{bmatrix}$$

is a good approximation of A, with $\kappa(A, B) < 9$. This bound follows from a simple path-embedding arguments [**32**], which shows that $3A - B$ and $3B - A$ are positive semidefinite. The quantitative parts of these arguments rest on the inequalities

$$\frac{1}{2M} + \frac{1}{2M} + \frac{1}{3-\epsilon} \leq \frac{1}{3-4\epsilon}$$

and

$$\frac{1}{2M} + \frac{1}{2M} + \frac{1}{1+2\epsilon} < \frac{1}{1-3\epsilon}\ ,$$

which hold for small ϵ.

2.3. Scaling and Assembling Element Approximations

Given a set of approximations $\{L_e\}$ to a set of element matrices $\{K_e\}$, our solver scales the L_e's so that their assembly $L = \sum_e \alpha_e L_e$ is a good approximation of $K = \sum_e K_e$. We can scale them in one of two equivalent ways. The next lemma shows that under these scalings, if every L_e is a good approximation of K_e, then L is a good approximation of

K. Both scaling rules require the computation of one extreme eigenvalue for each pair (K_e, L_e).

Lemma 2.3.1. *Let $\{K_e\}_{e\in E}$ and $\{L_e\}_{e\in E}$ be sets of symmetric positive (semi)definite matrices with null(K_e) = null(L_e). Let $\alpha_e = \min \Lambda(K_e, L_e)$ and let $\beta_e = \max \Lambda(K_e, L_e)$. Then*

$$\begin{aligned} \kappa\left(\sum_{e\in E} K_e, \sum_{e\in E} \alpha_e L_e\right) &\leq \max \kappa\left(K_e, L_e\right) \\ \kappa\left(\sum_{e\in E} K_e, \sum_{e\in E} \beta_e L_e\right) &\leq \max \kappa\left(K_e, L_e\right) \end{aligned}$$

Proof. Scaling L_e by α_e transforms the smallest generalized eigenvalue of $\Lambda(K_e, \alpha_e L_e)$ to 1, since

$$\Lambda(K_e, \alpha_e L_e) = \frac{1}{\alpha_e}\Lambda(K_e, L_e) \ .$$

Scaling L_e clearly does not change the generalized condition number, so $\max \Lambda(K_e, \alpha_e L_e) = \kappa(K_e, L_e)$.

By the splitting lemma [**32**],

$$\begin{aligned} \min \Lambda\left(\sum_{e\in E} K_e, \sum_{e\in E} \alpha_e L_e\right) &\geq \min\left\{\min \Lambda(K_e, \alpha_e L_e)\right\}_{e\in E} \\ &= \min\left\{1\right\}_{e\in E} \\ &= 1 \ . \end{aligned}$$

Also by the splitting lemma,

$$\begin{aligned} \max \Lambda\left(\sum_{e\in E} K_e, \sum_{e\in E} \alpha_e L_e\right) &\leq \max\left\{\max \Lambda(K_e, \alpha_e L_e)\right\}_{e\in E} \\ &= \max\left\{\kappa\left(K_e, L_e\right)\right\}_{e\in E} \ . \end{aligned}$$

Combining these two inequalities gives the result. The proof for scaling by β_e is the same. □

If we use inexact estimates $\tilde{\alpha_e}$ for the minimum of $\Lambda(K_e, L_e)$, the bound becomes

$$\kappa\left(\sum_{e\in E} K_e, \sum_{e\in E} \tilde{\alpha_e} L_e\right) \leq \frac{\max_e\left\{(\alpha_e/\tilde{\alpha_e})\,\kappa\left(K_e, L_e\right)\right\}}{\min_e\left(\alpha_e/\tilde{\alpha_e}\right)} \ ,$$

and similarly for estimates of β_e. This shows that $\kappa\left(\sum_{e\in E} K_e, \sum_{e\in E} \tilde{\alpha_e} L_e\right)$ depends on how much the estimates vary. In particular, if the relative estimation errors are close to 1, scaling by the estimates is almost as good as scaling by the exact eigenvalues.

2.4. Sparsification of the Assembled SDD Approximation

Once we obtain an SDD approximation $L = \sum_e \alpha_e L_e$ of K, we can use a combinatorial graph algorithm to construct an easier-to-factor SDD approximation M of L. Because M is spectrally close to L and L is spectrally close to K, M is also spectrally close to K.

By applying [**43**, Proposition 3.6] to both ends of the generalized spectra, we obtain the following lemma.

Lemma 2.4.1. *Let K, L, and M be symmetric (semi)definite matrices with the same dimensions and the same null space. Then*

$$\kappa(K, M) \leq \kappa(K, L)\kappa(L, M) \ .$$

There are several algorithms that can build M from L. All of them view an exactly (but not strictly) SDD matrix L as a weighted undirected graph G_L, to which they build an approximation graph G_M. The approximation G_M is then interpreted as an SDD matrix M. If L is strictly diagonal dominant, the approximation starts from an exactly SDD matrix $L - D$, where D is diagonal with nonnegative elements. From $L - D$ the algorithm builds an approximation $\tilde{M}$; If $\tilde{M}$ is a good approximation to $L - D$, then $\tilde{M} + D$ is a good approximation to L.

Lemma 2.4.2. *Let A and B be symmetric positive (semi)definite matrices with the same null space $\mathbb{S}$, and let C be a positive symmetric (semi)definite with a null space that is a subspace, possibly empty, of $\mathbb{S}$. Then*

$$\Lambda(A + C, B + C) \subseteq [\min \Lambda(A, B) \cup \{1\}, \max \Lambda(A, B) \cup \{1\}] \ .$$

Proof. The result is a simple application of the splitting lemma [**32**], since

$$\begin{aligned} \max \Lambda(A + C, B + C) &\leq \max \{\max \Lambda(A, B), \max \Lambda(C, C)\} \\ &= \max \{\max \Lambda(A, B), 1\} \\ &= \max \Lambda(A, B) \cup \{1\} \ , \end{aligned}$$

and similarly for the smallest generalized eigenvalue. □

This lemma is helpful, since most of the algorithms that construct approximations of SDD matrices provide theoretical bounds on $\Lambda(L - D, \tilde{M})$ that have 1 as either an upper bound or a lower bound. When this is the case, adding D to $L - D$ and to $\tilde{M}$ preserves the theoretical condition-number bound.

The earliest algorithm for this subproblem is Vaidya's algorithm [**32, 63, 215**]. This algorithm finds a maximum spanning tree in G_M and augments it with suitable edges. The quantity of extra edges that are added to the tree is a parameter in this algorithm. When few or no edges are added, $\kappa(L, M)$ is high (but bounded by $nm/2$, where m is the number of off-diagonal elements in L), but L is cheap to factor (can be factored in $O(n)$ operations when no edges are added to the tree). When many edges are added, $\kappa(L, M)$ shrinks but the cost of factoring L rises. When G_M is planar or nearly planar then the algorithm is very effective, both theoretically and experimentally. For more general classes of graphs, even with a bounded degree, the algorithm is less effective.

A heuristic for adding edges to the maximum spanning tree was proposed by Frangioni and Gentile [**100**]. They designed their algorithm for SDD linear systems that arise in the interior-point solution of network-flow problems. There are no theoretical convergence bounds for this algorithm.

Several algorithms are based on so-called *low-stretch spanning trees*. Boman and Hendrickson realized that a low-stretch spanning tree G_M of G_L leads to a better convergence-rate bound than maximum spanning trees [**42**]. Elkin et al. showed simpler and lower-stretch constructions for low-stretch trees [**93**]. Spielman and Teng proposed algorithms

that create denser graphs G_M (which are still based on low-stretch trees) [**200, 201, 202**]. The algorithms of Spielman and Teng lead to nearly-linear work bound for solving $Lx = b$.

There are other classes of combinatorial preconditioners, for example [**113, 112, 152**]. It is not clear whether they can be used effectively in our framework.

2.5. Dealing with Inapproximable Elements

When some of the element matrices cannot be approximated well by an SDD matrix, we split the global stiffness matrix K into $K = K_{\leq t} + K_{>t}$, where $K_{\leq t} = \sum_{e \in E(t)} K_e$ is a sum of the element matrices for which we found an SDD approximation L_e that satisfies $\kappa(K_e, L_e) \leq t$ for some threshold $t > 0$, and $K_{>t} = \sum_{e \notin E(t)} K_e$ is a sum of element matrices for which our approximation L_e gives $\kappa(K_e, L_e) > t$.

We then scale the approximations in $E(t)$ and assemble them to form $L_{\leq t} = \sum_{e \in E(t)} \alpha_e K_e$. We now apply one of the combinatorial graph algorithms discussed in Section 2.4 to construct an approximation $M_{\leq t}$ to $L_{\leq t}$. Once we have $M_{\leq t}$, we add it to $K_{>t}$ to obtain a preconditioner $M_1 = M_{\leq t} + K_{>t}$.

This construction gives a bound on $\kappa(K, M_1)$, but it is a heuristic in the sense that M_1 may be expensive to factor. The analysis of $\kappa(K, M_1)$ is essentially the same as the analysis of strictly dominant matrices in Section 2.4: by Lemma 2.4.2, a theoretical bound $\Lambda(K_{\leq t}, M_{\leq t}) \subseteq [\alpha, \beta]$ implies $\Lambda(K, M_1) \subseteq [\min\{\alpha, 1\}, \max\{\beta, 1\}]$.

The scaling technique of Lemma 2.3.1 ensures that either $\alpha, \beta \leq 1$ or $1 \leq \alpha, \beta$. But the interval $[\alpha, \beta]$ may be quite far from 1. If the interval is far from 1, the bound on $\kappa(K, M_1)$ can be considerably larger then the bound on $\kappa(K_{\leq t}, M_{\leq t})$. We observed this behavior in practice (experiments not reported here). To avoid this danger, we scale $M_{\leq t}$ before adding it to $K_{>t}$; that is, we use a preconditioner $M_\gamma = \gamma M_{\leq t} + K_{>t}$. We choose γ as follows. We find some vector v that is orthogonal to $\text{null}(M_{\leq t})$ and compute its generalized Raleigh quotient

$$\gamma = \frac{v^T K_{\leq t} v}{v^T M_{\leq t} v} .$$

The null space of $M_{\leq t}$ is determined by the connected components of its graph, so it is easy to quickly find such a v (we use a random v in this subspace). This definition ensures that $\gamma \in [\alpha, \beta]$. Since $\Lambda(K_{\leq t}, \gamma M_{\leq t}) \subseteq [\alpha/\gamma, \beta/\gamma]$, we have $1 \in [\alpha/\gamma, \beta/\gamma]$.

Lemma 2.5.1. *Under this definition of M_γ, $\kappa(K, M_\gamma) \leq \beta/\alpha$, where the interval $[\alpha, \beta]$ bounds $\Lambda(K_{\leq t}, M_{\leq t})$. In particular, if we take α and β to be the extremal generalized eigenvalues of $(K_{\leq t}, M_{\leq t})$, we obtain*

$$\kappa(K, M_\gamma) \leq \kappa(K_{\leq t}, M_{\leq t}) .$$

We expect that this overall heuristic will be effective when $E \setminus E(t)$ contains only few elements. If only a few elements cannot be approximated, then $K_{>t}$ is very sparse, so the sparsity pattern of $M_\gamma = \gamma M_{\leq t} + K_{>t}$ is similar to that of $M_{\leq t}$. Since $M_{\leq t}$ was constructed so as to ensure that its sparsity pattern allow it to be factored without much fill, we can expect the same to hold for M_γ. If $E \setminus E(t)$ contains many elements, there is little reason to hope that the triangular factor of M_γ will be particularly sparse.

If $E \setminus E(t)$ contains very few elements a different strategy can be used. Instead of introducing the ill-conditioned elements into M_γ they can be ignored. Each ignored element can be considered as an $n_e - 1$ rank perturbation. The results we discuss in

chapter 3 (and also appear in [**20**]) suggest that this will increase the number of iterations by at most $n_e - 1$. Therefore, if the number of inapproximable elements is small only a few more iterations will be needed. If many inapproximable elements are dropped convergence can be slow.

2.6. Asymptotic Complexity Issues

In this section we explain the asymptotic complexity of the different parts of our solver. We do not give a single asymptotic expression that bounds the work that the solver performs, but comment on the cost and asymptotic complexity of each phase of the solver. The cost of some of the phases is hard to fully analyze, especially when $E(t) \subsetneq E$. The next section presents experimental results that complement the discussion here.

The first action of the solver is to approximate each element matrix K_e by an SDD matrix $\alpha_e L_e$. For a given element type, this phase scales linearly with the number of elements and it parallelizes perfectly. The per-element cost of this phase depends on the approximation method and on the number n_e of degrees of freedom per element. Asymptotically, all the approximation methods require n_e^3 operations per element, but the uniform-clique is the fastest method. This phase also gives us $\kappa(K_e, \alpha_e L_e)$ which we use to decide which elements belong to $E(t)$ and which do not.

The next phase of the solver assembles the scaled SDD approximations in $E(t)$. The cost of this step is bounded by the cost to assemble $K = \sum_e K_e$, which most finite-elements solvers (including ours), perform. The assemblies of K and L performs $O(\sum_e n_e^2)$ operations: fewer than the first phase, but harder to parallelize.

The cost and the asymptotic complexity of the sparsification of L depends on the algorithm that is used. For Vaidya's sparsification algorithm, which our code uses, the amount of work is $O(n \log n + \sum_e n_e^2)$. For the algorithm of Spielman and Teng [**200, 201, 202**], the work is $O(m \log^{O(1)} m)$ where $m = \sum_e n_e^2$.

Next, the algorithm assembles the element matrices K_e that are not in $E(t)$ into $M_{\leq t}$. The cost of this phase is also dominated by the cost of assembling K.

The cost of computing the Cholesky factorization of M is hard to characterize theoretically, because the cost depends on the nonzero pattern of M in a complex way. The nonzero pattern of M depends on how many and which elements are not in $E(t)$, and on how much we sparsified $L_{\leq t}$. The number of operations in a phase of the solver is not the only determinant of running time, but also the computational rate. The Cholesky factorization of M usually achieves high computational rates.

The cost of the iterative solver depends on the number of iterations and on the per-iteration cost. The number of iterations is proportional to $\sqrt{\kappa(K, M_\gamma)} \leq t\sqrt{\kappa(L, M_{\leq t})}$. The amount of work per iteration is proportional to the number of nonzeros in K plus the number of nonzeros in the Cholesky factor of M. The sparsification algorithms of Vaidya and Spielman and Teng control the number of iterations, and if $E(t) = E$ than they also control the density of the Cholesky factor.

2.7. Experimental Results

This section presents experimental results that explore the effectiveness of our solver.

2.7.1. Setup. Our solver currently runs under MATLAB [**155**], but it is implemented almost entirely in C. The C code is called from MATLAB using MATLAB's CMEX interface. The element-by-element approximations are computed by C code that calls LAPACK [**12**]. The assembly of the element-by-element approximations (and possibly the inapproximable elements) is also done in C. The construction of Vaidya's preconditioners for SDD matrices is done by C code [**63**]. The Cholesky factorization of the preconditioner is computed by MATLAB's sparse `chol` function, which in MATLAB 7.2 calls CHOLMOD 1.0 by Tim Davis. We always order matrices using METIS version 4.0 [**138**] prior to factoring them. The iterative Krylov-space solver that we use is a preconditioned Conjugate Gradients written in C and based on MATLAB's PCG.; within this iterative solver, both matrix-vector multiplications and solution of triangular linear systems are performed by C code.

In most experiments we compare our solver to an algebraic multigrid solver, BoomerAMG [**123**]. We use the version of BoomerAMG that is packaged as part of HYPRE 1.2. We compiled it using GCC version 3.3.5, with options `-O` (this option is HYPRE's default compilation option). We note that BoomerAMG is purely algebraic and does not exploit the element-by-element information. There exist variations of algebraic multigrid that do exploit the element structure [**51, 62**]. We have not experimented with these variants. Our comparison with BoomerAMG is meant mainly to establish a credible baseline for the results and not to explore the behavior of algebraic multigrid solvers.

In some experiments we compare our solver to solvers that are based on incomplete Cholesky preconditioners [**154, 159, 177, 218**]. To compute these preconditioners, we use MATLAB's built-in CHOLINC routine. Here too, the matrices are preordered using METIS.

Since many of our test problems are ill conditioned, we iterate until the relative residual is at most 10^{-14}, close to $\epsilon_{\text{machine}}$, in order to achieve acceptable accuracy.

We use two mesh generators to partition the three-dimensional problem domains into finite elements. We usually use TETGEN version 1.4. [**195**]. In a few experiments we use DISTMESH [**170**], which can generate both two- and three-dimensional meshes.

Running times were measured on a 1.6 GHz AMD Opteron 242 computer with 8 GB of main memory, running Linux 2.6. This computer has 2 processors, but our solver only uses one. We used a 64-bit version of MATLAB 7.2. This version of MATLAB uses the vendor's BLAS, a library called ACML. The measured running times are wall-clock times that were measured using the FTIME Linux system call.

2.7.2. Test Problems. We evaluated our solver on several three-dimensional problems. We used both linear and quadratic tetrahedral elements. Table 1 summarizes the problems that we used in the experiments. The boundary conditions are always pure Neumann $\partial u/\partial n = 0$, and we removed the singularity by fixing the solution at a fixed unknown (algebraically, we remove the row and column of K corresponding to that unknown). We generate the right-hand side b of the linear system $Kx = b$ by generating a random solution vector x and multiplying it by K to form b.

In all the experiments reported below, except for a single experiment, our solver produced acceptable forward errors. The computed solution $\hat{x}$ satisfied

$$\frac{\|\hat{x} - x\|_2}{\|x\|_2} \leq 10^{-4} .$$

TABLE 1. Notation for the test problems.

The Domain Ω	
C	A 3-dimensional cube
B	A 3-dimensional box with aspect ratio 1-by-1-by-10000
CH	A 1-by-1-by-1 cube with a 1-by-0.1-by-0.79 hole in the middle
SC	A 10-by-10-by-10 cube containing a spherical shell of inner radius 3 and thickness 0.1.
The Mesh (the parameter indicates the number n of mesh points)	
G	3-dimensional, generated by TETGEN
D	3-dimensional, generated by DISTMESH
The Conductivity $\theta(x)$	
U	uniform and isotropic, $\theta = I$ everywhere
J	jump between subdomains but uniform and isotropic within subdomain (e.g., $\theta = 10^4 I$ in the spherical shell of domain SC and $\Theta = I$ elsewhere); the parameter indicates the magnitude of the jump
A	anisotropic within a subdomain (e.g. the spherical shell in SC) and $\theta = I$ elsewhere; θ is always 1 in the x and y directions and the parameter indicates the conductivity in the z direction.
The element type	
L	Linear tetrahedral element, 4-by-4 element matrix
Q	Quadratic tetrahedral element, 10-by-10 element matrix

In one of the experiments with well-conditioned elements and jumping coefficients with ratio 10^8 (Section 2.7.5), when running Vaidya's preconditioner with the goal 0.6, the forward error was $1.24 \cdot 10^{-3}$.

2.7.3. Choosing the Parameter t**.** We begin with simple problems that are designed to help us choose t, the approximation threshold. The behavior of our solver depends on two parameters, t and the aggressiveness of the combinatorial sparsification algorithm. These parameters interact in complex ways, because both influence the sparsity of the Cholesky factor of M and the number of iterations in the Krylov-space solver. It is hard to visualize and understand the performance of a solver in a two-dimensional (or higher) parameter space. Therefore, we begin with experiments whose goal is to establish a reasonable value for t, a value that we use in all of the subsequent experiments.

Figure 2.7.1 shows the results of these experiment, which were carried out on two meshes, one generated by DISTMESH and the other by TETGEN. The elements are all linear tetrahedral, and their approximations L_e are built using our nearly-optimal approximation algorithm. The graphs on the left show the distributions of $\kappa(K_e)$. With DISTMESH, we see that the elements belong to two main groups, a large group of elements with generalized condition numbers smaller than about 100, and a small set of so-called slivers with much higher condition numbers, ranging from 200 to 10^8. From results not shown here, it appears that for the non-slivers, $\kappa(K_e, L_e)$ is smaller than $\kappa(K_e)$ by roughly a constant factor. For the slivers, $\kappa(K_e, L_e)$ is close to $\kappa(K_e)$. With TETGEN, there are no highly ill-conditioned elements, and the distributions of $\kappa(K_e)$ and $\kappa(K_e, L_e)$ are smoother.

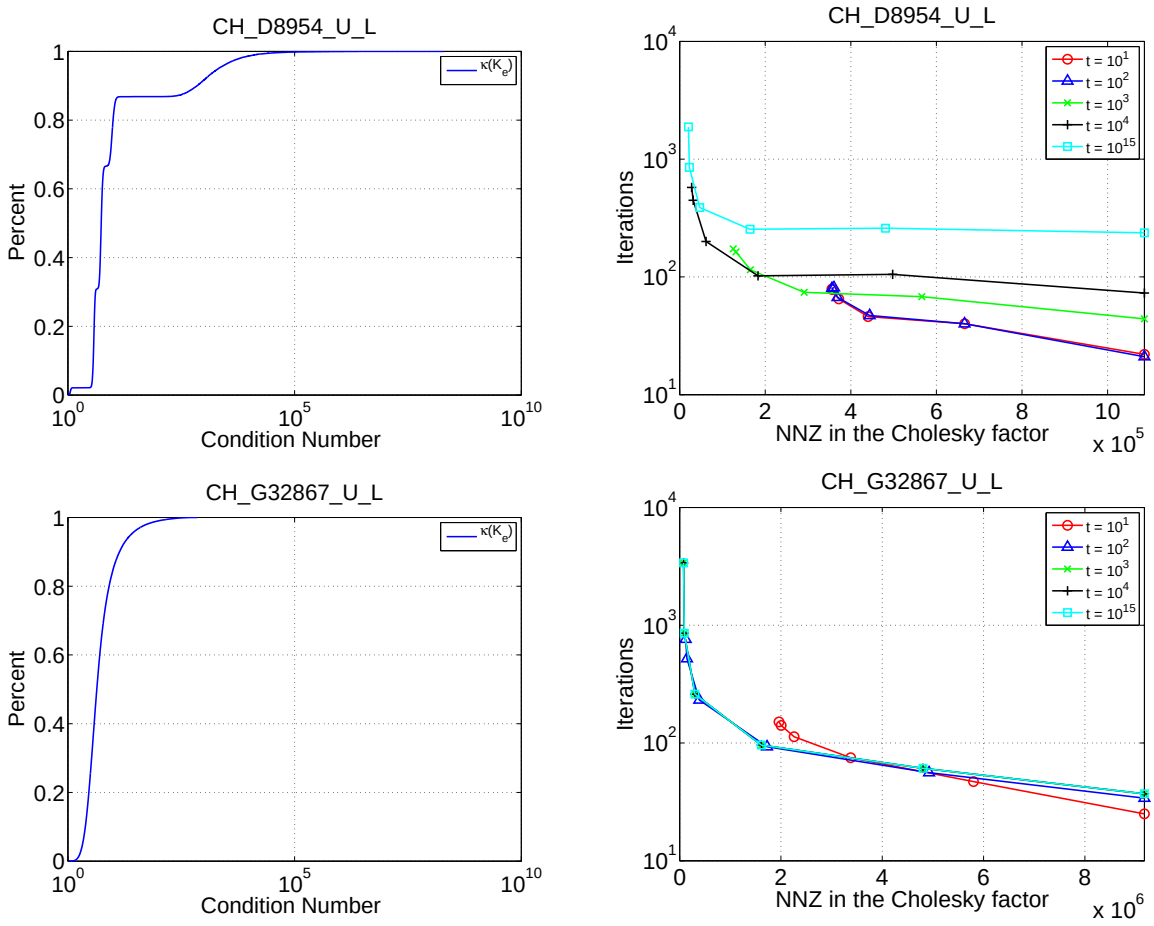


FIGURE 2.7.1. The distribution of element condition numbers (left graphs) and the number of iterations for several values of t and several sparsification levels (right graphs). The top row shows results on a mesh generated by DISTMESH, and the bottom row on a mesh generated by TETGEN.

The graphs on the right show the number of iterations that the Conjugate Gradients algorithm performs for several values of t and various levels of sparsification. In all the graphs in the chapter whose horizontal axis is fill in the Cholesky factor, the horizontal axis ranges from 0 to the number of nonzeros in the Cholesky factor of K. When t is small, $K_{>t}$ is relatively dense, so the sparsification algorithm cannot be very effective. Even when we instruct Vaidya's algorithm to sparsify $L_{\leq t}$ as much as possible, the Cholesky factor of M remains fairly similar to the Cholesky factor of K. On the other hand, a small t leads to faster convergence. With a large t we can construct M's with very sparse factors, but convergence is very slow. If all the elements are relatively well-conditioned then there is little dependence on t, as can be seen in the bottom right figure. A value of t near 1000 gives a good balance between high iteration counts caused by using L_e's with fairly high $\kappa(K_e, L_e)$ and the inability to construct a sparse preconditioner caused by a dense $K_{>t}$. We use the fixed value $t = 1000$ in the remaining experiments in order to clarify the role of the other parameter in the algorithm.

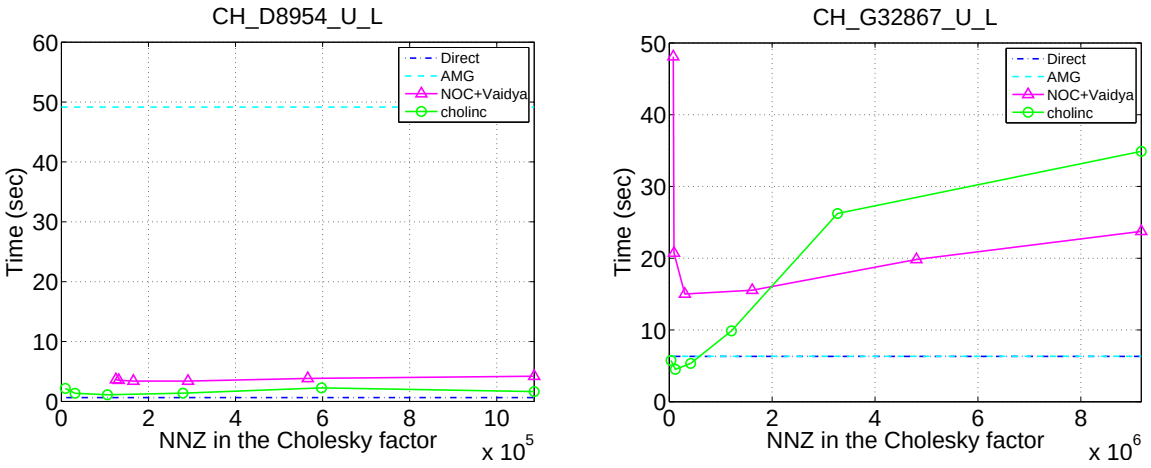


FIGURE 2.7.2. Running times for our solver, for incomplete Cholesky, for BoomerAMG, and for a direct solver on simple 3-dimensional problems. The graph on the left uses a mesh generated by DISTMESH, and the one on the right a mesh generated by TETGEN. See the first paragraph of Section 2.7.4 for a complete explanation of the graphs.

We stress that the selection of t in practice should not be static; it should be based on the actual distribution of the generalized condition numbers of the approximation and on analysis similar to the one described in this section.

2.7.4. Baseline Tests. The next set of experiments shows the performance of our solver relative to other solvers on the same problems and the same meshes, for a few relatively easy problems. The graphs in Figure 2.7.2 compare the running time of our solver to that of an incomplete Cholesky preconditioner, BoomerAMG, and a state-of-the-art direct solver, CHOLMOD. In these graphs the vertical axis represents wall-clock time for all the phases of the solution and the horizontal axis represents the number of nonzeros in the triangular factors of the preconditioner. The rightmost (largest) horizontal coordinate in the graphs always corresponds to the number of nonzeros in a complete sparse Cholesky factor of the coefficient matrix. When the complete factorization runs out of space, we still use this scaling of the horizontal axis, and we estimate the running time of the complete factorization based on the assumptions that it runs at 10^9 floating-point operations per second. The direct solver and BoomerAMG only give us one data point for each problem; their running times are represented in the graphs by horizontal lines. We ran each preconditioned solver with several values of the parameter that controls the sparsity of the factor (drop tolerance in incomplete Cholesky and the sparsification parameter in Vaidya's preconditioner). Therefore, for each preconditioned solver we have several data points that are represented by markers connected by lines. Missing markers and broken lines indicate failures to converge within a reasonable amount of time. Our algorithm is labeled as "NOC+Vaidya". NOC stands for "Nearly-Optimal Clique" because our algorithm uses a clique topology for the approximation. Most of the remaining graphs in this section share the same design.

The graphs in Figure 2.7.2 compare the running time of the solvers on easy problems with a relatively simple domain and uniform coefficients. The mesh produced by TETGEN

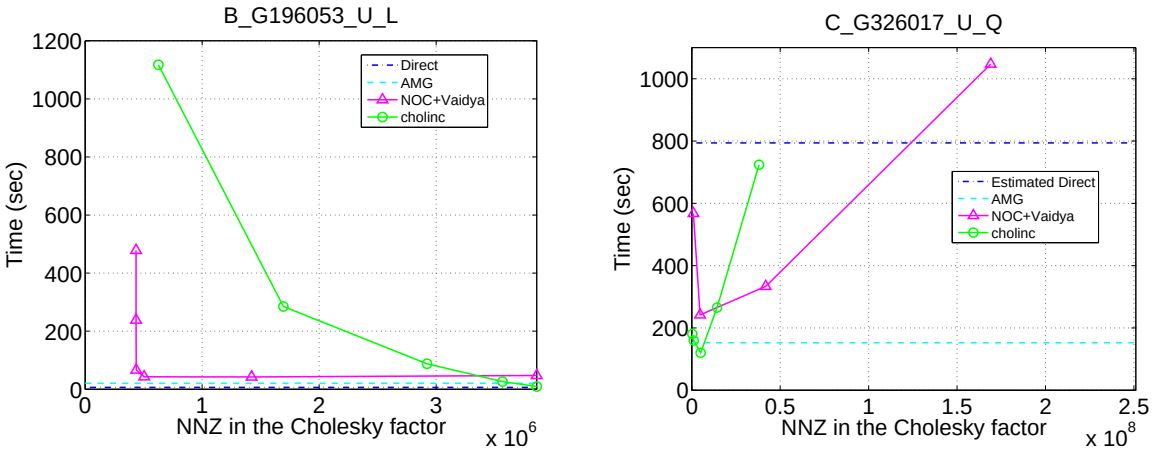


FIGURE 2.7.3. Experimental results on two additional problems with uniform coefficients; these problems are much larger than those analyzed in Figure 2.7.2.

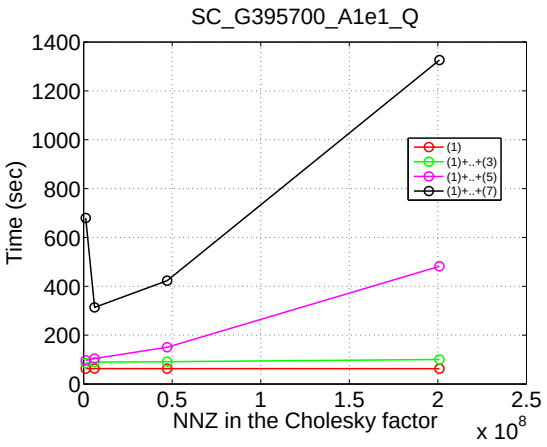
leads to a linear system that is easy for all the iterative solvers. With a good drop-tolerance parameter, incomplete Cholesky is the fastest, with little fill. Our solver is slower than all the rest, even with the best sparsity parameter. The mesh produced by DISTMESH causes problems to BoomerAMG, but incomplete Cholesky is faster than our solver.

Although the performance of incomplete Cholesky appears to be good in the experiments reported in Figure 2.7.2, it sometimes performs poorly even on fairly simple problems. Figure 2.7.3 shows that on a high-aspect-ratio 3-dimensional structure with uniform coefficients, incomplete Cholesky performs poorly: the sparser the preconditioner, the slower the solver. Our solver, on the other hand, performs reasonably well even when its factor is much sparser than the complete factor. On the high-aspect-ratio problem, as well as on any problem of small to moderate size, the direct solver performs well. But as the problem size grows the direct solver becomes slow and tends to run out of memory. The rightmost graph in Figure 2.7.3 shows a typical example.

Figure 8.5.7 shows a breakdown of the running time of our solver for one particular problem. The data shows that as the preconditioner gets sparse, the time to factor the preconditioner decreases. The running time of the iterative part of the solver also initially decreases, because the preconditioner gets sparser. This more than offsets the growth in the number of iterations. But when the preconditioner becomes very sparse, it becomes less effective, and the number of iterations rises quickly.

2.7.5. Well-Conditioned Elements and Jumping Coefficients. The next set of experiments explores problems with a large jump in the conductivity θ. We instructed the mesh generators to align the jump with element boundaries, so within each element, there is no jump. This leads to a large $\kappa(K)$, but the conditioning of individual element matrices is determined by their geometry, not by θ. The results, shown in Figure 2.7.5, show that the jump in θ does not influence any of the four solvers in a significant way.

2.7.6. Ill-Conditioned Elements: Anisotropy. Some of the experiments shown in Section 2.7.3 included ill-conditioned elements. The ill-conditioning of those elements



Notation for phases of the solver:
(1) Approximate K_e by L_e
(2) Assembly $L_{\leq t} = \sum_{E(t)} L_e$
(3) Sparsify $L_{\leq t}$ to obtain $M_{\leq t}$
(4) Assembly of $M = M_{\leq t} + K_{>t}$
(5) Order, permute, and factor M
(6) Assembly of $K = \sum_E K_e$
(7) Permute K and iterate

FIGURE 2.7.4. A breakdown of the running time of our solver, on a particular problem. The graph shows the time consumed by the different phases of the solver. Assembly phases are not separately shown because their running time is negligible.

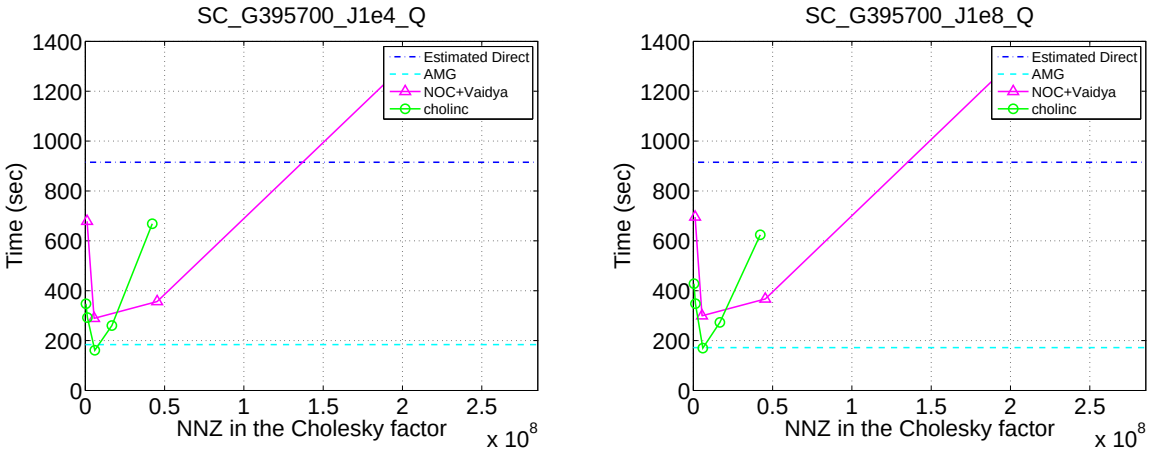


FIGURE 2.7.5. Running times for problems with jumping coefficients. The average of the ratio $\kappa(K_e)/\kappa(L_e, K_e)$ in both experiments is around 3.6 and is nearly constant. The condition numbers of the K_e's are always small.

resulted from their geometrical shape. Other mesh generators may be able to avoid such element shapes. Indeed, TETGEN did not produce such elements, only DISTMESH did. But in problems that contain anisotropic materials in complex geometries, ill-conditioned elements are hard to avoid.

Figure 2.7.6 compares the performance of our solver with that of other solvers on a problem in which the conductivity θ is anisotropic in one part of the domain. The results clearly show that anisotropy leads to ill-conditioned element matrices. As the anisotropy increases, BoomerAMG becomes slower and incomplete Cholesky becomes less reliable. The anisotropy does not have a significant influence on our solver. In experiments not reported here, our solver behaved well even with anisotropy of 10^8. The incomplete-factorization solver becomes not only slow, but also erratic, as the graphs in Figure 2.7.7

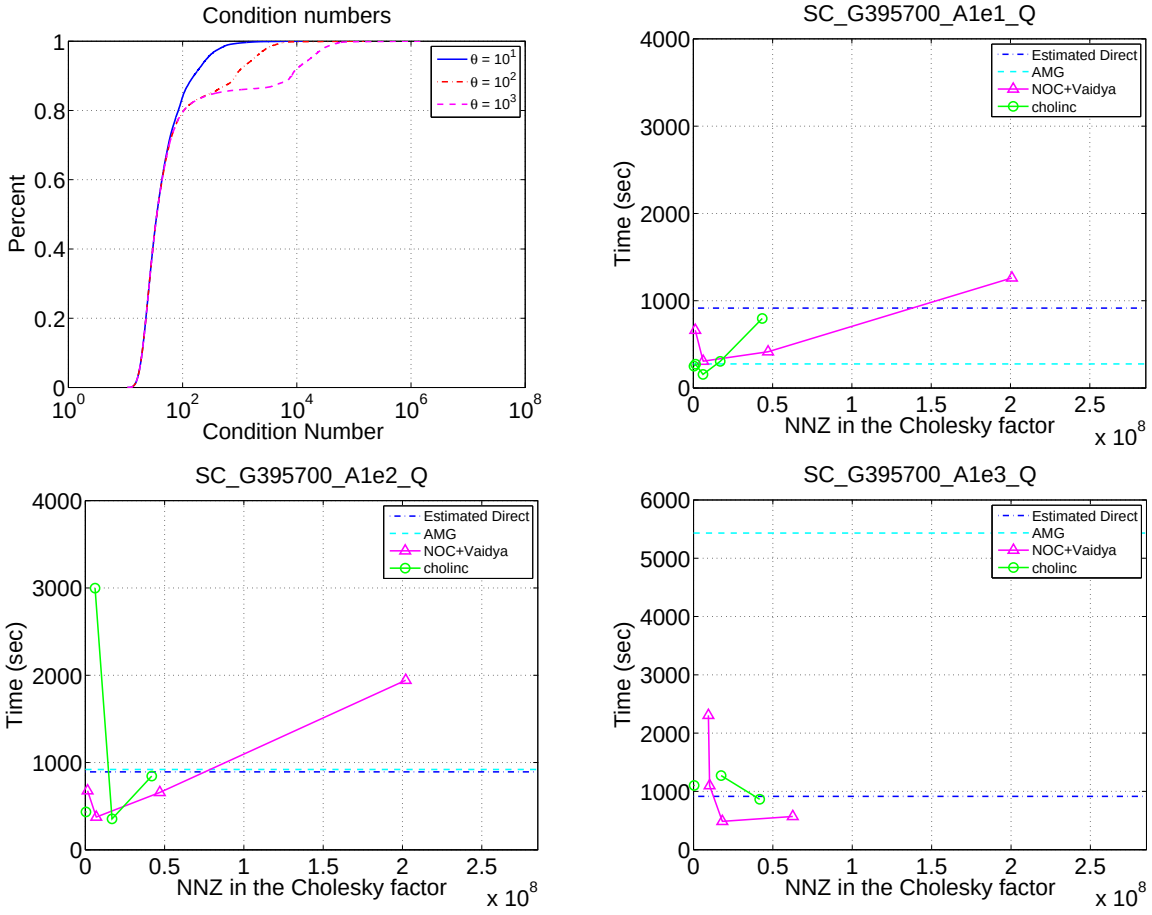


FIGURE 2.7.6. The behavior of our solver and other solvers on a solid 3-dimensional problem that contains a thin spherical shell with anisotropic material. The mesh was generated by TETGEN. We report here three experiments with anisotropy rates 10, 10^2, and 10^3 (θ in the legend). The top left graph shows the distribution of the element matrix condition numbers in the different experiments. The ratio $\kappa(K_e)/\kappa(L_e, K_e)$ is nearly constant within every experiment. This ratio for the isotropic elements is similar in all the experiments. For the anisotropic elements, the maximal ratio grows from about 19 for anisotropy 10 to about 806 for anisotropy 10^3.

show. The convergence of our preconditioner is always steady, monotonically decreasing, and the convergence rate is monotonic in the density of the preconditioner. The convergence of incomplete Cholesky is erratic, not always monotonic, sometimes very slow. Furthermore, sometimes one incomplete factor leads to much faster convergence than a much denser incomplete factor. We acknowledge that in some cases, when targeting larger relative residuals (like 10^{-2} or 10^{-5}), incomplete factorization and multigrid preconditioners are more effective than combinatorial preconditioners. This is evident in other combinatorial preconditioners [**63, 194**]. This is clearly shown in Figure 2.7.7.

These results show that the ability of our solver to detect inapproximable elements and to treat them separately allows it to solve problems that cause difficulty to other iterative

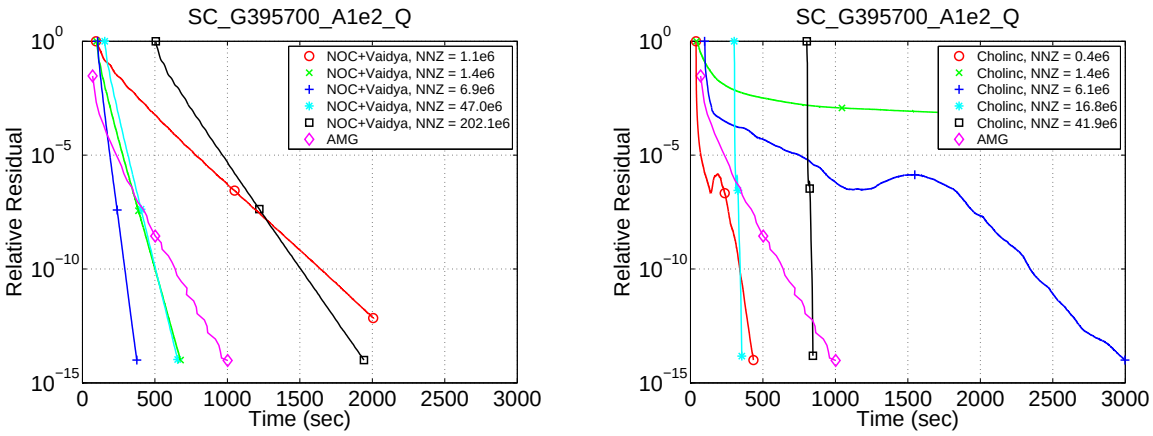


FIGURE 2.7.7. The relative norm of the residual as a function of the running time (and implicitly, of the iteration count) on one anisotropic problem, for our solver (left) and for incomplete Cholesky (right). The horizontal coordinate in which individual plots start indicates the time to construct and factor the preconditioner.

solvers. When there are few such elements, as there are here because the anisotropic shell is thin, these elements do not significantly increase the density of the factor of M. In problems in which most of the elements are inapproximable by SDD matrices, M would be similar to K and the characteristics of our solver would be similar to the characteristics of a direct solver.

This is perhaps the most important insight about our solver. As problems get harder (in the sense that more elements become inapproximable), its behavior becomes closer to that of a direct solver. As problems get harder we lose the ability to effectively sparsify the preconditioner prior to to factoring it. But unlike other solvers, our solver does not exhibit slow or failed convergence on these difficult problems.

2.7.7. Comparisons of Different Element-by-Element Approximations. We now explore additional heuristics for approximating K_e. The approximation methods that we compare are:

Nearly Optimal Clique (NOC): $L_e = ZDD^TZ$, where the columns of Z is the full set of edge vectors and D scales the columns of U^+Z to unit 2-norm. This method gives the strongest theoretical bound of the methods we tested on $\kappa(K_e, L_e)$: it is at most $n_e^2/2$ times larger than the best possible for an SDD approximation of K_e. Here and in the next four methods, we set the scaling factor α_e to be $\max \Lambda(K_e, L_e)$.

Nearly Optimal Star (NOS): $L_e = ZDD^TZ$, where the columns of Z are edge vectors that form a star, $\langle 1, -2\rangle, \langle 1, -3\rangle, \ldots, \langle 1, -n_e\rangle$ and D scales the columns of U^+Z to unit 2-norm. Sparser than the first but usually a worse approximation.

Uniform Clique (UC): L_e is the extension of the n_e-by-n_e matrix $B_{C(n_e)}$ to an n-by-n matrix. Computing L_e is cheap, but the approximation is only guaranteed to be good when K_e is very well conditioned. The low cost of this method stems from the facts that (1) $B_{C(n_e)}$ is a fixed matrix, and (2) $\alpha_n = \max \Lambda(K_e)$, so we

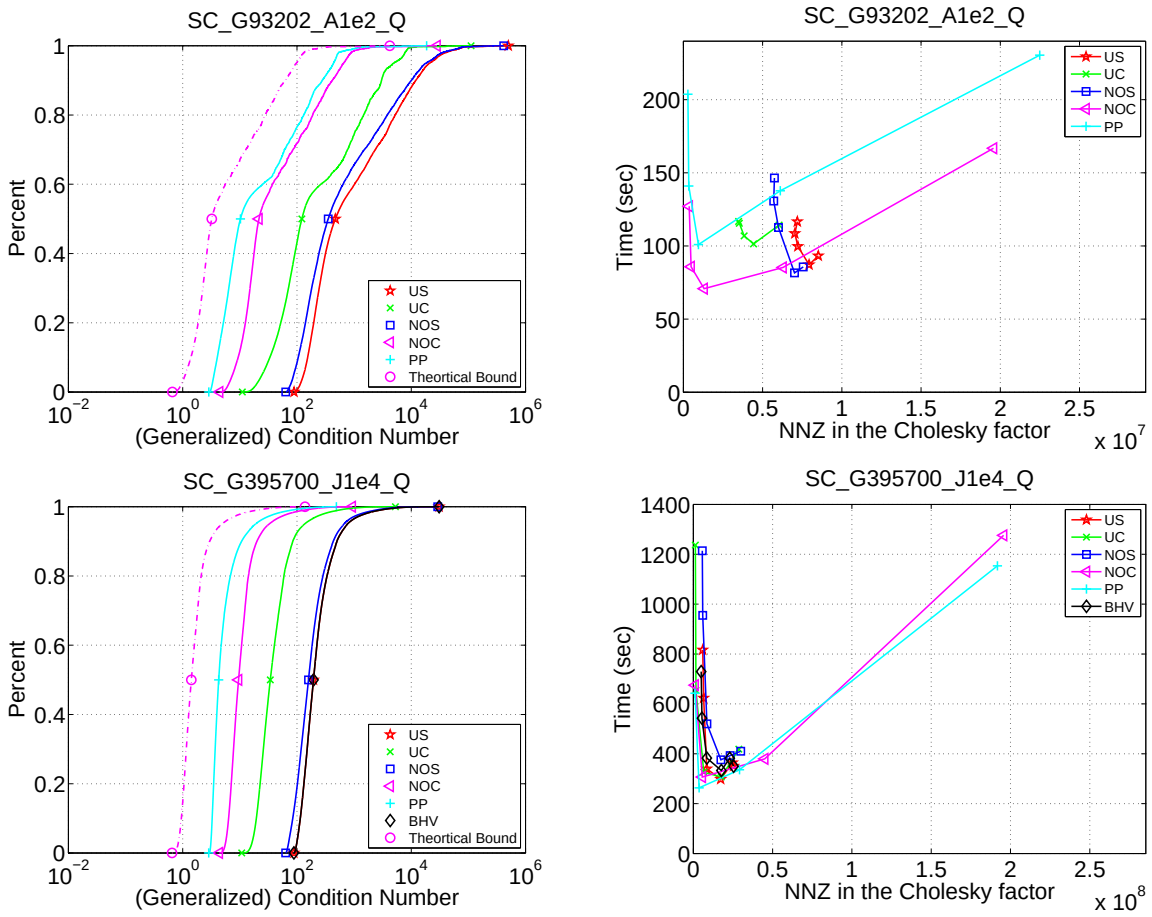


FIGURE 2.7.8. Different element-approximation methods. The graph on the left shows the distribution of $\kappa(K_e, L_e)$ for each approximation method. The method of Boman et al. is only applicable to well-conditioned elements, so it is not included in the graphs in the top row. The theoretical bound in the left figure is calculated using the fact that the spectral distance of NOC is within $n_e^2/2$ of the best possible. In the bottom left figure, the BHV (diamond markers) plot occludes the US (star markers) plot.

do not need to estimate an extreme generalized eigenvalue, only a single-matrix eigenvalue.

Uniform Star (US): L_e is the extension of the n_e-by-n_e matrix $B_{S(n_e)}$ to an n-by-n matrix. Sparser than the uniform clique, but more expensive, since we compute an extreme generalized eigenvalue to set α_e.

Positive Part (PP): $L_e = (K_e)_+$, defined in Section 2.2.6.

Boman et al. (BHV): $\alpha_e L_e$ is the Boman-Hendrickson-Vavasis approximation of K_e [**44**]. In their method, L_e is a uniform star, and the scaling factor α_e is computed from quantities associated with the finite-element discretization that produces K_e.

The results are shown in Figure 2.7.8. When element matrices are fairly well conditioned (bottom graphs), different approximation methods exhibit similar qualitative

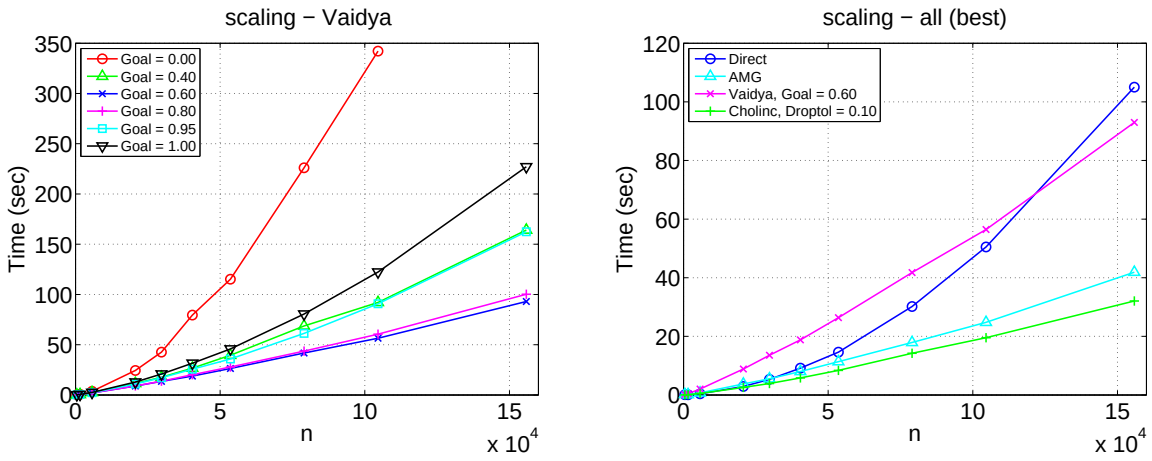


FIGURE 2.7.9. Solution times as a function of mesh size, on the same physical problem (a cube with uniform coefficients, discretized using linear tetrahedral elements). The graph on the left compares the running times of our solver with different levels of fill, and the graph on the right compares our solver (with the best-case fill level) with BoomerAMG and the direct solver. The fill in our solver is controlled by a parameter called goal in these graphs. A goal of 1 does not sparsify the approximation $M_{\leq t}$, and a goal of 0 sparsifies it as much as possible, resulting in a tree or forest graph structure for $L_{\leq t}$.

behaviors. But when there are ill-conditioned elements, naive approximation methods perform poorly. The results also show that the nearly-optimal approximation (which we used in all the other experiments) performs well relative to other approximation methods, but is usually not the best.

The comparison between PP and NOC in the top experiments in Figure 2.7.8 is interesting. The top left graph indicates that PP is a better approximation in that experiment. This was consistent in additional parameters that we explored and are not reported here: absolute condition number, number of elements that were inapproximable (beyond $t = 1000$), the distribution of the ratio $\kappa(K_e, L_e)/\kappa(K_e)$, sparsity of the preconditioner, and time to compute the approximation.

Nevertheless, the number of iterations with the NOC+Vaidya preconditioner is substantially smaller (about half) of the number of iterations with the PP+Vaidya preconditioner. The bottom line is that NOC+Vaidya performs better than PP+Vaidya. We do not have a good explanation for this; it may be a good subject for future research.

2.7.8. Running Times for Growing Problem Sizes. The results in Figure 2.7.9 present the growth in running time of our solver as the problem size grows. For very dense and very sparse preconditioners, the growth is highly nonlinear. This is consistent with the theory of sparse direct solvers on one side and with the theory of Vaidya's preconditioners on the other. For intermediate levels of fill, running times grow more slowly, but they still seem to grow superlinearly.

2.8. Open Problems

This chapter raises several interesting questions and challenges for further research. We mention four. One challenge is to extend the optimal-scaling method of Braatz and Morari [**49**] to rank-deficient and rectangular matrices. It may be possible to use the reduction in [**219**] to solve this problem, but we have not explored this. It is not even clear whether van der Sluis's nearly-optimal scaling for rectangular matrices [**216**] is also nearly-optimal for rank-deficient matrices. Another interesting question is to find a reliable and cheap-to-compute estimate of the spectral distance between a given symmetric positive (semi)definite matrix A and the closest SDD matrix to A. Our method can be used to estimate this distance, but for large matrices the method is expensive and its estimates are loose. We have also shown that $\kappa(A)$ is an upper bound on that distance, but this bound can be arbitrarily loose. The third and probably most important challenge is to find better ways to exploit the splitting $K = K_{\leq t} + K_{>t}$. There may be several ways to exploit it. For example, it is probably possible to build better preconditioners by sparsifying $L_{\leq t}$ with the objective to reduce fill in the Cholesky factor of $M_{\leq t} + K_{>t}$; the algorithm that we used for the sparsification phase ignores $K_{>t}$ and only tries to reduce fill in the factor of $M_{\leq t}$. The fourth question is connected to the selection of parameters. Our algorithm has several parameters: the approximation topology, approximation cutoff (t), and the level sparsification. The best choice for the parameters is affected by two other parameters: characteristics of the problem at hand, and the convergence tolerance. In our experiments, we fix the convergence tolerance, and find heuristically a set of parameters that work reasonably well for most problems. It is interesting to see if auto-tuning can help choose parameters, either beforehand or during run-time, that give best results for the class of problems a user wishes to solve. Auto-tuning here refers to building a performance model, perhaps over several runs, of how the algorithms behaves over a set of hard-to-choose tuning parameters, and using that model to choose good values of the parameters. This approach is widely used for dense linear algebra ([**223**]) and FFT ([**103**]), and being increasingly explored for sparse matrix problem where one has to wait until the matrix is available at run-time to do the exploration.

CHAPTER 3

Using Perturbed QR Factorizations to Solve Linear Least-Squares Problems

3.1. Introduction

This chapter[1] shows that the R factor from the QR factorization of a perturbation $\hat{A}$ of a matrix A is an effective least-squares preconditioner for A. More specifically, we show the R factor of the perturbation is an effective preconditioner if the perturbation can be expressed by adding/or dropping a few rows from A or if it can be expressed by replacing a few columns.

If A is rank deficient or highly ill-conditioned, the R factor of a perturbation $\hat{A}$ is still an effective preconditioner if $\hat{A}$ is well-conditioned. Such an R factor can be used in LSQR (an iterative least-squares solver [**166**]) to efficiently and reliably solve a regularization of the least-squares problem. We present an algorithm for adding rows with a single nonzero to A to improve its conditioning; it attempts to add as few rows as possible.

We also show that if an arbitrary preconditioner M is effective for $\hat{A}^*\hat{A}$ (where $\hat{A}^*$ is the adjoint of $\hat{A}$), in the sense that the generalized condition number of $(\hat{A}^*\hat{A}, M)$ is small, then M is also an effective preconditioner for A^*A. This shows that we do not necessarily need the R factor of the perturbation $\hat{A}$; we can use M as a preconditioner instead.

This chapter provides a comprehensive spectral analysis of the generalized spectrum of matrix pencils that arise from row and column perturbations. The analysis shows that if the number of rows/columns that are added, dropped, or replaced is small, then most of the generalized eigenvalues are 1 (or lie in some interval when R is not an exact factor). We bound the number of *runaway* eigenvalues, which are the ones that are not 1 (or outside the interval), which guarantees rapid convergence of LSQR.

These results generalize a simple observation. Let A be a given matrix and let $\hat{A} = \left[\begin{smallmatrix} A \\ B \end{smallmatrix}\right]$. Then

$$\begin{aligned} \left(\hat{A}^*\hat{A}\right)^{-1} A^*A &= \left(A^*A + B^*B\right)^{-1} A^*A \\ &= \left(A^*A + B^*B\right)^{-1} \left(A^*A + B^*B - B^*B\right) \\ &= I - \left(A^*A + B^*B\right)^{-1} B^*B \; . \end{aligned} \tag{3.1.1}$$

[1]The results in this chapter also appear in a paper co-authored with Esmond Ng and Sivan Toledo which was published in the *SIAM Journal on Matrix Analysis and Applications* [**20**]; proof of Theorem 3.3.4 appeared in a separate technical report [**19**]. This chapter also contains a more general version of Theorem 3.4.1. This general version was shortened in the paper due to a requests from the reviewers to shorten the paper. The numerical experiment in section 3.7.1, which was dropped from the paper due to lack of space, is included as well.

The rank of second term on the last line is at most the rank of B, so if B has low rank, then $\left(\hat{A}^*\hat{A}\right)^{-1} A^*A$ is a low-rank perturbation of the identity. A symmetric rank-k perturbation of the identity has at most k non-unit eigenvalues, which in exact arithmetic guarantees convergence in k iterations in several Krylov-subspace iterations. Therefore, the Cholesky factor of $\hat{A}^*\hat{A}$ (which is also the R factor of $\hat{A}$) is a good least-squares preconditioner for A. The same analysis extends to the case where we drop rows of A. This idea has been used by practitioners [**105**]. A special case of this idea appears in the graph sparsification literature. In this case deleting or adding a row correpsonds to deleting or adding an edge. See [**180, 199**].

We generalize this result in additional ways: to the case where $\hat{A}$ is singular, to column exchanges, and to preconditioners for $\hat{A}$ rather than its R factor. We also bound the size of the non-unit eigenvalues, which is important when A is rank deficient.

The rest of this chapter presents relevant background, our spectral analysis of perturbed factorizations, an algorithm for choosing the perturbations, and numerical results.

3.2. Background

3.2.1. LSQR, an Iterative Krylov-Subspace Least-Squares Solver. LSQR is a Krylov-subspace iterative method for solving the least-squares problem $\min_x \|Ax-b\|_2$. The method was developed by Paige and Saunders in 1982 [**166**].

The algorithm is based on the bidiagonalization procedure due to Golub and Kahan [**108**]. A sequence of approximations $\{x_k\}$ is generated such that the residual $\|Ax_k - b\|_2$ decreases monotonically. The sequence $\{x_k\}$ is, analytically, identical to the sequence generated by the conjugate gradients algorithm [**124, 68**] applied to A^*A. Therefore, the convergence theory of conjugate gradients applied to A^*A applies directly to the behavior of LSQR. In particular, the convergence of LSQR is governed by the distribution of the eigenvalues of A^*A (and can be bounded using its condition number). Another useful observation, which we will use extensively, is that if the matrix A^*A has l distinct eigenvalues, then LSQR will converge in at most l iterations (this is a simple consequence of the minimization property of conjugate gradients).

The relationship between the condition number and the convergence of LSQR and the relationship between the number of distinct eigenvalues and convergence of LSQR are essentially a special case of a result given by [**161**, Theorem 2.3] (Ng attributes the result to Van der Vorst). This result analyzes the convergence of Conjugate Gradients when all but $k + r$ eigenvalues lie outside a given interval. The result can also be adapted to singular matrices and LSQR, and it is used in Section 3.5.

In this chapter we use the preconditioned version of LSQR. Given an easy-to-invert preconditioner R, we have

$$\min_x \|Ax - b\|_2 = \min_x \|AR^{-1}Rx - b\|_2 .$$

This allows us to solve $\min_x \|Ax-b\|_2$ in two phases. We first solve $\min_y \|AR^{-1}y-b\|_2$ and then solve $Rx = y$. The first phase is solved by LSQR. The convergence is now governed by spectrum (set of eigenvalues) of $R^{-*}A^*AR^{-1}$, which is hopefully more clustered than the spectrum of A^*A. The spectrum of $R^{-*}A^*AR^{-1}$ is identical to the set of generalized eigenvalues $A^*Ax = \lambda R^*Rx$. We analyze these generalized eigenvalues.

3.2.2. Sparse QR Factorizations. In some of the applications that we describe below, the preconditioner is the R factor from a QR factorization of a perturbation of A.

The main approach to exploiting the sparsity of A in a QR factorization is to attempt to minimize the fill in the R factor. Since the R factor of A is also the Cholesky factor of A^*A, we can use an algorithm that reduces fill in sparse Cholesky by symmetrically permuting the rows and columns of A^*A [**104**]. Such a permutation is equivalent to a column permutation of A. Many algorithms can compute such a permutation without ever computing A^*A or its sparsity pattern [**76, 50**].

When A is well-conditioned it is possible to solve the least-squares problem $\min_x \|Ax - b\|_2$ using the QR factorization of A. When A is ill-conditioned it may be useful to *regularize* the equation by truncating singular values that are too small (see subsection 2.7.2 in [**40**]). A cheaper but effective regularization method approximates the truncated solution using a *rank revealing* QR factorization of A [**57, 58**].

Designing a sparse rank revealing QR factorization is a challenging task. There are basically two techniques to compute a rank revealing QR factorization. The first method, which is guaranteed to generate a rank revealing factorization, is to find a regular QR factorization and refine it to a rank revealing factorization [**57**]. In the sparse setting the correction phase can be expensive and can produce considerable fill. We can also find a rank revealing QR factorization using column pivoting [**106**]. This method can fail to produce a rank revealing factorization, but it usually does [**99**]. When A is sparse, extensive column pivoting destroys the fill reducing preordering, hence increasing fill. Column pivoting also requires more complex data structures and reduces the value of the symbolic analysis phase of the factorization.

Sparse rank-revealing QR factorizations do use column pivoting, usually with heuristics to restrict pivot selection (to avoid catastrophic fill). The heuristic nature of the pivot selection has a price: the ability of these factorizations to reveal rank is reduced compared to strict pivoting [**60, 171**]. Some algorithms [**37, 27**] address this problem by adding a correction phase at the end. The restricted pivoting in the first phase is aimed at reducing the amount of work that is needed in the second phase. We use this correction idea in one of our algorithms.

A sparse QR algorithm can be organized in three ways. The method of George and Heath [**104**] rotates rows of A into R using Givens rotations. The multifrontal method [**151**] uses Householder reflections, and so does the left-looking method [**74**]. It is not possible to incorporate column pivoting into methods that are based on rotating rows into R, because there is no way to estimate the effect of pivoting on a particular column. Consequently, column-pivoting QR factorizations are column-oriented, not row oriented, in which case Householder reflections are usually used rather than Givens rotations.

3.3. Preliminaries

In this section we give some basic definitions in order to establish terminology and notation. These definitions are not new. We also restate known theorems that we will use extensively in our theoretical analysis.

Definition 3.3.1. Let S and T be n-by-n complex matrices. We say that a scalar λ is a *finite generalized eigenvalue* of the matrix pencil (pair) (S, T) if there is a vector $v \neq 0$

such that

$$Sv = \lambda Tv$$

and $Tv \neq 0$. We say that ∞ is a *infinite generalized eigenvalue* of (S,T) if there exists a vector $v \neq 0$ such that $Tv = 0$ but $Sv \neq 0$. Note that ∞ is an eigenvalue of (S,T) if and only if 0 is an eigenvalue of (T,S). The finite and infinite eigenvalues of a pencil are *determined eigenvalues* (the eigenvector uniquely determines the eigenvalue). If both $Sv = Tv = 0$ for a vector $v \neq 0$, we say that v is an *indeterminate eigenvector*, because $Sv = \lambda Tv$ for any scalar λ.

Throughout the chapter eigenvalues are ordered from smallest to largest. We will denote the kth eigenvalue of S by $\lambda_k(S)$, and the kth determined generalized eigenvalue of (S,T) by $\lambda_k(S,T)$. Therefore $\lambda_1(S) \leq \cdots \leq \lambda_l(S)$ and $\lambda_1(S,T) \leq \cdots \leq \lambda_d(S,T)$, where l is the number of eigenvalues S has, and d is the number of determined eigenvalues that (S,T) has.

The solution of the least-squares equation $\min_x \|Ax - b\|_2$ is also the solution of the equation $A^*Ax = A^*x$. Matrix A^*A is Hermitian positive semidefinite. The LSQR method is actually a Krylov-space method on A^*A, and a preconditioner for the method is Hermitian positive semidefinite too. Therefore, the matrix pencils that we will consider in this chapter are *Hermitian positive semidefinite* (H/PSD) pairs.

Definition 3.3.2. A pencil (S,T) is *Hermitian positive semidefinite* (H/PSD) if S is Hermitian, T is Hermitian positive semidefinite, and $\text{null}(T) \subseteq \text{null}(S)$.

The generalized eigenvalue problem on H/PSD pencils is, mathematically, a generalization of the Hermitian eigenvalue problem. In fact, the generalized eigenvalues of an H/PSD can be shown to be the eigenvalues of an equivalent Hermitian matrix. The proof appears in the Appendix. Based on this observation it is easy to show that other eigenvalue properties of Hermitian matrices have an analogy for H/PSD pencils. For example, an H/PSD pencil, (S,T), has exactly $\text{rank}(T)$ determined eigenvalues (counting multiplicity), all of them finite and real.

A useful tool for analyzing the spectrum of an Hermitian matrix is the *Courant-Fischer Minimax Theorem* [**109**].

Theorem 3.3.3. *(Courant-Fischer Minimax Theorem) Suppose that $S \in \mathbb{C}^{n\times n}$ is an Hermitian matrix, then*

$$\lambda_k(S) = \min_{\dim(U)=k} \max_{\substack{x \in U \\ x \neq 0}} \frac{x^*Sx}{x^*x}$$

and

$$\lambda_k(S) = \max_{\dim(V)=n-k+1} \min_{\substack{x \in V \\ x \neq 0}} \frac{x^*Sx}{x^*x}.$$

As discussed above, the generalized eigenvalue problem on H/PSD pencils is a generalization of the eigenvalue problem on Hermitian matrices. Therefore, there is a natural generalization of Theorem 3.3.3 to H/PSD pencils, which we refer to as the *Generalized Courant-Fischer Minimax Theorem.* We now state the theorem. Although it is a natural

generalization of Theorem 3.3.3 we have not found any reference to it in the literature, and since we believe it is of interest even separately we present its proof.

Theorem 3.3.4. *(Generalized Courant-Fischer Minimax Theorem) Suppose that* $S \in \mathbb{C}^{n\times n}$ *is an Hermitian matrix and that* $T \in \mathbb{C}^{n\times n}$ *is an Hermitian positive semidefinite matrix such that* $\operatorname{null}(T) \subseteq \operatorname{null}(S)$. *For* $1 \leq k \leq \operatorname{rank}(T)$ *we have*

$$\lambda_k(S,T) = \min_{\substack{\dim(U)=k \\ U \perp \operatorname{null}(T)}} \max_{x\in U} \frac{x^*Sx}{x^*Tx}$$

and

$$\lambda_k(S,T) = \max_{\substack{\dim(V)=\operatorname{rank}(T)-k+1 \\ V \perp \operatorname{null}(T)}} \min_{x\in V} \frac{x^*Sx}{x^*Tx}\,.$$

We begin by stating and proving a generalization of the Courant-Fischer Theorem for pencils of Hermitian positive definite matrices.

Theorem 3.3.5. *Let* $S, T \in \mathbb{C}^{n\times n}$ *be Hermitian matrices. If* T *is also positive definite then*

$$\lambda_k(S,T) = \min_{\dim(U)=k} \max_{x\in U} \frac{x^*Sx}{x^*Tx}$$

and

$$\lambda_k(S,T) = \max_{\dim(V)=n-k+1} \min_{x\in V} \frac{x^*Sx}{x^*Tx}\,.$$

Proof. Let $T = L^*L$ be the Cholesky factorization of B. Let U be some k-dimensional subspace of $\mathbb{C}^n$, let $x \in U$, and let $y = Lx$. Since T is Hermitian positive definite (hence nonsingular), the subspace $W = \{Lx : x \in U\}$ has dimension k. Similarly, for any k-dimensional subspace W, the subspace $U = \{L^{-1}x : x \in W\}$ has dimension k. We have

$$\frac{x^*Sx}{x^*Tx} = \frac{x^*L^*L^{-*}SL^{-1}Lx}{x^*L^*Lx} = \frac{y^*L^{-*}SL^{-1}y}{y^*y}\,.$$

By applying the Courant-Fischer to $L^{-*}SL^{-1}$, we obtain

$$\begin{aligned}\lambda_k(L^{-*}SL^{-1}) &= \min_{\dim(W)=k} \max_{y\in W} \frac{y^*L^{-*}SL^{-1}y}{y^*y} \\ &= \min_{\dim(U)=k} \max_{x\in S} \frac{x^*Sx}{x^*Tx}\,.\end{aligned}$$

The generalized eigenvalues of (S,T) are exactly the eigenvalues of $L^{-*}SL^{-1}$ so the first equality of the theorem follows. The second equality can be proved using a similar argument. □

Before proving the generalized version of the Courant-Fischer Minimax Theorem we show how to convert an Hermitian positive semidefinite problem to an Hermitian positive definite problem.

Lemma 3.3.6. *Let $S, T \in \mathbb{C}^{n\times n}$ be Hermitian matrices. Assume that T is also a positive semidefinite and that* $\text{null}(T) \subseteq \text{null}(S)$. *For any $Z \in \mathbb{C}^{n\times \text{rank}(T)}$ with* $\text{rank}(Z) = \text{rank}(T)$, *the determined generalized eigenvalues of (S, T) are exactly the generalized eigenvalues of (Z^*SZ, Z^*TZ).*

PROOF. We first show that Z^*TZ has full rank. Suppose that $Z^*TZv = 0$. We have $TZv \in \text{null}(Z^*)$. Therefore, $TZv \perp \text{rank}(Z) = \text{rank}(T)$. Obviously $TZv \in \text{rank}(T)$, so we must have $v = 0$. Since $\text{null}(Z^*TZ) = \{0\}$, the matrix Z^*TZ has full rank.

Suppose that λ is a determined eigenvalue of (S, T). We will show that it is a determined eigenvalue of (Z^*SZ, Z^*TZ). The pencil (Z^*SZ, Z^*TZ) has exactly $\text{rank}(Z^*TZ)$ determined eigenvalues. We will show that Z^*TZ is full rank, so the pencil (Z^*SZ, Z^*TZ) has exactly $\text{rank}(T)$ eigenvalues. Since the pencil (S, T) has exactly $\text{rank}(T)$ determined eigenvalues, each of them an eigenvalue of (Z^*SZ, Z^*TZ), this will conclude the proof.

Now let μ be an eigenvalue of (Z^*SZ, Z^*TZ). It must be determined, since Z^*TZ has full rank. Let y be the corresponding eigenvector, $Z^*SZy = \mu Z^*TZy$, and let $x = Zy$. Now there are two cases. If $\mu = 0$, then $SZy = Sx = 0$ (since Z^* has full rank and at least as many columns as rows). The vector x is in $\text{rank}(Z) = \text{rank}(T)$, $Tx \neq 0$. This implies that $\mu = 0$ is also a determined eigenvalue of (S, T).

If $\mu \neq 0$ the analysis is a bit more difficult. Clearly, $TZy \in \text{rank}(T) = \text{rank}(Z)$. But $\text{rank}(Z) = \text{rank}(Z^{*+})$ [**34**, Proposition 6.1.6.vii], so $Z^{*+}Z^*TZy = TZy$ [**34**, Proposition 6.1.7]. We claim that $SZy \in \text{rank}(Z)$. If it were not so, Zy must be in $\text{null}(T) \subseteq \text{null}(S)$, but μ would have to be zero. Therefore, we also have $Z^{*+}Z^*SZy = SZy$, so by multiplying $Z^*SZy = \mu Z^*TZy$ by Z^{*+} we see that μ is an eigenvalue of (S, T). □

We are now ready to prove Theorem 3.3.4, the generalization of the Courant-Fischer Minimax Theorem. The technique is simple: we use Lemma 3.3.6 to reduce the problem to a smaller-sized full-rank problem, apply Theorem 3.3.5 to characterize the determined eigenvalues in terms of subspaces, and finally show a complete correspondence between the subspaces used in the reduced pencil and subspaces used in the original pencil.

PROOF. (of the Generalized Courant-Fischer Minimax Theorem) Let $Z \in \mathbb{C}^{n\times \text{rank}(T)}$ have $\text{rank}(Z) = \text{rank}(T)$. We have

$$\lambda_k(S, T) = \lambda_k(Z^*SZ, Z^*TZ) = \min_{\dim(W) = k} \max_{x\in W} \frac{x^*Z^*SZx}{x^*Z^*TZx}$$

and

$$\lambda_k(S, T) = \lambda_k(Z^*SZ, Z^*TZ) = \max_{\dim(W) = \text{rank}(T) - k + 1} \min_{x\in W} \frac{x^*Z^*TZx}{x^*Z^*TZx}.$$

The leftmost equality in each of these equations follows from Lemma 3.3.6 and the rightmost one follows from Theorem 3.3.5.

We now show that for every k-dimensional subspace $U \subseteq \mathbb{C}^n$ with $U \perp \text{null}(T)$, there exists a k-dimensional subspace $W \subseteq \mathbb{C}^{\text{rank}(T)}$ such that

$$\left\{ \frac{x^*Sx}{x^*Tx} : x \in U \right\} = \left\{ \frac{y^*Z^*SZy}{y^*Z^*TZy} : y \in W \right\},$$

and vice versa. The validity of this claim establishes the min-max side of the theorem.

We first need to show that $k \leq \text{rank}(T)$. This is true because every vector in U is in range(T), so its dimension must be at most rank(T).

Define $W = \left\{y \in \mathbb{C}^{\text{rank}(T)} : Zy \in U\right\}$. Let $b_1, \ldots, b_k$ be a basis for U. Because $U \perp \text{null}(T)$, $b_j \in \text{rank}(T)$, so there is a y_j such that $Zy_j = b_j$. Therefore, dimension of W is at most k. Now let the vectors y_i's be a basis of W and define $b_i = Zy_i$. The b_i's span U, so there are at most k of them, so the dimension of W is at least k. Therefore, it is exactly k.

Every $x \in U$ is orthogonal to null(T), so it must be in rank(T). There exist a $y \in \mathbb{C}^{\text{rank}(T)}$ such that $Zy = x$. So we have $x^*Sx/x^*Tx = y^*Z^*SZy/y^*Z^*TZy$. Combining with the fact that $y \in W$, we have shown inclusion of one side. Now suppose $y \in W$. Define $x = Zy \in U$. Again we have $x^*Sx/x^*Tx = y^*Z^*SZy/y^*Z^*TZy$, which shows the other inclusion.

Now we will show that for every k-dimensional subspace W there is a subspace U that satisfies the claim. Define $U = \{Zy : y \in W\}$. Because Z has full rank, $\dim(U) = k$. Also, $U \subseteq \text{rank}(Z) = \text{rank}(T)$ so $U \perp \text{null}(T)$. The equality of the Rayleigh-quotient sets follows from taking $y \in W$ and $x = Zy \in U$ or vice versa. □

3.4. Spectral Theory

The generalized spectrum of (A^*A, A^*A) is very simple: the pencil has rank(A) eigenvalues that are 1 and the rest are indeterminate. This section characterizes the structure of spectra of perturbed pencils, $(A^*A, A^*A + B^*B - C^*C)$ and $(A^*A, \tilde{A}^*\tilde{A})$, where $A = \begin{bmatrix} D & E \end{bmatrix}$ and $\tilde{A} = \begin{bmatrix} D & F \end{bmatrix}$.

These perturbations of A^*A shift some of the eigenvalues of (A^*A, A^*A). We call the eigenvalues that moved away from 1 *runaway* eigenvalues. This section analyzes these runaway eigenvalues, which govern the convergence of LSQR when a factorization or an approximation of the perturbed matrix is used as a preconditioner.

To keep the notation simple, we define the symmetric product A^*A, where A is an m-by-n matrix, to be the n-by-n zero matrix when $m = 0$.

3.4.1. Counting Runaway Eigenvalues. We begin by bounding the number of runaway eigenvalues. We show that when B, C, E, and F have low rank, the generalized eigenvalue 1 has high multiplicity in these pencils. We also bound the multiplicity of zero and indeterminate eigenvalues. The first result that we present bounds the number of runaways (and other aspects of the structure of the spectrum) when we add and/or subtract a symmetric product from a matrix.

Theorem 3.4.1. *Let $A \in \mathbb{C}^{m\times n}$ and let $B \in \mathbb{C}^{k\times n}$ and $C \in \mathbb{C}^{r\times n}$ for some $1 \leq k+r < n$. Define*

$$\chi = \begin{bmatrix} B \\ C \end{bmatrix}.$$

The following claims hold:

(1) *In the pencil $(A^*A, A^*A + B^*B - C^*C)$, at most $\text{rank}\,\chi) \leq k + r$ generalized determined eigenvalues may be different from 1 (counting multiplicities).*

(2) *If 1 is not a generalized eigenvalue of the pencil (B^*B, C^*C) and $A^*A + B^*B - C^*C$ is full rank, then (a) the pencil $(A^*A, A^*A + B^*B - C^*C)$ does not have indeterminate eigenvectors, (b) the multiplicity of the eigenvalue 1 is exactly*

$\dim \text{null}(\chi) \geq n - k - r$, *and (c) the multiplicity of the zero eigenvalue is exactly* $\dim \text{null}(A)$.

(3) *The sum pencil* $(A^*A, A^*A + B^*B)$ *cannot have an infinite eigenvalue and all its eigenvalues are in the interval* $[0, 1]$.

PROOF. First, notice that $v \in \text{null}(\chi)$ if and only if $v \in \text{null}(B) \cap \text{null}(C)$. We prove most of the claims by showing that if v is an eigenvector of the pencil $(A^*A, A^*A + B^*B - C^*C)$ corresponding to the eigenvalue λ, then the relationship of v to the null spaces of A and the relationship of B^*Bv to C^*Cv, determine λ in the following way:

	$v \in \text{null}(A)$	$v \notin \text{null}(A)$
$B^*Bv = C^*Cv$	indeterminate	$\lambda = 1$
$B^*Bv \neq C^*Cv$	$\lambda = 0$	$\lambda \neq 0$ and $\lambda \neq 1$

If $v \in \text{null}(A)$ and $B^*Bv = C^*Cv$ then clearly both $A^*Av = 0$ and $(A^*A + B^*B - C^*C)v = 0$ so v is an indetermined eigenvector of $(A^*A, A^*A + B^*B - C^*C)$.

Let $v \notin \text{null}(A)$ be a vector such that $B^*Bv = C^*Cv$. Therefore

$$(A^*A + B^*B - C^*C)\, v = A^*Av \neq 0\ ,$$

so v must be a finite generalized eigenvector of $(A^*A, A^*A + B^*B - C^*C)$ that corresponds to the eigenvalue 1.

If $v \in \text{null}(A)$ and $B^*Bv \neq C^*Cv$, then $A^*Av = 0$ and $(A^*A + B^*B - C^*C)v = A^*Av + B^*Bv - C^*Cv = B^*Bv - C^*Cv \neq 0$, so v is an eigenvector corresponding to 0.

If $v \notin \text{null}(A)$ and $B^*Bv \neq C^*Cv$, then λ can be neither 0 nor 1. It cannot be 0 because $A^*Av \neq 0$. It cannot be 1 because that would imply $B^*Bv - C^*Cv = 0$ which is a contradiction to the assumption that $B^*Bv \neq C^*Cv$.

To establish Claim 1 notice that if $v \in \text{null}(B) \cap \text{null}(C) = \text{null}(\chi)$ then clearly $B^*Bv = C^*Cv$. So, if v is a determined generalized eigenvector corresponding to a eigenvalue different from 1, then $v \notin \text{null}(\chi)$. Therefore, the dimension of the space spanned by these vectors is bounded by $\text{rank}(\chi) \leq k + r$, which bounds the number of such eigenvalues.

We now turn our attention to Claim 2. Assume that $A^*A + B^*B - C^*C$ is full rank and 1 is not a generalized eigenvalue of the pencil (B^*B, C^*C). Since $A^*A + B^*B - C^*C$ is full rank then for every vector $v \neq 0$ we have $(A^*A + B^*B - C^*C)v \neq 0$, so vector v cannot be an indetermined eigenvector of $(A^*A, A^*A + B^*B - C^*C)$, and the pencil has no indetermined eigenvalues. The multiplicity of the eigenvalue 1 follows from the fact that if v is a generalized eigenvector corresponding to 1 then we must have $B^*Bv = C^*Cv$. Since 1 is not a generalized eigenvalue of the pencil (B^*B, C^*C) then we must have $B^*Bv = C^*Cv = 0$. Therefore, the space of eigenvectors corresponding to 1 is exactly $\text{null}(\chi)$. The multiplicity of the eigenvalue 0 follows from the fact that every $0 \neq v \in \text{null}(A)$ satisfies $A^*Av = 0$ and $(A^*A + B^*B - C^*C)v \neq 0$ (because $A^*A + B^*B - C^*C$ has full rank). Therefore, v is indeed a generalized eigenvecctor. The converse is true from the table.

We now show that Claim 3 holds. In the sum pencil $(A^*A, A^*A + B^*B)$, λ cannot be infinite. Suppose for contradiction that it is. Then $(A^*A + B^*B)v = 0$ but $A^*Av \neq 0$. We get $v^*(A^*A + B^*B)v = 0$, but $v^*(A^*A + B^*B)v = v^*A^*Av + v^*B^*Bv > 0$. To show that the generalized eigenvalues are in the range $[0, 1]$, notice that if $\lambda \neq 0$ is a finite

generalized eigenvalue of $(A^*A, A^*A + B^*B)$ then there exist a vector $v \neq 0$ such that

$$\begin{aligned} \lambda &= \frac{v^*A^*Av}{v^*(A^*A + B^*B)v} \\ &= \frac{v^*A^*Av}{v^*A^*Av + v^*B^*Bv} . \end{aligned}$$

Since both v^*A^*Av and v^*B^*Bv are greater than or equal to 0 the claim follows immediately.

For completeness, we also characterize the infinite eigenvectors of the pencil $(A^*A, A^*A + B^*B - C^*C)$. Such an eigenvector v satisfies

$$\begin{aligned} A^*Av &\neq 0 \\ (A^*A + B^*B - C^*C)\, v &= 0 , \end{aligned}$$

or

$$(A^*A + B^*B)v = C^*Cv \neq 0 .$$

Thus, v is a generalized eigenvector of $(A^*A + B^*B, C^*C)$ corresponding to the eigenvalue 1. □

The second result of this section characterizes the generalized spectra of symmetric products that are formed by modifying a set of columns in a given matrix A. We denote the columns of A that are not modified in the factorization by D, the columns that are to be modified by E, and the new value in those columns by F.

Theorem 3.4.2. *Let $D \in \mathbb{C}^{m\times n}$ and let $E \in \mathbb{C}^{m\times k}$ and $F \in \mathbb{C}^{m\times k}$ for some $1 \leq k < n$. Let*

$$A = \begin{bmatrix} D & E \end{bmatrix} \in \mathbb{C}^{m\times(n+k)}$$

and let

$$\tilde{A} = \begin{bmatrix} D & F \end{bmatrix} \in \mathbb{C}^{m\times(n+k)} .$$

In the pencil $(A^*A, \tilde{A}^*\tilde{A})$, at least $n - k$ generalized finite eigenvalues are 1.

Proof. Expanding A^*A and $\tilde{A}^*\tilde{A}$, we obtain

$$A^*A = \begin{bmatrix} D^* \\ E^* \end{bmatrix} \begin{bmatrix} D & E \end{bmatrix} = \begin{bmatrix} D^*D & D^*E \\ E^*D & E^*E \end{bmatrix}$$

and

$$\tilde{A}^*\tilde{A} = \begin{bmatrix} D^* \\ F^* \end{bmatrix} \begin{bmatrix} D & F \end{bmatrix} = \begin{bmatrix} D^*D & D^*F \\ F^*D & F^*F \end{bmatrix} .$$

Let S be the vector space in $\mathbb{C}^{n+k}$ defined by

$$S = \left\{ \begin{bmatrix} v \\ 0 \end{bmatrix} \,:\, v \in \mathbb{C}^n \text{ such that } E^*Dv = F^*Dv \right\} .$$

Clearly, $\dim S = \dim \operatorname{null}(E^*D - F^*D) = n - \operatorname{rank}(E^*D - F^*D)$. The matrix $E^*D - F^*D$ has k rows so $\operatorname{rank}(E^*D - F^*D) \leq k$, which implies $\dim S \geq n - k$.

Let v be a vector such that $E^*Dv = F^*Dv$. The vector $\begin{bmatrix} v \\ 0 \end{bmatrix}$ is a generalized eigenvector of $(A^*A, \tilde{A}^*\tilde{A})$ corresponding to the eigenvalue 1, because

$$A^*A \begin{bmatrix} v \\ 0 \end{bmatrix} = \begin{bmatrix} D^*D & D^*E \\ E^*D & E^*E \end{bmatrix} \begin{bmatrix} v \\ 0 \end{bmatrix} = \begin{bmatrix} D^*Dv \\ E^*Dv \end{bmatrix}$$
$$= \begin{bmatrix} D^*Dv \\ F^*Dv \end{bmatrix} = \begin{bmatrix} D^*D & D^*F \\ F^*D & F^*F \end{bmatrix} \begin{bmatrix} v \\ 0 \end{bmatrix} = \tilde{A}^*\tilde{A} \begin{bmatrix} v \\ 0 \end{bmatrix}.$$

Since S is a subset of the generalized eigenspace of $(A^*A, \tilde{A}^*\tilde{A})$ corresponding to the eigenvalue 1, the multiplicity of 1 as a generalized eigenvalue is at least $\dim S \geq n-k$. □

3.4.2. Runaways in a Preconditioned System. We now show that if a preconditioner M is effective for a matrix A^*A, then it is also effective for the perturbed matrices $A^*A+B^*B-C^*C$ and $\tilde{A}^*\tilde{A}$. If the rank of the matrices B, C, E, and F is low, then most of the generalized eigenvalues of the perturbed preconditioned system will be bounded by the extreme generalized eigenvalues of the unperturbed preconditioned system. In other words, the number of runaways is still guaranteed to be small, but the non-runaways are not necessarily at 1: they can move about the interval whose size determines the condition number of the original preconditioned system. [**161**, Theorem 2.3] shows that this spectral characterization guarantees rapid convergence; in exact arithmetic, after an iteration for each row in B and C, the convergence rate bound is governed by the unperturbed condition number (after two iterations for every column exchanged for column perturbations).

Theorem 3.4.3. *Let $A \in \mathbb{C}^{m\times n}$ and let $B \in \mathbb{C}^{k\times n}$ and $C \in \mathbb{C}^{r\times n}$ for some $1 \leq k+r < n$. Let $M \in \mathbb{C}^{n\times n}$ be an Hermitian positive semidefinite matrix. Suppose that* $\mathrm{null}(M) \subseteq \mathrm{null}(A^*A)$, $\mathrm{null}(M) \subseteq \mathrm{null}(B^*B)$ *and* $\mathrm{null}(M) \subseteq \mathrm{null}(C^*C)$. *If*

$$\alpha \leq \lambda_1(A^*A, M) \leq \lambda_{\mathrm{rank}(M)}(A^*A, M) \leq \beta\,.$$

then

$$\alpha \leq \lambda_{r+1}(A^*A + B^*B - C^*C, M) \leq \lambda_{\mathrm{rank}(M)-k}(A^*A + B^*B - C^*C, M) \leq \beta\,.$$

Proof. Denote $t = \mathrm{rank}(M)$. We prove the lower bound using the second equality of Theorem 3.3.4. Let $p = \mathrm{rank}(C)$, we have

$$\lambda_{p+1}(A^*A + B^*B - C^*C, M) = \max_{\substack{\dim(U) = t-p \\ U \perp \mathrm{null}(M)}} \min_{x\in U} \frac{x^*(A^*A + B^*B - C^*C)x}{x^*Mx}\,.$$

We prove the bound by showing that for one specific U, the ratio for any $x \in U$ is at least α. This implies that the minimum ratio in U is at least α, and that the maximum over all admissible subspaces U is also at least α.

Let $U = \mathrm{null}(C^*C) \cap \mathrm{rank}(M)$. Because M is Hermitian positive semidefinite, $\mathrm{null}(M) \perp \mathrm{rank}(M)$. This implies that $U \perp \mathrm{null}(M)$. Since $\mathrm{null}(M) \subseteq \mathrm{null}(C^*C)$, we have $\dim(U) = t - p$ (here we use $\mathrm{rank}(C^*C) \subseteq \mathrm{rank}(M)$). For every $x \in U$ we have $x^*(A^*A + B^*B -$

$C^*C)x = x^*(A^*A + B^*B)x \geq x^*A^*Ax$, so

$$\begin{aligned}\frac{x^*(A^*A + B^*B - C^*C)x}{x^*Mx} &\geq \frac{x^*A^*Ax}{x^*Mx} \\ &\geq \min_{x \in \operatorname{rank}(M)} \frac{x^*A^*Ax}{x^*Mx} \\ &= \lambda_1(A^*A, M) \\ &\geq \alpha\,.\end{aligned}$$

Therefore,

$$\min_{x \in U} \frac{x^*(A^*A + B^*B - C^*C)x}{x^*Mx} \geq \alpha\,,$$

so $\lambda_{p+1}(A^*A + B^*B - C^*C, M) \geq \alpha$. Since $p = \operatorname{rank}(C) \leq r$ we have shown that

$$\lambda_{r+1}(A^*A + B^*B - C^*C, M) \geq \lambda_{p+1}(A^*A + B^*B - C^*C, M) \geq \alpha\,.$$

For the upper bound we use a similar strategy, but with the first equality of Theorem 3.3.4. Let $l = \operatorname{rank}(B)$, we have

$$\lambda_{t-l}(A^*A + B^*B - C^*C, M) = \min_{\substack{\dim(V) = t-l \\ V \perp \operatorname{null}(M)}} \max_{x \in V} \frac{x^*(A^*A + B^*B - C^*C)x}{x^*Mx}\,.$$

Let $V = \operatorname{null}(B^*B) \cap \operatorname{rank}(M)$. Since M is Hermitian positive semidefinite $V \perp \operatorname{null}(M)$ and $\dim(V) = (n-l)-(n-t) = t-l$. For every $x \in V$ we have $x^*(A^*A+B^*B-C^*C)x = x^*(A^*A - C^*C)x \leq x^*A^*Ax$, so

$$\begin{aligned}\frac{x^*(A^*A + B^*B - C^*C)x}{x^*Mx} &\leq \frac{x^*A^*Ax}{x^*Mx} \\ &\leq \max_{x \in \operatorname{rank}(M)} \frac{x^*A^*Ax}{x^*Mx} \\ &= \lambda_t(A^*A, M) \\ &\leq \beta\,.\end{aligned}$$

Since

$$\max_{x \in V} \frac{x^*(A^*A + B^*B - C^*C)x}{x^*Mx} \leq \beta\,,$$

we have $\lambda_{t-l}(A^*A + B^*B - C^*C, M) \leq \beta$, and since $l = \operatorname{rank}(B) \leq k$ we have shown that

$$\lambda_{t-k}(A^*A + B^*B - C^*C, M) \leq \lambda_{t-l}(A^*A + B^*B - C^*C, M) \leq \beta\,.$$

□

We now give the analogous theorem when columns are modified.

Theorem 3.4.4. *Let $D \in \mathbb{C}^{m\times n}$ and let $E \in \mathbb{C}^{m\times k}$ and $F \in \mathbb{C}^{m\times k}$ for some $1 \leq k < n$. Let*

$$A = \begin{bmatrix} D & E \end{bmatrix} \in \mathbb{C}^{m\times(n+k)}$$

and let

$$\tilde{A} = \begin{bmatrix} D & F \end{bmatrix} \in \mathbb{C}^{m\times(n+k)}\,.$$

Let $M \in \mathbb{C}^{(n+k)\times(n+k)}$ *be an Hermitian positive semidefinite matrix, such that* $\operatorname{null}(M) \subseteq \operatorname{null}(A^*A)$ *and* $\operatorname{null}(M) \subseteq \operatorname{null}(\tilde{A}^*\tilde{A})$. *Suppose that*

$$\alpha \leq \lambda_1(A^*A, M) \leq \lambda_{\operatorname{rank}(M)}(A^*A, M) \leq \beta\,.$$

Then we have

$$\alpha \leq \lambda_{k+1}(\tilde{A}^*\tilde{A}, M) \leq \lambda_{\operatorname{rank}(M)-k}(\tilde{A}^*\tilde{A}, M) \leq \beta\,.$$

Proof. We denote $t = \operatorname{rank}(M)$ and $r = \operatorname{rank}(E^*D - F^*D) \leq k$ (because $E^*D - F^*D$ has k rows). We prove both sides by applying Theorem 3.3.4. We define the linear subspace of $\mathbb{C}^{n+k}$

$$U = \left\{ \begin{bmatrix} v \\ 0 \end{bmatrix} \,:\, v \in \mathbb{C}^n \text{ and } E^*Dv = F^*Dv \right\} \cap \operatorname{rank}(M)\,.$$

Clearly, U is a linear space and $U \perp \operatorname{null}(M)$. For any $\begin{bmatrix} v \\ 0 \end{bmatrix} \in \operatorname{null}(M)$, the vector $v \in \mathbb{C}^n$ satisfies $E^*Dv = F^*Dv = 0$, because $\operatorname{null}(M) \subseteq \operatorname{null}(A^*A)$ and $\operatorname{null}(M) \subseteq \operatorname{null}(\tilde{A}^*\tilde{A})$. This implies that set of v's for which $v \in \operatorname{null}(E^*D - F^*D)$ contains the set of v's for which $\begin{bmatrix} v \\ 0 \end{bmatrix} \in \operatorname{null}(M)$. This allows us to determine the dimension of U,

$$\begin{aligned} \dim(U) &= \dim \operatorname{null}(E^*D - F^*D) - \dim\left(\left\{ v \in \mathbb{C}^n \,:\, \begin{bmatrix} v \\ 0 \end{bmatrix} \in \operatorname{null}(M) \right\}\right) \\ &= (n-r) - (n-t) \\ &= t - r\,. \end{aligned}$$

It is easy to see that for every $x \in U$ we have $A^*Ax = \tilde{A}^*\tilde{A}x$, so

$$\begin{aligned} \frac{x^*\tilde{A}^*\tilde{A}x}{x^*Mx} &= \frac{x^*A^*Ax}{x^*Mx} \\ &\geq \min_{x \in \operatorname{rank}(M)} \frac{x^*A^*Ax}{x^*Mx} \\ &= \lambda_1(A^*A, M) \\ &\geq \alpha\,. \end{aligned}$$

Since, by the second equality of the Theorem 3.3.4,

$$\lambda_{r+1}(\tilde{A}^*\tilde{A}, M) = \max_{\substack{\dim U = t-r \\ U \perp \operatorname{null}(M)}} \min_{x \in U} \frac{x^*\tilde{A}^*\tilde{A}x}{x^*Mx}\,,$$

we conclude that $\lambda_{k+1}(\tilde{A}^*\tilde{A}, M) \geq \lambda_{r+1}(\tilde{A}^*\tilde{A}, M) \geq \alpha$. Similarly, for every $x \in U$,

$$\begin{aligned} \frac{x^*\tilde{A}^*\tilde{A}x}{x^*Mx} &= \frac{x^*A^*Ax}{x^*Mx} \\ &\leq \max_{x \in \operatorname{rank}(M)} \frac{x^*A^*Ax}{x^*Mx} \\ &= \lambda_t(A^*A, M) \\ &\leq \beta\,. \end{aligned}$$

Since, by the first equality of the Theorem 3.3.4,

$$\lambda_{t-r}(\tilde{A}^*\tilde{A}, M) = \min_{\substack{\dim U = t-r \\ U \perp \mathrm{null}(M)}} \max_{x \in U} \frac{x^*\tilde{A}^*\tilde{A}x}{x^*Mx},$$

we conclude that $\lambda_{t-k}(\tilde{A}^*\tilde{A}, M) \leq \lambda_{t-r}(\tilde{A}^*\tilde{A}, M) \leq \beta$. □

3.4.3. Using the Simple Spectrum of A^*A to Bound the Magnitude of Runaways. In some cases it is useful to know that runaway eigenvalues are either very small or very close to 1. For example we want to ensure that if we perturb an ill-conditioned A^*A to a well-conditioned A^*A+B^*B, the numerical rank of A^*A and of (A^*A, A^*A+B^*B) are the same, up to an appropriate relaxation of the rank threshold. We need the following lemma.

Lemma 3.4.5. *Suppose that $S \in \mathbb{C}^{n\times n}$ is an Hermitian matrix and that $T \in \mathbb{C}^{n\times n}$ is an Hermitian positive definite matrix. For all $1 \leq k \leq n$ we have*

$$\frac{\lambda_k(S)}{\lambda_n(T)} \leq \lambda_k(S,T) \leq \frac{\lambda_k(S)}{\lambda_1(T)}.$$

Proof. Let $u_1, \ldots, u_n$ be a set of orthonormal eigenvectors corresponding to $\lambda_1(S), \ldots, \lambda_n(S)$. Using the subspaces $U_k = \mathrm{span}\{u_1, \ldots, u_k\}$ and $V_k = \mathrm{sp}\{u_k, \ldots, u_n\}$ in the first and second inequality of Theorem 3.3.4 respectively gives the two bounds. □

We now state and prove the main results.

Theorem 3.4.6. *Let $A \in \mathbb{C}^{m\times n}$ and let $B \in \mathbb{C}^{k\times n}$ for some $1 \leq k < n$. Assume that $A^*A + B^*B$ is full rank. Denote $\alpha = ||A^*A||_2$. If there are d eigenvalues of A^*A that are smaller than or equal to $\epsilon\alpha$ for some $1 > \epsilon > 0$, then d generalized eigenvalues of $(A^*A, A^*A + B^*B)$ are smaller than or equal to $\epsilon\kappa(A^*A + B^*B)$.*

Proof. We denote $S = A^*A$ and $T = A^*A + B^*B$. We first note that $\lambda_n(T) \geq \lambda_n(S) \geq \alpha$. By the lemma,

$$\begin{aligned} \lambda_k(S,T) &\leq \frac{\lambda_k(S)}{\lambda_1(T)} = \frac{\lambda_k(S)\lambda_n(T)}{\lambda_n(T)\lambda_1(T)} \\ &= \frac{\lambda_k(S)}{\lambda_n(T)}\kappa(T) \\ &\leq \frac{\lambda_k(S)}{\alpha}\kappa(T). \end{aligned}$$

For any k such that $\lambda_k(S) \leq \epsilon\alpha$, we obtain the desired inequality. □

Theorem 3.4.7. *Let $A \in \mathbb{C}^{m\times n}$ and let $B \in \mathbb{C}^{k\times n}$ for some $1 \leq k < n$. Assume that $A^*A + B^*B$ is full rank. Denote $\alpha = ||A^*A||_2$ and suppose that $||B^*B||_2 \leq \gamma\alpha$. If there are d eigenvalues of A^*A that are larger than or equal to $\eta\alpha$ for some $1 > \eta > 0$ then d generalized eigenvalues of $(A^*A, A^*A + B^*B)$ are larger than or equal to $\eta/(1+\gamma)$.*

PROOF. We use the same notation as in the previous proof. We have $\lambda_n(T) \leq ||A^*A||_2 + ||B^*B||_2 \leq (1+\gamma)\alpha$. Therefore

$$\begin{aligned} \lambda_k(S,T) &\geq \frac{\lambda_k(S)}{\lambda_n(T)} \\ &\geq \frac{\lambda_k(S)}{(1+\gamma)\alpha}, \end{aligned}$$

which gives the desired bound for any k such that $\lambda_k(S) \geq \eta\alpha$. □

The theorems show that the numerical rank of the preconditioned system is the same as of the original system, up to an appropriate relaxation of the rank threshold. Suppose that after the dth eigenvalue there is a big gap. That is, there are d eigenvalues of A^*A are smaller than $\epsilon\alpha$, and the remaining $n-d$ are larger than $\eta\alpha$, where α is the largest eigenvalue of A^*A. The ratio between the largest eigenvalue and the dth smallest is at least $1/\epsilon$, and between the largest eigenvalue and the $(n-d)$th largest is at most $1/\eta$. Recall that 1 is the largest eigenvalue of (A^*A, A^*A+B^*B). Therefore, the ratio between the largest eigenvalue of the pencil and the d smallest is at least $\kappa^{-1}(A^*A+B^*B)/\epsilon$, and the ratio between the largest eigenvalue of (A^*A, A^*A+B^*B) and the $n-d$ largest eigenvalues is at most $(1+\gamma)/\eta$. Therefore if B^*B is not too large relative to A^*A, and A^*A+B^*B is well-conditioned, then the ratios are roughly maintained.

In Section 3.6 below we present an efficient algorithm that finds a B such that $||B^*B||_2 \leq m||A^*A||_2$ and $\kappa(A^*A+B^*B) \leq \tau^2$, where $\tau \geq n+1$ is a given threshold, and a slightly more expensive algorithm that only requires $\tau \geq \sqrt{2n}$ and guarantees $||B^*B||_2 \leq ||A^*A||_2$.

3.5. Applications To Least-Square Solvers

This section describes applications of the theory to the solution of linear least-squares problems. We show that we can often obtain useful algorithms by combining a sparse QR factorization of a modified matrix with a preconditioned iterative solver. We focus on improving the utility and efficiency of sparse QR factorizations, not on the more general problem of finding effective preconditioners.

In all the applications, we compute the R factor of a QR factorization of a modified matrix and use it as a preconditioner in LSQR. Our spectral theory in Section 3.4 shows that the preconditioned system has only a few runaway eigenvalues. We then can use [**161**, Theorem 2.3] to bound the number of iterations.

3.5.1. Dropping Dense Rows for Sparsity. The R factor of $A = QR$ is also the Cholesky factor of A^*A. Rows of A that are fairly dense cause A^*A to be fairly dense. Hence, R will be dense. In the extreme case, a completely dense row in A causes A^*A and R to be completely dense (unless there are exact cancellations, which are rare). This happens even if the other rows of A all have a single nonzero.

If A has few rows that are fairly dense, we recommend that they be dropped before the QR factorization starts. More precisely, these rows should be dropped even before the column ordering is computed. If we dropped k dense rows, we expect LSQR to converge in at most $k+1$ iterations (see subsection 3.2.1).

Heath [**120**] proposed a different method for handling dense rows (see also [**39**] and [**160**]), in which the dominant costs are the factorization of the first m rows of A (same as in our

approach), k triangular solves with R^*, and a dense QR factorization of an $(n+k)$-by-k matrix. In most cases (e.g., when R is denser than A), the asymptotic cost of the two methods is similar; there are also cases in which one method is cheaper than the other (in both directions).

3.5.2. Updating and Downdating. Updating a least-squares problem involves adding rows to the coefficient matrix A and to the right-hand-side b. Downdating involved dropping rows. Suppose that we factored the original coefficient matrix A, that updating added additional rows represented by a matrix B, and that downdating removed rows of A that are represented by a matrix C. The coefficient matrix of the normal equations of the updated/downdated problem is $A^*A + B^*B - C^*C$. As long as this coefficient matrix is full rank and the number of rows in B and C is small, Theorem 3.4.3 guarantees that the R factor of A is an effective preconditioner.

3.5.3. Adding Rows to Solve Numerically Rank-Deficient Problems. We propose two methods for solving numerically rank-deficient problems.

3.5.3.1. *Using an Iterative Method.* When A is rank deficient, there is an entire subspace of minimizers of $\|Ax - b\|_2$. When A is full rank but highly ill-conditioned, there is a single minimizer, but there are many x's that give almost the same residual norm. Of these minimizers or almost-minimizers, the user usually prefers a solution with a small norm.

The factorization $A = QR$ is not useful for solving rank-deficient and ill-conditioned least-squares problems. The factorization is backward stable, but the computed R is ill-conditioned. This usually causes the solver to produce a solution $x = R^{-1}Q^*b$ with a huge norm. This also happens if we use R as a preconditioner in LSQR: the iterations stop after one or two steps with a solution with a huge norm. Even after the first iteration the solution vector has a huge norm.

The singular-value decomposition (SVD) and rank-revealing QR factorizations can produce minimal-norm solutions, but they are difficult to compute. The SVD approach is not practical in the sparse case. The rank-revealing QR approach is practical ([**171, 37, 27, 60, 120**]), but sparse rank-revealing QR factorizations are complex and only a few implementations are available.

The approach that we propose here is to add rows to the coefficient matrix A to avoid ill-conditioning in R. That is, we dynamically add rows to A to avoid ill-conditioning in R. The factor R is no longer the R factor of A, but the R factor of a perturbed matrix $\left[\begin{smallmatrix} A \\ B \end{smallmatrix}\right]$, where B consists of the added rows. Section 3.6 outlines an algorithm for dynamically adding rows to A, so that the R factor of the perturbed matrix will not be ill-conditioned.

The well-conditioned R factor of the perturbed matrix is then used as a preconditioner for LSQR. The convergence threshold of LSQR allows the user to control a trade-off between the norm of the residual and the norm of the solution. Suppose that the user wishes to find a minimizer of $\min_x \|\bar{A}x - b\|_2$, where $\bar{A}$ has the same singular value decomposition as A except that the k smallest singular values of A are replaced by 0. When LSQR's convergence threshold is larger than $r = \sigma_{n-k}/\sigma_1$, it computes such a minimizer [**166**].

When the R factor of $\left[\begin{smallmatrix} A \\ B \end{smallmatrix}\right]$ is used as a preconditioner, correct truncation at σ_{n-k} depends on the preconditioned system preserving the singular gap above σ_{n-k}. This is why the results in subsection 3.4.3 are important: they guarantee this preservation.

In exact arithmetic, the number of rows in B bounds the number of iterations in LSQR. It may be smaller if the runaway eigenvalues are clustered.

3.5.3.2. *Using a Direct Method.* If the number of rows in B is *exactly* the same as the number of singular values we wish to truncate, and if A^*A+B^*B is well-conditioned, then a direct method can find an approximation of a small-norm minimizer. Let $A \in \mathbb{C}^{m\times n}$ and let $B \in \mathbb{C}^{k\times n}$. Let $\left[\begin{smallmatrix} A \\ B \end{smallmatrix}\right] = QR$ and $P = \left[\begin{smallmatrix} I_{m\times m} & 0_{m\times k} \end{smallmatrix}\right]$. We show that if the k smallest singular values of A are small enough then $\hat{x} = R^{-1}(PQ)^*b$ is close to a minimizer of $\min_x \|\bar{A}x - b\|_2$, as defined above.

We start with a simple lemma that forms the basis to our method.

Lemma 3.5.1. *Let $A \in \mathbb{C}^{m\times n}$ and let $B \in \mathbb{C}^{k\times n}$. Suppose that* $\operatorname{rank}(A) = n-k$ *and that $\left[\begin{smallmatrix} A \\ B \end{smallmatrix}\right]$ has full rank. Let $\left[\begin{smallmatrix} A \\ B \end{smallmatrix}\right] = QR$ be a QR factorization of $\left[\begin{smallmatrix} A \\ B \end{smallmatrix}\right]$. All the singular values of AR^{-1} are exactly 0 or 1.*

Proof. The singular values of AR^{-1} are the square root of the eigenvalues of $R^{-*}A^*AR^{-1}$. The eigenvalues $R^{-*}A^*AR^{-1}$ are exactly the eigenvalues of (A^*A, R^*R). It is easy to see that $R^*R = A^*A + B^*B$. If we apply Claim 2 of Theorem 3.4.3 we conclude that the multiplicity of the 0 eigenvalue of (A^*A, R^*R) is exactly $\dim \operatorname{null}(A) = n - \operatorname{rank}(A) = k$, and the multiplicity of the 1 eigenvalue of (A^*A, R^*R) is exactly $n - \operatorname{rank}(B) = n - k$. Therefore, n eigenvalues of (A^*A, R^*R), which are all the eigenvalues of (A^*A, R^*R), are either 0 or 1. □

We now show our claim for the case that A is exactly rank deficient by k, so $\bar{A} = A$. This is a simpler case than the case where the k smallest singular values are small but not necessarily zero. In this case the vector $\hat{x} = R^{-1}(PQ)^*b$ is an exact minimizers of $\min_x \|\bar{A}x - b\|_2$.

Lemma 3.5.2. *Let $A \in \mathbb{C}^{m\times n}$, $B \in \mathbb{C}^{k\times n}$ and $b \in \mathbb{C}^{m\times r}$. Suppose that* $\operatorname{rank}(A) = n-k$ *and $\left[\begin{smallmatrix} A \\ B \end{smallmatrix}\right]$ has full rank. Let $\left[\begin{smallmatrix} A \\ B \end{smallmatrix}\right] = QR$ be a QR factorization of $\left[\begin{smallmatrix} A \\ B \end{smallmatrix}\right]$. Let $P = \left[\begin{smallmatrix} I_{m\times m} & 0_{m\times k} \end{smallmatrix}\right]$. The vector $\hat{x} = R^{-1}(PQ)^*b$ is a minimizer of* $\min_x \|Ax - b\|_2$.

Proof. We show that $\hat{y} = R\hat{x} = (PQ)^*b$ is the minimum norm solution to $\min_y \|AR^{-1}y - b\|_2$. The minimum solution norm to $\min_y \|AR^{-1}y - b\|_2$ is

$$y_{\min} = \left(AR^{-1}\right)^+ b\,.$$

According to Lemma 3.5.1 the singular values of AR^{-1} are exactly 0 and 1. Therefore,

$$\left(AR^{-1}\right)^+ = \left(AR^{-1}\right)^*\,.$$

Notice that

$$AR^{-1} = P\left[\begin{smallmatrix} A \\ B \end{smallmatrix}\right] R^{-1} = PQRR^{-1} = PQ\,.$$

Therefore, $y_{\min} = (PQ)^*b = \hat{y}$. □

We now analyze the case where A is has k small but possibly nonzero singular values. In this case, $\hat{x} = R^{-1}(PQ)^*b$ is not necessarily a minimizer of $\min_x \|Ax - b\|_2$ and, more importantly, not even a minimizer of $\min_x \|\bar{A}x - b\|_2$. But if $\left[\begin{smallmatrix} \bar{A} \\ B \end{smallmatrix}\right] = \bar{Q}\bar{R}$, then the vector $\hat{z} = \bar{R}^{-1}(P\bar{Q})^*b$ is a minimizer of $\min_x \|\bar{A}x - b\|_2$. If the truncated singular values are small enough, then the pairs (Q, R) and $(\bar{Q}, \bar{R})$ will be closely related because they are QR factorizations of nearby matrices. Therefore, $\hat{x}$ and $\hat{z}$ should not be too far from each other. The next theorem shows that this is indeed the case.

Theorem 3.5.3. *Let $A \in \mathbb{C}^{m\times n}$, $B \in \mathbb{C}^{k\times n}$ and $b \in \mathbb{C}^m$. Let $\bar{A}$ be the matrix with the same singular value decomposition as A except that the k smallest singular values are truncated to 0. Denote*

$$C = \left[\begin{smallmatrix} A \\ B \end{smallmatrix}\right] \text{ and } D = \left[\begin{smallmatrix} \bar{A} \\ B \end{smallmatrix}\right]$$

Assume that C and D are both full rank. Let $C = QR$ be a QR factorization of C and $D = \bar{Q}\bar{R}$ be the QR factorization of D. Denote

$$\delta = \frac{\sigma_{n-k+1}(A)}{\sigma_{\min}(C)}$$

where $\sigma_{n-k}(A)$ is the kth smallest singular value of A and $\sigma_{\min}(C)$ is the smallest singular value of C. Let $P = \begin{bmatrix} I_{m\times m} & 0_{m\times k} \end{bmatrix}$. Define the solutions

$$\hat{x} = R^{-1}(PQ)^* b$$

and

$$\hat{z} = \bar{R}^{-1}(P\bar{Q})^* b\,.$$

Then, provided that $\delta < 1$,

$$\frac{\|\hat{x} - \hat{z}\|_2}{\|\hat{x}\|_2} \leq \frac{\delta}{1-\delta}\left(2 + (\kappa(R)+1)\frac{\|r\|_2}{\|R\|_2\|\hat{z}\|_2}\right)$$

where

$$r = b - A\hat{x}\,.$$

Before we prove the theorem, we explain what it means. The algorithm computes $\hat{x}$ and can therefore compute $r = b - A\hat{x}$. The theorem states that if δ is small (which happens when C is well conditioned and A has k tiny singular values) and R is not ill conditioned and not too large, and the norm of r is not too large, then $\hat{x}$ is a good approximation of the minimizer $\hat{z}$ that we seek. The quantity that is hard to estimate in practice is δ, which depends on the small singular values of A. Therefore, the method is useful mainly when we know a-priori the number of small singular values of A.

PROOF. Notice that $\hat{x}$ is the solution of $\min_x \|Cx - P^*b\|_2$ and that $\hat{z}$ is the solution of $\min_x \|Dx - P^*b\|_2$. Furthermore, we can write $D = C + \Delta C$ where $\|\Delta C\|_2 \leq \sigma_{n-k+1}(A)$. If we define $\epsilon = \sigma_{n-k+1}(A)/\|C\|_2$ then $\kappa(C)\epsilon = \delta < 1$ and $\|\Delta C\|_2 \leq \epsilon\|C\|_2$. We can apply a variant of result from Wedin [**222**] (see [**125**, Theorem 2.1] for the specific version that we use) and conclude that

$$\frac{\|\hat{x} - \hat{z}\|_2}{\|\hat{x}\|_2} \leq \frac{\delta}{1-\delta}\left(2 + (\kappa(R)+1)\frac{\|r\|_2}{\|R\|_2\|\hat{z}\|_2}\right).$$

□

3.5.4. Solving What-If Scenarios. The theory presented in this chapter allows us to efficiently solve what-if scenarios of the following type. We are given a least squares problem $\min \|Ax - b\|_2$. We already computed the minimizer using the R factor of A or using some preconditioners. Now we want to know how the solution would change if we fix some of its entries, without loss of generality $x_{n-k+1} = c_{n-k+1}, \ldots, x_n = c_n$, where the c_i's are some constants. We denote $A = \begin{bmatrix} D & E \end{bmatrix}$, where E consists of k columns. To solve the what-if scenario, we need to solve $\min \|Dx_{1:n-k} - (b - Ec)\|_2$. We solve instead $\min \|\tilde{A}x - (b - Ec)\|_2$ where $\tilde{A} = \begin{bmatrix} D & 0 \end{bmatrix}$, a matrix that we obtain from A by replacing

the last k columns by zeros. Clearly, the last k entries of x do not influence the norm of the residual in this system, so we can ignore them. By Theorem 3.4.2, for small k the factor or the preconditioner of A is effective for this least-squares system as well.

3.6. An Algorithm for Perturbing to Improve the Conditioning

In this section we show an algorithm that perturbs a given input matrix A to improve its conditioning. The algorithm only adds rows, which all have a single nonzero. The algorithm finds the perturbation during and after a standard Householder QR factorization (the technique applies to any column-oriented QR algorithm). Therefore, it can be easily integrated into a sparse QR factorization code; unlike rank-revealing QR algorithms, our algorithm does not exchange columns.

The goal of the algorithm is to build an R whose condition number is below a given threshold τ, with as few modifications as possible. More specifically, the goal is to find a $B \in \mathbb{C}^{k \times n}$ and upper triangular $R \in \mathbb{C}^{n \times n}$ such that

(1) $A^*A + B^*B = R^*R$,
(2) The Cholesky factors of A^*A and $A^*A + B^*B$ are structurally the same (except for accidental cancellations),
(3) $\kappa(R) \leq \tau$, and
(4) k is small.

We ensure that the first goal is met as follows. If, during the factorization of column j, the algorithm finds that it needs to add a row to B, it adds a row with zeros in columns 1 to $j-1$. This ensures that the first $j-1$ columns computed so far are also the factor of the newly-perturbed matrix. (In fact, it always adds a row with a nonzero only in column j.)

By restricting the number of non-zeros in each row of B to one, we automatically achieve the second goal, since B^*B is diagonal.

The algorithm works in two stages. In the first stage, the matrix is perturbed during the Householder QR factorization. In step j, we factor column j, and then run a condition-number estimator to detect ill-conditioning in the leading j-by-j block of R. If this block is ill-conditioned, we add a row to B, which causes only $R_{j,j}$ to change. A trivial condition estimation technique is to estimate the large singular value of A using its one or infinity norm, and then to estimate the smallest singular value using the smallest diagonal element in R. This method, however, is not always reliable. There are incremental condition estimators for triangular matrices that are efficient and more reliable [**36, 38, 35, 91**].

Let $c_A = ||A||_1$ be an estimation of the norm of A. Other norms can be used, and will modify some of the values below. All the rows of B will be completely zero except a single non-zero, which we set to $\pm c_A$. Each row of B has a different nonzero column. It follows that B^*B is diagonal with $||B||_2 = c_A$. Therefore,

$$\begin{aligned} ||R||_2 &= \sqrt{||R^*R||_2} = \sqrt{||A^*A + B^*B||_2} \\ &\leq \sqrt{||A^*A||_2 + ||B^*B||_2} \\ &\leq \sqrt{nc_A^2 + c_A^2} \\ &\leq c_A\sqrt{n+1}\,. \end{aligned}$$

Therefore, we add a row to B whenever the incremental condition estimator suspects that $||R^{-1}||_2 > \tau/c_A\sqrt{n+1}$. If we estimate $c_A = ||A||_2$ directly (using power iteration), we only need to ensure that $||R^{-1}||_2 > \tau/c_A\sqrt{2}$, so we can use fewer perturbations.

Condition estimators can fail to detect ill-conditioning. For example, if we estimate $\|R^{-1}\|_2 \approx 1/\min_j R_{j,j}$, it will not perturb the following matrix at all. Let

$$T_n(c) = \operatorname{diag}(1, s, \ldots, s^{n-1}) \begin{bmatrix} 1 & -c & -c & \cdots & -c \\ 0 & 1 & -c & \cdots & -c \\ & \ddots & & \vdots & \vdots \\ \vdots & & & 1 & -c \\ 0 & & \ldots & 0 & 1 \end{bmatrix}$$

with $c^2 + s^2 = 1$ with $c, s > 0$. For $n = 100$ the smallest diagonal value of $T_n(0.2)$ is 0.13, but its smallest singular value is $O(10^{-8})$ [**109**].

Better condition estimators will not fail on this example, but they may fail on others. It is relatively easy to safeguard our algorithm against failures of the estimator. A few inverse iterations on R^*R will reliably estimate the smallest singular value. Inverse iteration is cheap because R is triangular. If we find that R is still ill-conditioned, we add more rows to B and rotate them into R using Givens rotations. The resulting factorization remains backwards stable.

To find a perturbation that will reduce the number of tiny singular values, we find an approximation of the smallest singular value and a corresponding right singular vector of R. Suppose that σ and v are such a pair, with $||v||_2 = 1$ and $||Rv||_2 = \sigma$. Let i be the index of the largest absolute value in v. Since $||v||_2 = 1$ we must have $|v_i| \geq 1/\sqrt{n}$. We add to B a row b^*,

$$b_j = \begin{cases} c_A & j = i \\ 0 & j \neq i \end{cases}$$

We now have

$$\begin{aligned} \left\| \begin{bmatrix} R \\ b^* \end{bmatrix} v \right\|_2 &\geq \|b^* v\|_2 \\ &= |b^* v| \\ &\geq c_A v_i \\ &\geq c_A/\sqrt{n}\,. \end{aligned}$$

If $\tau \geq n + 1$ then

$$\left\| \begin{bmatrix} R \\ b^* \end{bmatrix} v \right\|_2 \geq \frac{\sqrt{n+1}c_A}{\tau}\,,$$

and the number of singular values that are smaller than $\sqrt{n+1}c_A/\tau$ is reduced by one. We repeat the process until all singular values are large enough. If we estimate $c_A = ||A||_2$ directly (using power iteration), then the constraint on τ can be relaxed to $\tau > \sqrt{2n}$.

The combination of a less-than-perfect condition estimation with the kinds of perturbations that we use during the factorization (rows with a single nonzero) can potentially lead to a cascade of unnecessary perturbations. Suppose that the jth column of the matrix is dependent (or almost dependent) on the first $j-1$ columns, but that the condition estimator missed this and estimated that the leading j-by-j block of R is well conditioned.

Suppose further that after the factorization of column $j+1$, the condition estimator finds that the leading $(j+1)$-by-$(j+1)$ block of R is ill conditioned (it is). Our algorithm will perturb column $j+1$, which does not improve the conditioning of R. This can keep on going. From now on, R remains ill conditioned, so whenever the condition estimator finds that it is, our algorithm will perturb another column. These perturbations do not improve the conditioning but they slow down the iterative solver. Situations like these are unlikely, but in principle, they are possible. Therefore, we invoke the condition estimator before and after each perturbation. If a perturbation does not significantly improve the conditioning, we refrain from further perturbations. We will fix the ill conditioning by perturbing R after it is computed (it may also be possible to use inverse iteration to produce a more reliable perturbation during the factorization rather than wait until it is complete).

3.7. Numerical Examples

In this section we give simple numerical examples for the applications described in Section 3.5. The goal is to illustrate the benefits of the tools developed in this chapter.

3.7.1. Dropping Dense Rows for Sparsity; Updating. Consider the matrix

$$A = \begin{bmatrix} \alpha_1 & & \\ & \ddots & \\ & & \alpha_n \\ \beta_1 & \cdots & \beta_n \end{bmatrix},$$

for some (real or complex) $\alpha_1, \ldots, \alpha_n$ and $\beta_1, \ldots, \beta_n$. Suppose that we want to find the least squares solution to $\min_x ||Ax - b||_2$. The R factor of the QR factorization of A will be completely full, because A^*A is full. Therefore, solving the equation using the QR factorization will take $\Theta(n^3)$ time. If the equation is solved using LSQR then every iteration will cost $\Theta(n)$ operations, but the number of iterations done is proportional to $\kappa(A)$. The value of $\kappa(A)$ can be very large for certain values of $\alpha_1, \ldots, \alpha_n$ and $\beta_1, \ldots, \beta_n$.

Our analysis suggests a new method for solving the problem. We can remove the last row of A and form the preconditioner $R = \mathrm{diag}(\alpha_1, \ldots, \alpha_n)$. Our analysis shows that when solving the equation using LSQR preconditioned by R only 2 iterations will be done. Each iteration still cost $\Theta(n)$ operations, amounting to a linear time algorithm for solving the equation. In general, if there are $m \ll n$ full rows, an application of LSQR with a preconditioner that is only the diagonal will converge in m iterations, each of them with $\Theta(nm)$ operations. The total running time will be $\Theta(nm^2)$, while regular LSQR will complete in $\Theta(n^2 m)$ operations, and a QR based algorithm will complete in a $\Theta(n^3)$ operations. With Heath's method [**120**] the total running time will be $\Theta((n+m)m^2)$. We conducted experiments that validate this analysis.

The same analysis also applies to updating problems. If we are given A without its last row and compute the R factor of the input matrix, we can use this factor for efficiently solving least-squares problems involving an additional arbitrary constraint.

We conducted a simple experiment to test the actual convergence time using MATLAB 7.2 [**155**]. We generated $\alpha_1, \ldots, \alpha_n$ and $\beta_1, \ldots, \beta_n$ using RAND for various values of n. We then ran all the algorithms, setting the tolerance of LSQR to 10^{-6}, and measured

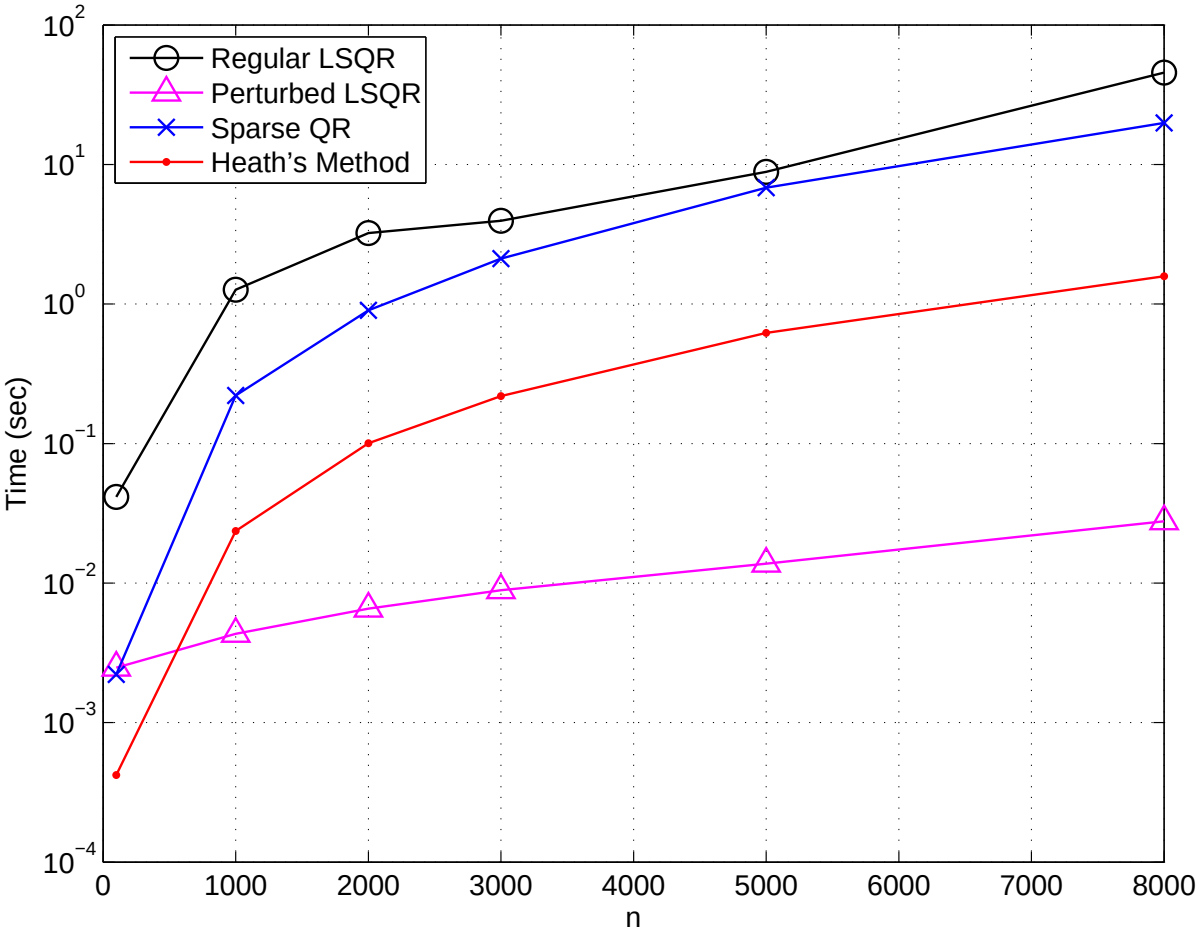


FIGURE 3.7.1. Running time when solving the least squares equation $\min_x ||Ax - b||_2$, when A consists of a diagonal and a dense row. We measured all methods for an increasing matrix size. The x-axis is the number of columns in the matrix. The diagonal and dense row values are chosen randomly.

the solution time using MATLAB's internal timer. The results are shown in Figure 3.7.1. It is easy to see that our method is superior to the other methods in this case.

3.7.2. Adding Rows to Solve Rank-Deficient Problems. Consider the matrix A and vector b generated by the following commands in MATLAB:

```
rand('state', 0);
m = ceil(n/4);
A0 = rand(n, m);
[U,Sigma1,V] = svd(A0, 0);
Sigma = diag(10 .^ [linspace(1, -4, m-1) -12]);
A1 = U*Sigma*V';
A = [A1 rand(n, m)];
b = rand(m, 1);
```

The codes builds a $n \times \frac{n}{2}$ matrix A, which is ill-conditioned ($\kappa \approx 10^{12}$ and norm around 1). We wish to solve the least squares problem $\min \|Ax-b\|_2$. The matrix is built so that column $n/4$ is close to a linear combination of the columns to its left. The first command resets the random number generator so that in each run will generate the same matrix and vector.

A QR factorization without pivoting generates a very small diagonal value (around 10^{-12}) in position $(\frac{n}{4}, \frac{n}{4})$ of R. Using the factorization to solve $\min_x \|Ax-b\|_2$ leads to a solution with norm around 10^{11}. In many cases, the desired solution is the minimizer in the subspace that is orthogonal to right singular vectors of A that correspond to singular values around 10^{-12} and smaller. We refer to such solution as a *truncated solution*. A QR factorization without pivoting is useless for finding the truncated solution or an approximation of it.

One way to compute a low-norm almost-minimizer of the truncated problem is to use a rank-revealing QR. If A is dense (as in our example), this is an effective solution. Rank-revealing QR factorization algorithms have also been developed for sparse matrices, but they are complex and sometimes expensive (since they cannot control sparsity as well as non-rank-revealing algorithms) [**60, 171, 37**].

In our example, running LSQR with a convergence threshold of $r = 10^{-10}$ (for $n = 100$) led to an acceptable solution (with norm around 10^3). With $r = 10^{-15}$, LSQR returned a solution with norm 10^{11}, clearly not a good truncated solution. Due to the ill-conditioning of A many iterations are required for LSQR to converge. Even with $r = 10^{-10}$, LSQR converged slowly, taking 423 iterations to converge.

We propose to use instead the algorithm described in Section 3.6 to generate an effective preconditioner that allows LSQR to solve the truncated problem. We generated two preconditioners using the two versions of our algorithm, one with $c_A = ||A||_1$ and the other with $c_A = ||A||_2$. We set the threshold τ to 10^{10}. In both cases a single row was added with a single nonzero in column 25. The need to add a row was detected, in both cases, using the incremental condition estimator during the initial QR factorization. With $c_A = ||A||_1$ the condition number of the factor was $\kappa(R) \approx 3.41 \times 10^5$, while with $c_A = ||A||_2$ the condition number was $\kappa(R) \approx 1.78 \times 10^5$. When using R as a preconditioner to LSQR with threshold 10^{-10} a single iteration was enough to converge in both cases. The norm of the minimizer x was of order 10^3 in both cases.

The different methods that produced solutions with norm around 10^3 produced different solution vectors x with slightly different norms (even the two preconditioned LSQR methods). To see why, let v be the singular vector that corresponds to the singular value 10^{-12}. LSQR uses the norm $||A^*(Ax-b)||_2$ as a stopping criterion. Adding ρv to a vector x changes $||A^*(Ax-b)||_2$ by at most $\rho \times 10^{-24}$, so even a large ρ rarely affects this stopping criterion. Therefore, different methods can return different solutions, say x and $x + \rho v$, possibly with $\rho \gg \|x\|$. Such solutions are very different from each other, but with norms and residual norms that both differ by at most $\rho \times 10^{-12}$. Both solutions are good, but they are different; this is a reflection of the ill conditioning of the problem.

3.8. Conclusions

This chapter presented theoretical analysis of certain preconditioned least-square solvers. The solvers use a preconditioner that is related to a low-rank perturbation of the coefficient matrix. The perturbation can be the result of an updating or downdating (following

the computation of a preconditioner or a factor of the original coefficient matrix), of dropping dense rows, or of an attempt to make the preconditioner well conditioned when the coefficient matrix is ill conditioned or rank deficient. We note that further research is required to determine how to drop rows effectively in sparse QR factorizations; we only gave here evidence that this idea can be effective, but we did not provide a row-dropping algorithm.

This chapter also proposed a specific method to perturb a QR factorization of an ill-conditioned or rank-deficient matrix.

Our theoretical analysis uses a novel approach: we count the number of generalized eigenvalues that move away from a cluster of eigenvalues (sometimes consisting only of the value 1) due to perturbations. This allows us to bound the number of iterations in iterative least-squares solvers like LSQR, which are implicit versions of Conjugate Gradients on the normal equations.

This approach complements the more common way of bounding iteration counts in optimal Krylov-subspace solvers, which is based on bounding the condition number of the preconditioned system.

We have also presented limited experimental results, which are meant to illustrate the use of the techniques rather to establish their effectiveness or efficiency. In the next chapter we present extensive numerical experimentation that demonstrate the effectiveness of row additions in solving rank deficient overdetermined systems.

CHAPTER 4

A Solver for Rank-deficient Overdetermined Least-squares Problems

4.1. Introduction

This chapter shows the effectiveness of using perturbed factorizations (QR or Cholesky) to solve (over)determined linear-least squares problems with a numerically rank deficient matrix. More specifically, it shows that the theoretical algorithms presented in Chapter 3 are indeed effective for sparse numerically rank-deficient problems.

Given a numerically rank-deficient matrix $A \in \mathbb{R}^{m \times n}$ $(m \geq n)$, and a vector $b \in \mathbb{R}^m$, we want to find a vector $x \in \mathbb{R}^n$ that minimizes the residual $\|Ax - b\|_2$. If A is numerically rank-deficient the minimizer x will have a huge norm, and some type of regularization is desired (see Section 2.7 in [**40**] and [**119**]). We assume that the desired regularization is given in the form of a numerical threshold $\tau > 1$, and that it is based on truncating the small singular values. Let $\sigma_1, \ldots, \sigma_n$ be the singular value of A and let r be index such that

$$\frac{\sigma_1}{\sigma_r} \leq \tau < \frac{\sigma_1}{\sigma_{r+1}}.$$

Let $\bar{A}$ have the same singular value decomposition as A except that the $n - r$ smallest singular values are truncated to 0. The goal is to find a minimizer x of $\|\bar{A}x - b\|_2$ with a small norm.

Figure 4.1.1 summarizes a few possible ways to try to tackle this problem. If no factorization of the matrix is found, and no preconditioner is built, then an iterative algorithm, such as LSQR [**166**], must be used. In this case the user must rely on the the iterative method's inherent regularization features. If LSQR is used, this usually yields good results from the numerical standpoint but can be very slow if the matrix is ill-conditioned. If the user is willing to build a QR factorization of the matrix, $A = QR$, he might try the direct formula $x = R^{-1}Q^T x$ (assuming that the matrix is not structurally rank-deficient). In this case the norm of x will be huge due to lack of regularization. Using R as a preconditioner to LSQR will give the same results since the iterative method will converge in one iteration and its regularization features will fail. Rank-revealing QR factorization based on column pivoting [**171, 37, 27, 60, 120**] and SVD decompositions enable the user to apply a regularization, but are considerably slower then pivot-less QR factorization, and are harder to parallelize.

Our algorithm uses a pivot-less QR and Cholesky factorization, therefore allowing faster running times, both sequential and parallel. We do so by forming a QR factorization of a perturbed matrix or a Cholesky factorization of the normal form of the perturbed matrix, and using it inside LSQR, relying on its inherent regularization features. The use of LSQR also ensures that as long as we use an appropriate threshold our method is backward stable [**61**], even if we use the Cholesky factorization of the normal equations.

	Direct	LSQR
No Factorization or Preconditioner	-	potentially slow low $\|Ax-b\|_2$ low $\|x\|_2$ low $\|A^Tr\|_2$
Perturbed-QR Factorization (new)	very fast low $\|Ax-b\|_2$ low $\|x\|_2$	fast low $\|Ax-b\|_2$ low $\|x\|_2$ low $\|A^Tr\|_2$
Perturbed-Cholesky Factorization (new)	-	very fast low $\|Ax-b\|_2$ low $\|x\|_2$ low $\|A^Tr\|_2$ may fail
QR Factorization	huge $\|x\|_2$	huge $\|x\|_2$
Rank-Revealing QR Factorization	potentially slow harder to parallelize	-
Truncated SVD	controllable but slow	-

FIGURE 4.1.1. Quick summary of algorithms for solving rank-deficient linear least-squares problems. This table shows the advantage of the algorithm used by our new solver.

Experiments on various test problems (reported in Section 4.4) show that our method finds well regularized solutions, and is faster than LSQR.

4.2. Algorithms

This section describes the algorithms used in our solvers. We only describe what the algorithm does, and do not give a theoretical analysis of it. We refer the reader to chapter 3 for the theoretical analysis of the algorithms.

The first step of our algorithms is to find a matrix $B \in \mathbb{R}^{k\times n}$ such that $\kappa(\left[\begin{smallmatrix}A\\B\end{smallmatrix}\right]) \leq \eta \leq \tau$, where η is an algorithmic parameter whose role we will discuss and investigate later. Our experiments indicate that the choice $\eta = \tau$ is usually good. The goal is to find matrix B with as few rows as possible, and with a single non-zero in each row. Our algorithm then finds a QR factorization of this matrix, $\left[\begin{smallmatrix}A\\B\end{smallmatrix}\right] = \tilde{Q}\tilde{R}$ (in a practical implementation there is no need to actually form $\tilde{Q}$).

We find B using an iterative algorithm that adds rows to B until $\kappa(\left[\begin{smallmatrix}A\\B\end{smallmatrix}\right]) \leq \eta$. After iteration i we have a matrix B_i with i rows, and hold a factorization of the matrix $\left[\begin{smallmatrix}A\\B_i\end{smallmatrix}\right] = \tilde{Q}_i\tilde{R}_i$. We initialize B_0 to an empty $0 \times n$ matrix, and find the factorization $A = \tilde{Q}_0\tilde{R}_0$ using a standard sparse QR algorithm. After we have found B_i we approximate the condition number of $\tilde{R}_i$ using power iterations. If the condition number is small enough we have found $B = B_i$. If the condition number is too large we find an approximate right

singular vector of the smallest singular value. We then add a row to B_i that holds a single non-zero valued $\|A\|_1$ in the index of largest magnitude in the singular vector, thereby forming B_{i+1}. In order to calculate $\tilde{Q}_{i+1}$ and $\tilde{R}_{i+1}$ we need to update an already existing factorization with a new row with single non-zero. We use the method of Irony et al [**198**] (which is a special case of Davis and Hager [**78**]) to do so.

If we inspect the above procedure we see that only $\Theta(\text{nnz}(R))$ operations are required for each perturbation. Nevertheless, this procedure can be very time consuming if many perturbations are done. It is useful to optimize the algorithm so that it can detect most perturbations during the initial factorization of the matrix, therefore saving work by doing them early. In a column-oriented factorization if a column has a tiny norm, this implies a small singular pair. We can do the perturbations on the spot, which is much cheaper than waiting till the end of the factorization. We do a perturbation if the diagonal value is smaller than $\|A\|_1\sqrt{n+1}/\eta$.

After we have found the factorization, we have two options. Either return

$$x = \tilde{R}^{-1}(P\tilde{Q})^T b \tag{4.2.1}$$

where $P = \begin{pmatrix} I_{m\times m} & 0_{m\times k}\end{pmatrix}$, or use $\tilde{R}$ as a preconditioner in an iterative solution of the problem using LSQR. If LSQR is used the convergence threshold must be set to reflect the desired regularization. See [**166**] for a discussion on the relationship between regularization and convergence thresholds in LSQR. In our implementation we used MATLAB's [**209**] built in LSQR function and set the single convergence parameter to τ^{-1}.

If we use the direct method we have an additional requirement that $\kappa([\begin{smallmatrix}A\\B\end{smallmatrix}]) \approx \sigma_1/\sigma_r$, which we achieve by setting $\eta = \tau$. In the LSQR method any value smaller than τ can be used for η. In this method the value of η entails a trade-off. If we use a smaller value for η our algorithm will be more accurate in regularizing the problem, but more row-perturbations will be required and more iterations will be done in the iterative phase. We explore this trade-off in our experiments (Section 4.4).

Instead of finding the R factor of $[\begin{smallmatrix}A\\B\end{smallmatrix}]$ we can find the Cholesky factorization of $A^TA + B^TB = \tilde{R}^T\tilde{R}$, which under exact arithmetic should be the same. The factor can than be used in the iterative variant of our solver. It can also be used in the direct formula $x = \tilde{R}^{-1}\tilde{R}^{-T}A^Tb$ based on the identity $P\tilde{Q} = A\tilde{R}^{-1}$. A solver based on the direct formula will be not be backward stable for the same reasons as the normal equations method is not backward stable (see Section 20.4 of [**126**]). In the iterative solver the factor is used only as a preconditioner, so the method will be as stable if an appropriate threshold is used. The perturbations in the Cholesky factorization are found using exactly the same way as they are found in the QR factorization.

The algorithms described above are applicable, without modifications except replacing the transpose operation with the adjoint operation, for complex A and b. We described the real-valued variant since in this work we tried the algorithms only on real-valued matrices.

4.3. Implementation

We have implemented and tested all the variants of the algorithms described in Section 4.2. To examine the QR based algorithms we have developed a completely new sparse QR factorization code. To experiment with the Cholesky based algorithms we modified an existing sparse Cholesky with our perturbation technique.

4.3.1. A New Sparse QR Factorization with Row Additions. We have implemented a new sparse multifrontal QR factorization code. The algorithm used by the code is based on the work presented in [**151, 11**] although some of the details are different. The new multifrontal QR factorization code is now a part of TAUCS[1]. The experiments reported in subsection 4.4.2 shows that our code performs well.

Optimally, the diagonal value of R should be examined, and if necessary a perturbation done, right after its corresponding column is formed. In this way no redundant operations will be done. In practice, a well implemented QR factorization code will try to employ level-3 Basic Linear Algebra Subroutines (BLAS) [**85**] operations, thereby delaying the examination of the diagonal values.

The BLAS operations are employed inside a factorization kernel. This kernel is applied on each supercolumn. It is responsible for forming its frontal matrix, and then partially factoring it using the BLAS. Only the pivotal columns are factored, and updates are applied to the non pivotal part. The end result is set of rows in R and a reduced block that is kept for the factorization of subsequent supercolumns.

Our code examines the diagonal values only after the partial factorization of the frontal matrix is complete. If we detect a diagonal value below the threshold, we need to do a perturbation in the corresponding column. We add the perturbation row to the partially factored frontal matrix, and use Givens rotations to refactor the perturbed frontal matrix, thereby updating the newly found rows in R and the reduced block. Givens rotations are restricted only to the rows and columns in the current frontal matrix.

4.3.2. Modifications to CHOLMOD. As explained in Section 4.2 we can use the Cholesky factorization of $A^TA + B^TB = \tilde{R}^T\tilde{R}$ instead of the QR factorization of $\left[\begin{smallmatrix} A \\ B \end{smallmatrix}\right]$. To do so we need a Cholesky factorization code that can find perturbations. We chose to implement the Cholesky-variant of our solver by modifying an existing factorization code. In particular, we chose to modify Tim Davis's CHOLMOD package [**66, 77**].

To find perturbations during the factorization we modified the dense kernel, which is responsible for factoring the frontal matrices. The kernel first assembles the frontal matrix. Next, the pivotal part of the frontal matrix is factored using a BLAS operation. We need to handle two cases: the initial factorization may fail due to singularity and/or numerical indefiniteness, or the result might have a small diagonal value. In both cases we do a perturbation on the unfactored frontal matrix and restart the factorization of the pivotal part. The factorization of the pivotal part is cheap, so the overhead is small. The rest of the dense kernel is left unmodified.

4.4. Experimental Results

This section presents experimental results that explore the effectiveness of our solver.

4.4.1. Setup. Our solvers is currently a hybrid between MATLAB 7.2 [**155**] code and C-code. The most important part of the computation, adding rows and finding the factorization, is implemented in C, either as part of the TAUCS library (for QR factorizations) or as modifications to CHOLMOD (for Cholesky factorizations). We use MATLAB for running LSQR. Another use for MATLAB is as a scripting tool to run the experiments, which calls

[1]Available from `http://www.tau.ac.il/~stoledo/taucs/`.

TAUCS or CHOLMOD using the CMEX interface. While some computation (as opposed to scripting) is done by MATLAB (running of LSQR) the time spent on those computations was a very small portion of the overall time spent by the algorithm (only a few iterations were needed). Therefore, the running time we report indicate well the running time of a pure C implementation.

Test matrices were obtained from the University of Florida sparse matrix collection [**72**]. For the baseline experiments, where we compared the performance of the factorization code alone, we used all matrices which are not square and that can be factorized in memory using a QR factorization. For experiments involving regularization of overdetermined rank deficient systems we focused on three interesting matrices: LPI_GRAN (matrix 717), LANDMARK (matrix 903) and GRAPHICS (matrix 981). Those matrices are rank deficient and large enough to be interesting.

Running times were measured on a 3.0 GHz Intel Pentium 4 computer with 2 GB of main memory, running Linux 2.6. This computer has 2 processors, but our solver only uses one. We used a 32-bit version of MATLAB 7.2. The measured running times are wall-clock times that were measured using the FTIME Linux system call. We use LAPACK [**12**] and ATLAS [**223**] for BLAS operations.

4.4.2. Baseline Experiments. The goal of our first set of experiments was to test the performance of our new baseline QR code. We ran our factorization code without doing any perturbations, and tested that the code performance was good enough. We compared our QR code to other available implementations of QR factorizations: MATLAB's built-in implementation (which is based on the method of George and Heath [**104**]) and SQR2 [**157, 3**]. We also compared the performance of our code to CHOLMOD's implementation of factorizing A^TA.

Figure 4.4.1 examines performance of our new code shows that it out performs CHOLMOD and SQR2 . The left graph shows performance on large (more than $10,000$ non-zeros) matrices with more rows than columns, which are the matrices of interest for this chapter. For completeness, the right graph of Figure 4.4.1 examines performance on matrices with more columns than rows.

We see that our solver is always either comparable or faster than MATLAB and SQR2 on underdetermined systems, although it usually does not find the sparsest factor. CHOLMOD is much faster than our code. It appears that QR factorization of A cannot compete, in terms of speed, with Cholseky factorization of A^TA. The missing points in the CHOLMOD plot are cases where the matrix was rank deficient and the Cholesky factorization failed. The missing points in the SQR2 plot are cases where SQR2 failed. For underdetermined systems we could only compare to CHOLMOD (SQR2 failed on all these matrices). Our code is nearly always faster than MATLAB by large factors.

4.4.3. Experiments on Structurally and Numerically Rank Deficient Matrices. The next phase was to test the solver on rank deficient matrices. We compare six solvers:

(1) Direct method: using QR factorization with empty B (no row additions).
(2) LSQR without any preconditionning. We run a maximum of 1000 iterations.
(3) Perturbed-Direct: build QR factorization with non empty B (adding rows), solve using equation 4.2.1.

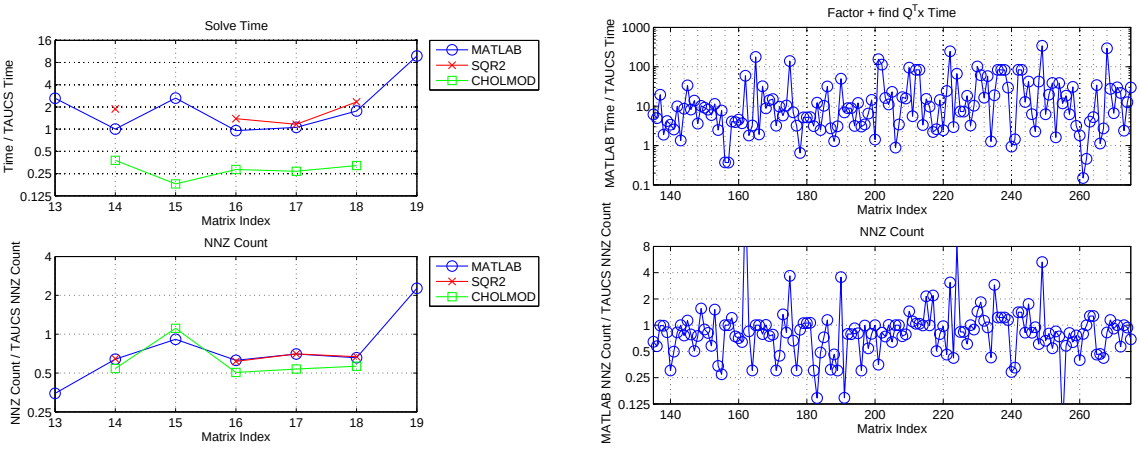


FIGURE 4.4.1. Solve time and number of non-zeros in R for various codes for finding the QR factorization of A or the Cholesky factorization of A^TA. The left graph is for matrices with more rows than columns (overdetermined systems), and the right graph is for matrices with columns than rows (underdetermined systems). Matrices are sorted according to the number of non-zeros. Shown are only matrices with more than 1000 non-zeros.

(4) Perturbed-LSQR (QR): build QR factorization with non empty B (adding rows), use R as preconditioner in LSQR.
(5) Perturbed-LSQR (CHOL): build Cholesky factorization of $A^TA + B^TB$, us as preconditioner in LSQR.
(6) Truncated SVD (TSVD) **[40]**. This is not a practical algorithm for large sparse matrices like these, and we use it for evaluating the numerical quality results.

Test matrices were taken from Tim Davis's collection. We found three interesting rank deficient matrices: LANDMARK, LPI_GRAN and GRAPHICS. See Table 4.4.2 for information on these matrices. We evaluate the quality of the solutions based on three parameters:

(1) The norm of the residual $\|r\|_2$. This is the actual term that is minimized and we expect this value to be close to the minimum.
(2) Orthogonality of the residual to rank(A): $\|A^Tr\|_2/(\|A\|_2 \cdot \|r\|_2)$. This is the term minimized by LSQR. It should be treated as an approximation to $\|\bar{A}^Tr\|_2/(\|\bar{A}\|_2 \cdot \|r\|_2)$ which should be zero for a true minimizer of $\|\bar{A}x - b\|_2$.
(3) The norm of the solution $\|x\|_2$: we are looking for almost-minimizers with small norm. The TSVD solution has the smallest possible norm for true minimizers.

Figures 4.4.3, 4.4.4 and 4.4.5 contain the result of the analysis for the three matrices. The tables show the results of each algorithm with respect to the various parameters. For LANDMARK and LPI_GRAN we also plot the singular values of the matrix. Matrix GRAPHICS was too big to allow us to compute the singular values.

The singular values of matrix LANDMARK drop quickly starting at around 10^{-5}, so setting $\tau = 10^6$ is reasonable, although it is actually hard to determine the numerical rank. The direct method fails completely. It finds a factorization but the R factor is rank deficient, so it cannot be used to find a solution. LSQR does not converge in 1000

Matrix	#rows	#cols	NNZ(A)	NNZ(A^TA)	NNZ(R)
PEREYRA/LANDMARK	71,952	2704	1,146,848	120,203	1,543,996
LPNETLIB/LPI_GRAN	2,658	2,525	20,111	60,459	57,636
SUMNER/GRAPHICS	29,493	11,822	117,954	82,642	124,909

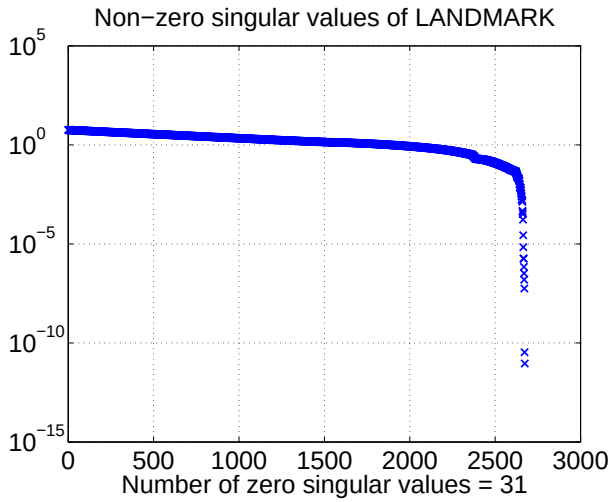
FIGURE 4.4.2. Rank deficient test matrices. NNZ(R) lists the number of non-zeros in the R factor of A when the columns are ordered using COLAMD (like our code) without supernodes. The number of non-zeros in the perturbed factorization is always the same as the number of non-zeros in the unperturbed factorization.

iterations, but it does make a nice progress. Given enough time it will eventually find a solution, albeit very slowly. All other solvers (Pert-Direct, and the Pert-LSQR solvers) find almost minimizers with small norm. Their norm is even smaller the TSVD solution, probably because they are only almost-minimizers (thus their value of $\|r\|_2$ is larger then TSVD's). Both Pert-Direct and Pert-LSQR (QR) find the same solution in about the same time. Pert-LSQR (CHOL) solution is of lesser quality, although it is the fastest.

The matrix LPI_GRAN is structurally rank deficient and ill conditioned ($\kappa > 10^{15}$). There is a big drop in singular values around 10^{-8}, so we set the threshold to $\tau = 10^{10}$. Again, the plain direct method and unpreconditioned LSQR failed. Perturbed direct was the fastest, but its results are less reliable. The norm of r is larger then all other method (which are close to the TSVD norm), but not too high. More importantly the residual is not orthogonal to A^T. Both version of Pert-LSQR find near-minimizers, but of different quality. The QR version finds a solution with smaller residual. The Cholesky version find a slightly higher residual, but the solution is more orthogonal to A^T and the norm x is smaller. The QR variant is a bit faster than the CHOL version.

The matrix GRAPHICS is too big for TSVD. It is also ill-conditioned, but not full rank. MATLAB estimated the condition number to be $\kappa_{\text{est}} \approx 1.59 \times 10^{10}$. We set the threshold to $\tau = 10^6$, although lack of singular value information prevents us from determining whether this is a correct choice. In practice we are more interested in finding near-minimizers with small norm then fixing the regularization parameter correctly. Under these settings the number of rows added with quite large (around 6000). The matrix is not rank-deficient, so the direct method did not fail on it. The direct method found a solution that is very close to being a minimizer, as can be observed from the near orthogonality of the residual to A^T, yet the norm of the solution is large relative to the solution found by other methods. Other methods found better solution, with norm around 10^{-6}. LSQR failed to declare convergence, as its residual was not orthogonal to A^T, but it did find a solution that is better than Pert-Direct in terms of norm of the solution and norm of the residual. The Pert-LSQR solvers computed the best results, but they are slower than other methods. This is because many rows are added, so a lot of LSQR iterations are done. The Cholesky version does more perturbations so it is slower.

4.4.4. Value of η. In previous experiments we used $\eta = \tau$, where τ is the regularization parameter and η is the bound on the preconditioner's condition number. We now investigate the tradeoff of selecting lower values for η.



	Time	$\|r\|_2$	$\frac{\|A^T r\|_2}{\|A\|_2\|r\|_2}$	$\|x\|_2$
Direct	25.0s	FAIL	FAIL	FAIL
Pert-Direct	26.4s	4.90×10^{-1}	6.12×10^{-7}	3.66×10^{0}
LSQR	197s	4.90×10^{-1}	1.74×10^{-4} FAIL	9.71×10^{-1}
Pert-LSQR,QR	25.7s	4.90×10^{-1}	6.12×10^{-7}	3.66×10^{0}
Pert-LSQR, CHOL	1.79s	4.90×10^{-1}	4.92×10^{-6}	9.87×10^{-1}
TSVD	N/A	4.90×10^{-1}	1.88×10^{-9}	4.27×10^{2}

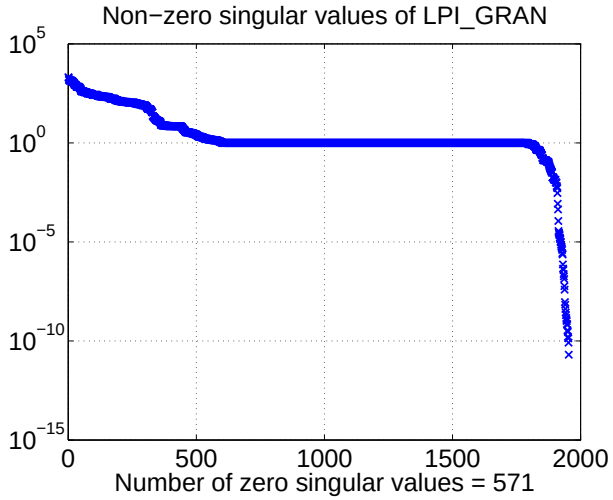
FIGURE 4.4.3. Detailed results for matrix LANDMARK with $\eta = \tau = 10^6$. The top graph shows the singular values of LANDMARK.

The value of η guides the factorization of A, so that the preconditioner will *not* be a too accurate factorization of A. LSQR has inherent regularization capabilities, provided that the iterates do not skip the desired solution. This is the reason an exact factorization is worthless with LSQR: the first iterate skips over all intern solutions right to the unregularized solution. The row additions are designed to slow the progress of LSQR so that the correct solution is encountered, but not too much. The maximum value is always τ, otherwise the first iterate will skip the desired solution. This is why we require $\eta \leq \tau$.

From the above discussion the tradeoff for lower values of η is clear. Lower values of η will allow LSQR to progress more slowly, thereby finding a better regularized solution. The tradeoff is a slower solver: both more rows will be added (QR factorization will take more time), and more LSQR iteration will be needed. This tradeoff is evident in Figure 4.4.6: when we set $\eta = \tau \times 10^{-2}$ we find a solution that is better regularized ($\|x\|_2$ is lower while $\|r\|_2$ stays roughly the same), but more iterations are done. In one case (matrix GRAPHICS with $\tau = 10^6$ and $\eta = 10^4$) the solver failed to converge (did more than 1000 LSQR iterations).

4.5. Conclusions

We have shown that a perturbed QR or Cholesky factorization can be an effective part of a linear least-squares solver. Most of the required perturbations are discovered



	Time	$\|r\|_2$	$\frac{\|A^T r\|_2}{\|A\|_2 \|r\|_2}$	$\|x\|_2$
Direct	0.34s	FAIL	FAIL	FAIL
Pert-Direct	0.68s	4.59×10^{-1}	1.08×10^{-2}	1.86×10^{4}
LSQR	3.95s	5.04×10^{-1}	1.45×10^{-3} FAIL	7.05×10^{-1}
Pert-LSQR, QR	1.34s	4.48×10^{-1}	1.73×10^{-4}	4.12×10^{6}
Pert-LSQR, CHOL	1.43s	4.50×10^{-1}	5.16×10^{-8}	5.63×10^{4}
TSVD	N/A	4.48×10^{-1}	1.77×10^{-8}	2.77×10^{4}

FIGURE 4.4.4. Detailed results for matrix LPI_GRAN with $\eta = \tau = 10^{10}$. The top graph shows the singular values of LPI_GRAN.

	Time	$\|r\|_2$	$\frac{\|A^T r\|_2}{\|A\|_2 \|r\|_2}$	$\|x\|_2$
Direct	0.61s	3.85×10^{-1}	4.77×10^{-12}	2.00×10^{-2}
Pert-Direct	1.16s	8.59×10^{-1}	1.98×10^{-5}	1.68×10^{-6}
LSQR	2.17s	8.46×10^{-1}	3.52×10^{-4} FAIL	1.60×10^{-6}
Pert-LSQR, QR	4.22s	7.95×10^{-1}	4.01×10^{-6}	7.36×10^{-5}
Pert-LSQR, CHOL	7.27s	7.95×10^{-1}	5.05×10^{-6}	7.16×10^{-5}

FIGURE 4.4.5. Detailed results for matrix GRAPHICS with $\eta = \tau = 10^{6}$.

during the factorization, when they can be applied cheaply. The perturbations remove the need to pivot, even when the matrix is rank deficient. The lack of pivoting simplifies the sparse factorizations, makes them relatively easy to parallelize [**146**], and eliminates a potential performance penalty. Using the R factor as a preconditioner for LSQR allows us to use Cholesky factorization without sacrificing numerical accuracy.

		$\eta = \tau$			$\eta = \tau \times 10^{-2}$		
Matrix	τ	#iterations	$\|x\|_2$	$\|r\|_2$	#iterations	$\|x\|_2$	$\|r\|_2$
LANDMARK	10^{10}	1	9.7×10^{-3}	4.90×10^{-1}	5	1.7×10^{-3}	4.90×10^{-1}
LPI_GRAN	10^{8}	6	8.3×10^{4}	4.54×10^{-1}	66	2.8×10^{2}	4.56×10^{-1}
GRAPHICS	10^{6}	45	7.36×10^{-5}	7.95×10^{-1}	did not converge	6.8×10^{-6}	8.3×10^{-1}

FIGURE 4.4.6. Comparison of two different values for η for the same matrix and τ combination.

CHAPTER 5

Application: ℓ_1-sparse Reconstruction of Sharp Point Set Surfaces

5.1. Introduction

This chapter[1] shows a real-life computer-graphics application that uses a perturbed QR factorization. The application requires a customized linear solver that uses perturbed factorizations suggested in the previous chapters.

The specific application is denoising of point clouds obtained from 3D laser scanners. Scanning devices have turned in the course of the last few years into commercial off-the-shelf tools. Current scanners are capable of producing large amounts of raw, dense point sets. One of today's principal challenges is the development of robust point processing and reconstruction techniques that deal with the inherent noise of the acquired data set.

Early point set surface methods [**7, 168, 8, 139**] assume the underlying surface is smooth everywhere. Hence, robustly reconstructing sharp features in presence of noise is more challenging. To account for sharp features and discontinuities, advanced methods rely on explicit representations of these characteristics [**2, 114**], use anisotropic smoothing [**98, 134**], robust statistics [**97, 163**] or feature aware methods [**147**]. These methods are typically fast, but they employ their operators locally and do not seek an objective function with a global optimum.

In this chapter, we introduce a technique based on a global approach that utilizes sparsity. Our method is motivated by the emerging theories of sparse signal reconstruction and compressive sampling [**86, 55**]. A key idea here is that in many situations signals can be reconstructed from far fewer data measurements compared to the requirements imposed by the Nyquist sampling theory. In sparse signal reconstruction, instead of using an over-complete representation, the reconstruction contains the sparsest possible representation of the object. This technique can become effective in the context of surface reconstruction, since common objects can often be characterized in terms of a rather small number of features, even if they are geometrically complex.

Our method is inspired by ℓ_1 minimization in sparse signal reconstruction and feature selection. We use the ℓ_1-sparsity paradigm since it avoids one of the main pitfalls of methodologies such as least squares, namely smoothed out error. Indeed, the ℓ_2 norm tends to severely penalize outliers and propagate the residual in the objective function uniformly. Hence, the solution is typically of a very poor degree of sparsity. Since least

[1]The results in this chapter also appear in a paper submitted to the *ACM Transactions on Graphics*, co-authored by Andrei Sharf, Chen Greif, and Daniel Cohen-Or [**21**]. The paper has been accepted subject to a minor revision.
The paper is a computer graphics paper, but there is a numerical solver involved. I was involved both in the numerical aspects and in the computer graphics aspects of this project.

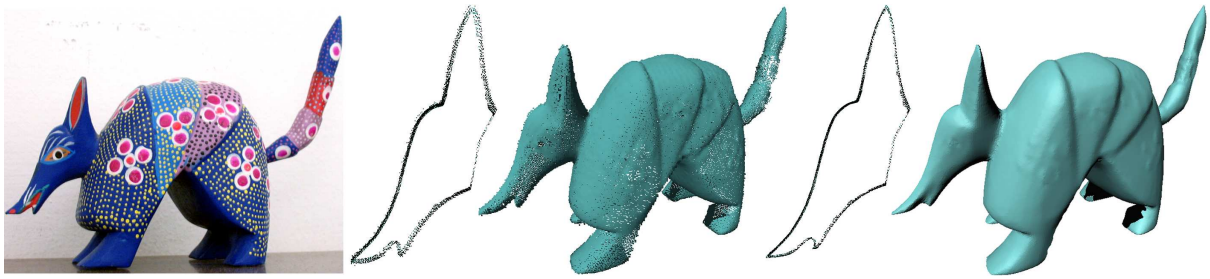


FIGURE 5.1.1. A demonstration of the effectiveness of our ℓ_1 sparsity-based approach on a scanned model. The Armadillo statue (left) is scanned generating a noisy point-cloud (middle). Using our method we are able to reconstruct it and recover its sharp features (right). Close-up view of a cross section of its head reveals the sharpness of the reconstructed surface.

squares methods by their nature handle smooth surfaces better than non-smooth ones, common techniques that use them must rely on working in a locally smooth fashion.

The last observation is particularly important for objects with sharp features. An ℓ_1 based method such as the one we introduce is not restricted by the above mentioned locality limitations that ℓ_2 entails. Since outliers are not excessively penalized, most of the "energy" of the residual in the objective function is expected to be concentrated near the sharp features. Thus, by minimizing the objective function in the ℓ_1 sense we encourage sparsity of non-smooth singularities in the solution. The ℓ_1 norm is not differentiable, and formulations associated with it are harder to solve, compared to ℓ_2-based formulations. Nevertheless, the objective function is convex and as long as convexity is preserved, it is well understood how to solve the problem in hand, and efficient solvers are available. Theory and applications based on ℓ_1-sparsity have been enjoying a huge boost in the last few years in the scientific community at large. Motivated by this success we seek to utilize ℓ_1-sparsity in computer graphics. In this work we use ℓ_1-sparsity to build a novel method for reconstructing point set surfaces with sharp features.

Our global reconstruction method incorporates also inequality constraints that ensure that the points are sufficiently close to the original input. We first solve for the point orientations, and then, based on the piecewise smooth normals, solve for the point positions. We show that we reconstruct point sets effectively, and compare our technique to state-of-the-art methods by running on synthetic models as well as real scanned models.

This chapter offers two main contributions. First is the formulation of a global ℓ_1-sparse optimization problem for 3D surfaces. We show that sparsity and ℓ_1-minimization are highly effective in surface reconstruction of real scanned objects. Second, we show that this problem can be solved efficiently by convex optimization techniques. We demonstrate that our method runs on large data sets within a reasonable time.

Most aspects of this project belong to the realm of computer graphics. Nevertheless, the numerical aspect of solving the linear equations involved in the interior point method was crucial and is highly relevant to this thesis. I was involved both in the numerical aspect and in the computer-graphics aspect of the project. Section 1.5 focused on the numerical aspects of the project. For completeness, this chapter follows the paper closely and its main foci is the computer graphics aspects.

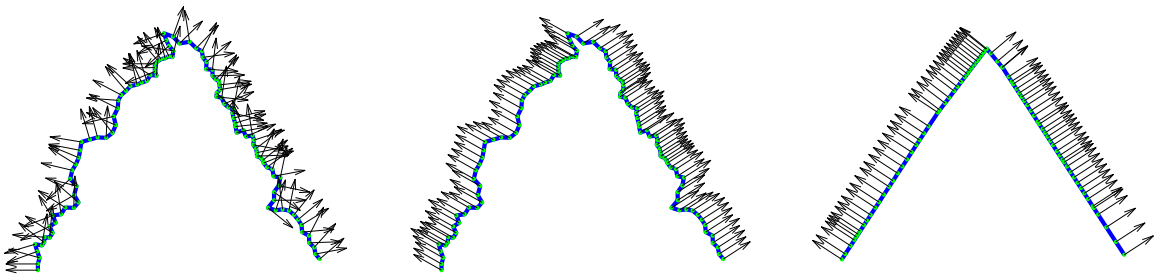


FIGURE 5.1.2. A 2D demonstration of our ℓ_1 sparse reconstruction method applied to a synthetic V-shaped model. Given a 2D noisy curve (left), the point orientations are solved first, using an interior point solver (middle). In the second step, the correctly computed orientations are used to recover consistent positions. The rightmost figure is the V-shape result of our algorithm.

5.2. Related Work

In this section we provide a review of relevant work separated into two parts. First, we focus on surface reconstruction methods in the context of sharp feature approximation. Then, we review methods for sparse signal reconstruction and total variation.

5.2.1. 3D Surface Reconstruction. Since the early 1990s, there has been a substantial amount of work in the domain of surface reconstruction from range scanned data.

Moving Least Squares (MLS) [**191**] is a classical and popular method for functional approximation. Levin [**144**] and Alexa et al. [**7**] have demonstrated its effectiveness and promise for representing point set surfaces. A number of papers have followed, improving and extending the MLS operator; see [**6, 8, 9, 190, 83, 139**]. At the core of these methods is an iterative projection that involves a local optimization to find the (local) reference plane and a bivariate polynomial fitting. Although state-of-the-art MLS based methods handle non-uniform data and noise, these methods are limited by the locality of the operator.

MLS-based techniques are ideally designed to reconstruct smooth surfaces. To reconstruct sharp features, various approaches have been proposed. Methods that rely on an explicit representation of sharp features [**176, 2, 114**] classify the input samples into piecewise linear components to model sharp features. A more challenging task is to automatically detect and reconstruct features present in noisy point clouds in a global framework.

Several feature aware filters have been developed for 3D mesh denoising and smoothing [**98, 134**]. Following image denoising approaches [**212**], they use an underlying Gaussian smoothing kernel that accounts for position and normal similarity in a continuous manner. Samples across a sharp feature can be seen as outliers, and robust statistics-based methods are presented in [**158**] to locally deal with those outliers and reconstruct sharp features. Similarly, Fleishman et al. [**97**] derive an iterative refitting algorithm that locally classifies the samples across discontinuities by applying a robust statistics framework. More recently, [**70, 163, 147, 148**] have presented robust methods for surface

reconstruction from point clouds, which handle sharp features and noise. Common to all these methods is their locality approach; both detection and reconstruction of sharp features are performed within a local context. This leads to the possible detection of local minima, where locally, high noise-to-signal ratio yields redundant features or in the other extreme over-smoothing. This phenomena is demonstrated in Figure 5.4.1 (middle row), which show the result of applying a bilateral filter (local method) on a shallow V-shaped noisy model. In contrast, we apply a global method that solves for the whole surface at once, avoiding such effects and generating a global optimal solution that preserves the sharp features (bottom row).

Note that some of the local methods discussed above, use some form of ℓ_1-norm minimization. Nevertheless, their usage is fundamentally different from our method. In their methods ℓ_1 minimization is applied as a robust statistic, based on the observation that data across discontinuity can be viewed as outliers. Thus, the ℓ_1 minimization is used to make the local filter more robust. Nevertheless, their local nature poses inherent limitations. In contrast, our method is based on the ℓ_1-sparsity paradigm (discussed in section 5.3) which naturally leads to a global method.

Somewhat similar to us, [**189**] have recently used a lower-than-ℓ_1 minimization scheme which they apply to dynamic surface reconstruction. The problem is formulated as an implicit incompressible flow volume minimization which accounts for volume boundaries using a reweighted $\ell_{0.8}$ norm. Both works have in common the observation that a metric below ℓ_2 better accounts for sharp features.

5.2.2. Sparse Signal Reconstruction. Sparse signal processing has had over the last few years a significant impact on many fields in applied science and engineering such as statistics, information theory, computer vision, and others. From a general viewpoint, sparsity and compressibility lead to dimensionality reduction and efficient modeling.

The general idea of ℓ_1 regularization for the purpose of sparse signal reconstruction or feature selection has been used in geophysics since the early 1970s; see, e.g., [**67**]. In signal processing, the idea of ℓ_1 regularization comes up in several contexts, including basis pursuit [**65**] and image decomposition [**92, 204**]. In these works it is shown that while the ℓ_2 norm yields a minimum length solution, it lacks properties of sparsity and the reconstructed signal error is spatially spread out. Although ℓ_0 is in fact the sparsest solution, in [**86**] it is shown that under bounded noise conditions, the recovered sparse representation using ℓ_1 is both correct and stable. Similar to us, [**213**] show that convex optimization techniques are useful for sparse signal reconstruction.

The ℓ_1-sparsity paradigm has been applied successfully to image denoising and deblurring using total variation (TV) methods [**182, 181, 59, 143**]. The underlying model for TV methods aims at exploiting the sparsity of the gradient of the image. The continuous variant yields a model of a nonlinear PDE, hence is less relevant in the context of the current work. The discrete variant, on the other hand, yields the following convex objective function:

$$TV(u) = \Sigma_{ij} \left\| D_{ij} u \right\|_2 ,$$

where D_{ij} is the discrete gradient operator at pixel (i, j) and u is a vector containing the gray-level pixel values. TV methods filter the image by minimizing $TV(u)$ under a certain constraint, for example one that keeps the output sufficiently close to the input:

$$\left\| u - u_0 \right\|_2 \leq \varepsilon.$$

TV is designed for images, and hence is not directly applicable to our problem. Our method is close in spirit to TV. Similar to their sparse gradient minimization, we formulate our piecewise smoothness reconstruction problem as a sparse minimization of orientation differences and position projections.

The analog of TV for surface denoising is total curvature (TC), which corresponds to minimizing the integral of the local curvature values [**207, 208, 94**]. TC methods preserve edges and sharp features, but by their nature, being represented by a nonlinear PDE, they require significant computational resources and an underlying parametrization. Given the nature of TC, it seems hard to develop a convex discrete variant for it.

Rather than working on curvatures, we examine the normal field of the surface. This allows us to develop a convex ℓ_1-based formulation. It should be stressed that convexity is of much importance, since it gives rise to a robust, efficient and stable solution procedure, in contrast to non-convex nonlinear problems such as the above mentioned ones.

5.3. ℓ_1 Sparsity Overview

Suppose we are given discrete samples of a continuous time signal $u(t)$. Signal reconstruction deals with the problem of reconstructing u from those samples as a linear combination of basis functions $\phi_i \in \mathbb{R}^n, i = 1, ..., m$, with $m > n$. The problem amounts to finding the coefficients α_i such that

$$u = \sum_{i=1}^{m} \alpha_i \phi_i.$$

The Nyquist sampling theorem states that the number of samples needed to reconstruct a signal without error is dictated by its bandwidth. Inside the bandwidth the signal can be dense, containing information in all frequencies. Nevertheless, it has been shown in a set of seminal papers [**55, 86**], that when the signal is sparse, i.e. only a small set of frequencies are present, the signal can be reconstructed from a number of samples much lower than required by Nyquist's theorem using ℓ_1 minimization.

Sparse signal reconstruction utilizes the fact that if the underlying signal indeed has a sparse representation, then its coefficients α can be cast as the solution of the optimization problem:

$$\min_{\alpha} \|\alpha\|_0 \quad \text{s.t.} \quad u(t_j) = \sum_{i=1}^{m} \alpha_i \phi_i(t_j),$$

where the samples are given at times t_j. The zero norm, $\|.\|_0$, represents the number of nonzero elements in a vector. Unfortunately, formulating the problem in terms of this norm requires a combinatorial solution time; the problem is highly non-convex. It is thus common practice to replace ℓ_0 by (the convex) ℓ_1 norm. It has been shown [**55, 86, 213**] that in some cases minimizing the ℓ_1 norm is equivalent to minimizing the ℓ_0 norm.

It is well established in the scientific community that in presence of discontinuities, ℓ_2 minimization tends to produce solutions that are smooth. To overcome this problem, the iterative reweighted ℓ_2 minimization (IRLS) method has been introduced [**128**] that achieves lower-than-ℓ_2 sparsity using a predesigned reweighting scheme. The weights essentially concentrate the error at discontinuities and enhance sparsity. Formulating an ℓ_1 minimization scheme instead, achieves sparsity in a much more direct way. Although theoretically ℓ_1 minimization can be approximated using IRLS, interior point methods

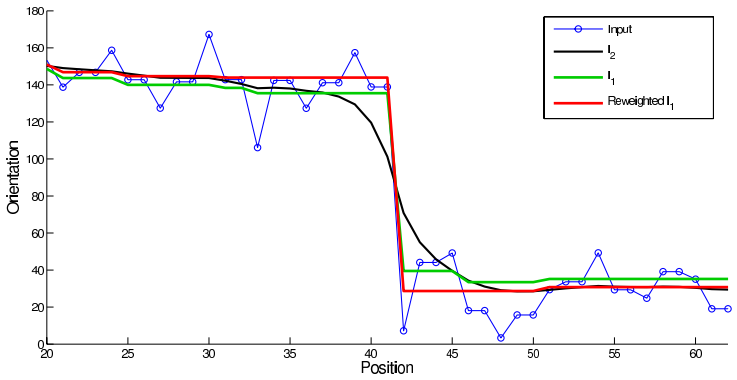


FIGURE 5.3.1. A comparison of applying ℓ_2 (black), ℓ_1 (green) and reweighted-ℓ_1 (red) to a set of noisy 2D orientations (blue). Sharpness in the reconstruction first appears in the ℓ_1 norm and is enhanced by reweighting.

(like the one we use) are usually faster and more reliable in terms of convergence and stability.

Sparse reconstruction is a broad methodology with many flavors. In some application areas, the basis functions are either known or predefined by the method. In these cases it is often possible to establish formal guarantees on the reconstruction. The most notable is compressive sampling which economically translates data into a compressed digital form subject to ℓ_1 minimization; for a survey see [**52**]. Other sparse reconstruction methods and most notably TV filters, do not assume a predefined overcomplete dictionary. Instead, these methods assume sparsity under some transformation, and set up an optimization scheme that filters the signal using the sparsity prior.

Our method is inspired by sparse ℓ_1 filtering methods like TV. As already mentioned, given that our scan data is sampled from a piecewise smooth shape, we seek to approximate it by a sparse set of smooth pieces that connect at the edges. We approximate smoothness in terms of pairwise orientation differences; next, positions are integrated from orientations by assuming local planarity. We provide the exact details of our formulation in the next section.

It is important to discuss the relationship between ℓ_1 minimization and sparse methods. Broadly speaking, there are two major research domains where ℓ_1 minimization has been extensively applied: robust statistics (i.e. as an M-estimator) and sparse recovery. Robust statistics use ℓ_1 due to its robustness to outliers. Sparse recovery methods use ℓ_1 as an approximation of ℓ_0. The problem is formulated such that the solution is a sparse vector, and ℓ_1 is used to find a sparse-as-possible approximation. Our method belongs to the latter class of methods.

5.4. Reconstruction Model

Since scanned information is generally noisy, we cannot assume that we have high quality point orientations, and rely on that in our solution procedure. Hence, similarly to [**149**], we decouple orientations and positions. We solve first for the orientations, and then use them to compute consistent positions. We formulate both problems in a similar

ℓ_1 nature, yielding a consistent solution. For a 2D demonstration of this process see Figure 5.1.2.

Our goal is to formulate a global surface approximation that is piecewise smooth and consists of a sparse set of singularities at sharp edges. We denote as $P(X, N)$ a point cloud defined by positions $x_i \in X \subseteq \mathbb{R}^3$ and corresponding orientations $n_i \in N \subseteq \mathbb{R}^3$. Thus, a point is defined by the pair $p_i = (x_i, n_i)$.

Our orientation reconstruction model is based on the same key observation made by [**208**]: smooth surfaces have smoothly varying normals, thus penalty functions should be defined on the surface normals. However their method assumes a local parametrization of the surface and involves solving second-order PDEs. These limitations have led us to adopt a simplified generic approach. Instead of using a quadratic form for curvature, we use pairwise normal differences as an estimator for shape smoothness. If two points p_i and p_j belong to the same smooth part, and the distance between them is small enough in local feature size, then $n_i \approx n_j$. Furthermore, we assume that there is a minimum crease angle at singularities between smooth parts. Hence, at crease angles where p_i and p_j belong to different smooth parts, the distance between n_i and n_j is above a small threshold τ. This leads to an observation that the reconstructed normals should be such that only a small (i.e., sparse) number of local normal differences are large.

We use computed orientations to define consistent positions by assuming that the surface can be approximated well by local planes. Given a pair of neighbor points (p_i, p_j), we examine $n_{ij} \cdot (x_i - x_j)$ (where n_{ij} is the average normal). Indeed, if both p_i and p_j belong to a smooth part of the surface then $n_{ij} \cdot (x_i - x_j) \approx 0$. At sharp features we expect $|n_{ij} \cdot (x_i - x_j)| > 0$.

Note that this last assumption is not necessarily true and in some point configurations $n_{ij} \cdot (x_i - x_j) \approx 0$ even at sharp features (e.g. p_i and p_j are equidistant from both sides of a sharp perpendicular edge). Nevertheless, such specific configurations do not occur much in practice and the use of large neighborhoods for each point compensates for this discrepancy.

In the following subsections we distinguish between an arbitrary point cloud $P(X, N)$, the input point cloud $P(X^{in}, N^{in})$, output point cloud $P(X^{out}, N^{out})$ and the intermediate point cloud after reconstructing the orientations $P(X^{in}, N^{out})$.

Re-weighted ℓ_1. We use in this work a re-weighted ℓ_1 scheme in order is to achieve *lower-than-*ℓ_1 sparsity. Such an aggressive sparsity is desired in cases where ℓ_1 is too smooth. Our motivation for using re-weighting is drawn from the nature of our problem. For scanned models it is common to have a high correlation of noise in the normals. This effect occurs since normals are commonly approximated using local PCA and it is enhanced since scanners tend to sample more points near sharp edges. While ℓ_1 minimization works well for uncorrelated random noise, it penalizes high values too strongly to break correlation-induced smoothness in the normals.

A higher degree of sparsity can be accomplished by driving the minimization norm below ℓ_1, using a norm ℓ_p , $0 < p < 1$. Unfortunately, such penalty functions are non-convex. Instead of applying them directly, we use ℓ_1 as the basic building block, and achieve the effect of lower-than-ℓ_1 sparsity using re-weighting. Here we rely on the theoretical observations made in [**56**], by which re-weighting within the ℓ_1 realm may enhance the sparsity and significantly improve the quality of the reconstruction. We can

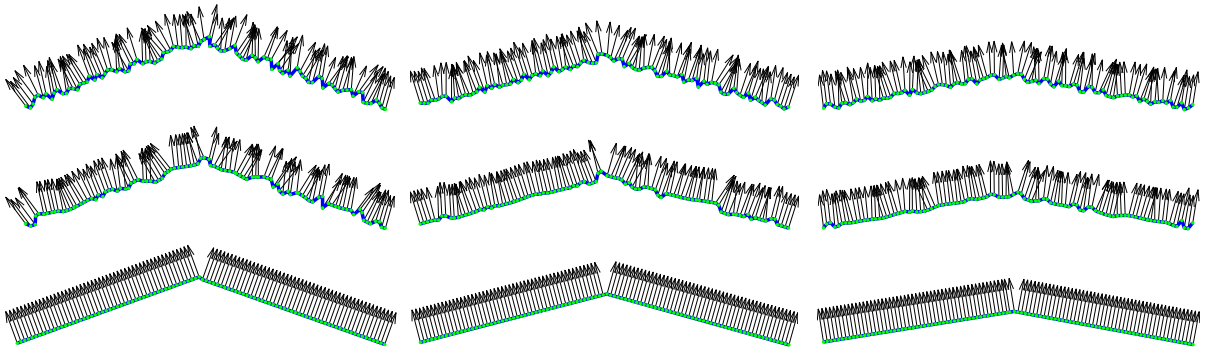


FIGURE 5.4.1. A comparison between ℓ_1 sparse reconstruction and bilateral filtering applied on a series of synthetic shallow 2D V-shaped models with different signal-to-noise ratios. Top, left-to-right are the noisy inputs with angles 140°, 150° and 160° and noise of 1.45%, 1.20% and 1.00% respectively. Middle is the result of applying a local bilateral filter. Bottom is our ℓ_1 sparse reconstruction.

see in Figure 5.3.1 that re-weighted ℓ_1 is sparser than ℓ_1 and hence achieves a better approximation in this example.

5.4.1. Orientation Reconstruction. Our orientation minimization consists of two terms: one is the global ℓ_1 minimization of orientation (normal) distances; the second amounts to constraining the solution to be reasonably close to the initial orientations. Following the discussion above, our minimization term is composed of the normal differences between adjacent points p_i and p_j, formulated as a pairwise penalty function c_{ij}^N:

$$c_{ij}^N(N) = \|n_i - n_j\|_2 \ .$$

For a piecewise smooth surface the set $\{c_{ij}^N \geq \tau\}$ is sparse for some small τ depending on the smoothness of the surface, so it makes sense to define a global weighted ℓ_1 penalty function C^N:

$$C^N(N, W, E) = \sum_{(p_i,p_j)\in E} w_{ij} c_{ij}^N(N)$$

where $W = \{w_{ij}\}$ is a set of weights whose role is to achieve lower-than-ℓ_1 sparsity as discussed above, and E is the adjacency set computed using k-nearest neighbors. We perform two ℓ_1 iterations. The first iteration is unweighted. Using the results of the first iteration, we compute weights and re-solve. The weights are set to

$$w_{ij} = e^{-(\theta_{ij}/\sigma_\theta)^4},$$

where θ_{ij} is the angle between the initial normals of p_i and p_j and σ_θ is a parameter (usually set to 10 degrees). We use an exponent of 4 instead of the usual 2 for a more aggressive sparsity enhancement.

Notice that $C^N(N, W, E)$ is in fact the ℓ_1-norm of the $|E|$-element vector $\begin{bmatrix} \cdots & w_{ij}c_{ij}^N(N) & \cdots \end{bmatrix}$.

We compute the new orientations as a minimization of the penalty function:

$$N^{out} = \arg\min_N C^N(N, W, E)$$

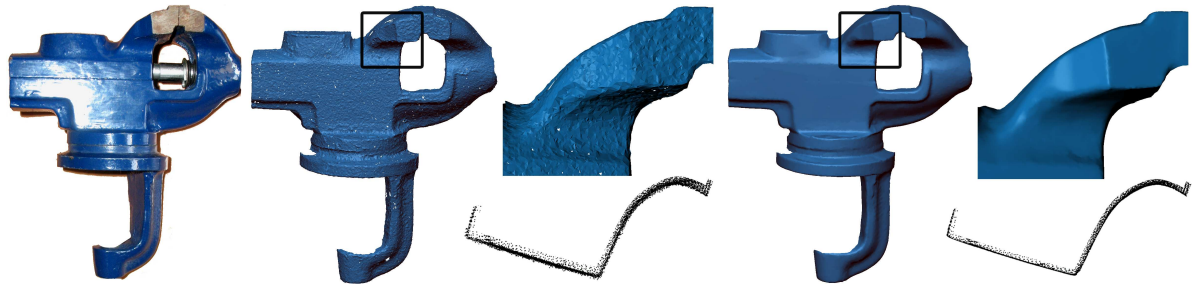


FIGURE 5.4.2. Reconstruction of a scanned iron vise. Left to right: the original vise photo; the noisy scan and a zoomed region with its corresponding cross section (bottom); our ℓ_1 reconstructed model; Rightmost is the zoomed result, demonstrating the sharp piecewise-smooth reconstruction and its cross section (bottom).

To avoid the degenerate minimum of $N^{out} = 0$ in our minimization we impose additional constraints on the system. We require change in normal orientation to be within some prescribed expected noise level (γ_n). This comes in the form of imposing a bounded ℓ_∞-norm on the orientation change. We define our global optimization problem by:

$$N^{out} = \arg\min_{N} \sum_{(p_i,p_j)\in E} w_{ij} \left\| n_i - n_j \right\|_2 \quad \text{s.t.} \quad \forall_i \left\| n_i - n_i^{in} \right\|_2 \leq \gamma_n$$

This is a convex optimization problem. Imposing the additional normalization constraints $\|n_i^{out}\|_2 = 1$ will result in a non-convex optimization problem. Therefore we solve and renormalize the solution afterward. Renormalization is usually mild since we constrain orientations in the solution to be close to originals.

Initial orientations. Although our method reconstructs orientations based on their initial values, it does not require them as raw-inputs. Indeed, our implementation loosely approximates orientations from point-positions by applying a local PCA with fixed size neighborhoods. Although more advanced techniques exist for approximating point orientations [**84**], our algorithm successfully handles coarse initial orientation approximations. Therefore, using such a simple heuristic was sufficient in our case.

5.4.2. Positions Reconstruction. Given the reconstructed normal orientation obtained from the previous step, we reconstruct point position by assuming a local planarity criteria. For each pair of neighbor points $(p_i, p_j) \in E$ we define a penalty function $c_{ij}^X(X, N)$; it measures the projection distance of the points from the local plane defined by their average normal n_{ij} and average position x_{ij}:

$$c_{ij}^X(X, N) = \left| n_{ij} \cdot (x_i - x_j) \right| .$$

For a piecewise smooth surface the set $\{c_{ij}^X \geq \tau\}$ is sparse for some small τ depending on the smoothness of the surface and sampling resolution, so it makes sense to define a global weighted ℓ_1 penalty function

$$C^X(X, N, W, E) = \sum_{(p_i,p_j)\in E} w_{ij} c_{ij}^X(X, N) .$$

The weights $W = \{w_{ij}\}$ are designed to favor smooth parts over non-smooth parts, so we use the same formula as the one used for orientations, but we recompute them using the new orientation values.

We find the new positions as a minimization of the penalty function where N^{out} is already known and kept fixed:

$$X^{out} = \arg\min_X C^X(X, N^{out}, W, E)\,.$$

To minimize the amount of degrees of freedom we restrict ourselves to movement along normals. This fairly mild restriction has several benefits: it reduces the problem size and avoids the known effect of point clustering. Moreover, in our early experiments, we have noticed no significant benefit in using general movement. Thus, we define reconstructed positions as

$$x_i = x_i^{in} + t_i n_i^{out},$$

and our local penalty functions are

$$c_{ij}^X(X, N^{out}) = \left|n_{ij}^{out} \cdot (x_i - x_j)\right| = \left|(n_{ij}^{out})^T n_i^{out} t_i - (n_{ij}^{out})^T n_j^{out} t_j + (n_{ij}^{out})^T \cdot \left(x_i^{in} - x_j^{in}\right)\right|\,.$$

The goal penalty function becomes

$$C^X(X, N^{out}, W, E) = \|At + f\|_1\,,$$

where $A \in \mathbb{R}^{|E|\times|P|}$ and $f \in \mathbb{R}^{|E|}$ ($|.|$ is the size). Each row of A corresponds to a single $(p_i, p_j) \in E$, and is equal to

$$\left[\ \cdots\ \ w_{ij}(n_{ij}^{out})^T n_i^{out}\ \ \cdots\ \ -w_{ij}(n_{ij}^{out})^T n_j^{out}\ \ \cdots\ \right]\,.$$

Each index in f corresponds to a single pair $(p_i, p_j) \in E$ and is equal to

$$f_{ij} = (n_{ij}^{out})^T \cdot \left(x_i^{in} - x_j^{in}\right)\,.$$

We regularize the problem by constraining the norm of the correction term

$$\|t\|_2 \le \gamma_x,$$

where γ_x is a parameter proportional to the noise level. We use the following formula to determine γ_x:

$$\gamma_x = 0.7 \cdot \eta_x \ell(P)\sqrt{|P|}$$

where $\ell(P)$ is the length of the largest diagonal of the bounding box of the points in P, and η_x is the assumed noise level (in percents of the object size). Our implementation assumes η_x is a parameter provided by the user. Nevertheless, since η_x is directly related to noise, it can be approximated from the local point covariance matrix in the spirit of [**169**].

Overall, to find t we solve the following convex optimization problem:

$$\arg\min_t \|At + f\|_1 \ \text{ s.t. } \ \|t\|_2 \le \gamma_x\,.$$

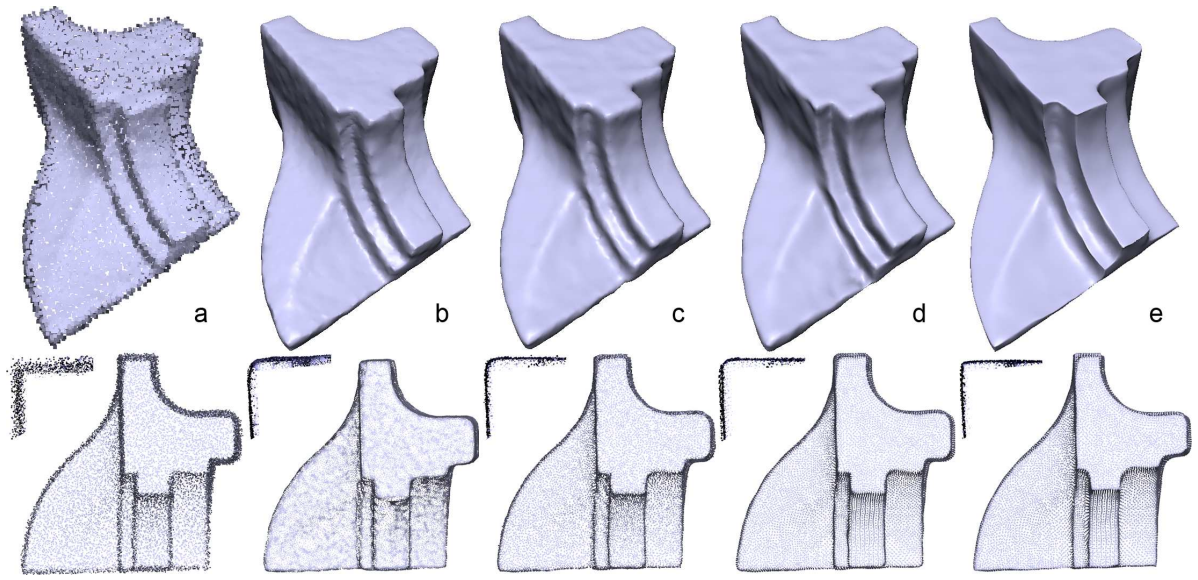


FIGURE 5.5.1. Comparison between our ℓ_1 method and state-of-the-art robust reconstruction methods: In (a) we show a noisy fandisk with noise in positions of 1% of the bounding box. Output LOP (b), DDMLS (c), RIMLS (d) and our ℓ_1 method (e). The bottom row contains, for each of the above, the cross-section of a corner and a top orthographic view.

5.5. An Efficient Convex Optimization Solver

In the previous sections we formulated our method as a sequence of convex optimization problems. These problems are nonlinear, but since they are convex they can be solved efficiently and reliably. Formulating the problem as an ℓ_1 minimization allows us to use an interior point solver, which usually converges faster than IRLS. We show in section 5.6 that our solver is indeed fast and reliable.

The problems described in Section 5.4 are second-order cone problems (SOCP). That is, problems of the form

$$\min_x p^T x \quad \text{s.t.} \quad \|A_i x + b_i\|_2 \leq c_i^T x + d_i.$$

We now describe a technique for solving the problem described in Section 5.4.2. A similar technique can be used to solve the problem described in Section 5.4.1. Nevertheless, for solving orientations we currently use the external package called `CVX` [**110**], which we found sufficient for our needs.

We recast

$$\arg\min_t \|At + f\|_1 \text{ s.t. } \|t\|_2 \leq \gamma_x \,.$$

as the second-order cone problem (SOCP) by first adding variables $u \in \mathbb{R}^M$. We set $x = (t, u)$ and set $p = (\ 0_{N\times 1} \quad 1_{M\times 1}\ )$. Each row of A adds two inequality constraints

$$A_{i,:}t - u + f_i \leq 0$$

and

$$-A_{i,:}t - u + f_i \leq 0$$

where $A_{i,:}$ is row i of A. We get the following equivalent SOCP:

$$\min_{t,u} \sum_{i=1}^{M} u_i \quad \text{s.t.} \quad \begin{aligned} & At - u + f \leq 0 \\ & -At - u - f \leq 0 \\ & \frac{1}{2}\left(\|t\|_2^2 - \gamma_x^2\right) \leq 0. \end{aligned}$$

where $M = |E|$ (size of adjacency list) and $N = |P|$ (number of points).

We implement a primal solver, using a log-barrier method [**47, 53**]. (See [**150**] for a primal-dual formulation). The method involves solving a series of nonlinear minimization problems of the form

$$(t_k, u_k) = \arg\min_{t,u} \sum_{i=1}^{M} u_i + \frac{1}{\tau_k} \sum_{i=1}^{2M+1} -\log\left(-g_i\left(t, u\right)\right),$$

where $\tau_k > \tau_{k-1}$ and each g_i corresponds to a constraint in the SOCP (each row of A defines two constraints; the constraint on the size of t is the last constraint). The inequality constraints $\{g_i\}$ have been incorporated into the functional via a penalty function which is infinite when constraints are violated and smooth elsewhere. As τ_k gets large the solution t_k approaches the optimal solution. The method stops when τ_k is large enough, based on the *duality gap* (see [**47, 53**] for more details). Each nonlinear minimization problem is solved iteratively using Newton's method, i.e. each Newton iteration involves solving a linear system of equations.

Let us write $g_{(1)} = At - u + f$, $g_{(2)} = -At - u - f$ and $g_{(\gamma)} = \frac{1}{2}\left(\|t\|_2^2 - \gamma_x^2\right)$. For a vector v denote by $D(v)$ the diagonal matrix with v on its diagonal, and v^{-1} the vector with the values in v inverted. For our SOCP each Newton iteration involves solving a series of *normal equations* of the form

$$\left(A^T \Sigma_t A - g_{(\gamma)}^{-1} \mathrm{I}_{N\times N} + g_{(\gamma)}^{-2} r r^T\right) \Delta t = w_0,$$

where $\Sigma_t = \Sigma_1 - \Sigma_2^2 \Sigma_1^{-1}$, $\Sigma_1 = D(g_{(1)})^{-2} + D(g_{(2)})^{-2}$, $\Sigma_2 = -D(g_{(1)})^{-2} + D(g_{(2)})^{-2}$ and $r = t \in \mathbb{R}^N$ is a dense vector. The solution of the linear system is the *search direction* of the corresponding Newton iteration.

Each Newton step requires the numerical solution of a linear system of equations. It is imperative to solve these equations efficiently, and this requires dealing with sparsity and conditioning issues. The matrix of the normal equations is symmetric positive definite, but for large scale problems it tends to be ill-conditioned, which in turn may result in an incorrect search direction. Furthermore, the vector r is dense, whereas the original problem is sparse. Therefore, factoring the matrix using the Cholesky decomposition may require a prohibitive amount of computational work and storage allocation. It is thus preferred, both from a numerical stability point of view and from a sparsity point of view, to adopt an iterative solution technique. We use the LSQR method [**166**], which is especially suited for least-squares problems. We write

$$\tilde{A} = \begin{pmatrix} \Sigma_t^{1/2} A \\ (-g_{(\gamma)})^{-1/2} \mathrm{I}_{N\times N} \\ g_{(\gamma)}^{-1} r^T \end{pmatrix},$$

and the search direction can now be found by solving

$$\min_{\Delta t} \left\| \tilde{A}\Delta t - w \right\|_2,$$

where

$$w = \begin{pmatrix} \Sigma_t^{-1/2}(g_{(1)}^{-1} - g_{(2)}^{-1} - \Sigma_2\Sigma_1^{-1}(\tau_k 1_{N\times 1} + g_{(1)}^{-1} + g_{(2)}^{-1})) \\ 0_{N\times 1} \\ 1 \end{pmatrix}$$

For brevity we omit the formula for w_0, but state that $w_0 = \tilde{A}^T w$.

To deal with the dense row associated with the vector r, we form a preconditioner using the strategy proposed in chapter 3 (and [**20**]). We remove the dense row from $\tilde{A}$; let us call the resulting matrix $\tilde{A}_0$. We then compute the Cholesky factorization of the *sparse* matrix associated with $\tilde{A}_0$:

$$R^T R = \tilde{A}_0^T \tilde{A}_0,$$

using CHOLMOD [**79**]. The R factor is used as a preconditioner for the augmented system associated with $\tilde{A}$, and now we apply LSQR. The important point here is that removing r amounts to a rank-1 change of the matrix that corresponds to the least squares operator. Therefore, only two iterations are needed for convergence in exact arithmetic (see chapter 3 and [**20**]). The matrix $\tilde{A}_0^T\tilde{A}_0$ may become very ill-conditioned, which can sometimes cause the Cholesky factorization to fail (by encountering a negative diagonal value due to inaccurate arithmetic). In such cases we use SuiteSparseQR [**75**] to compute a QR factorization instead and use R as a preconditioner.

Finally, we use one additional heuristic to speed up our solver. We have noticed that in some of the iterations, $\left|g_{(\gamma)}\right|$ tends to be considerably smaller than the maximum value on the diagonal of Σ_t. In those cases, LSQR on $\tilde{A}$ without a preconditioner tends to converge very quickly, because the singular values of $\tilde{A}$ are strongly clustered. We thus work on $\tilde{A}$ directly when conditioning allows for it (if $\|\Sigma_t\|_2 / \left|g_{(\gamma)}\right| \geq 10^3$), which saves the cost of a preconditioner solve.

The iterative method stops when the backward error drops below a certain threshold. This ensures backward stability relative to the desired level of accuracy. We use LSQR so the relevant condition number is $\kappa(\tilde{A})$ (and not $\kappa(\tilde{A}^T\tilde{A})$ as is the case for the normal equations). This ensures that a good search directions is found, which is crucial for the log-barrier method, even if low convergence threshold is used. The threshold we used in our experiments is 10^{-8}. As for accuracy of interior point method, its stopping criteria is based on a threshold as well. Here we found that only a coarse level of accuracy was sufficient and further adjustments had no visual significance.

5.6. Results and Discussion

We present results that demonstrate the viability and effectiveness of our method. For rendering purposes, our point set surfaces were meshed using the "Ball Pivoting" algorithm [**33**] and rendered using a standard illumination model.

We show in Figure 5.1.2 a simple example where we apply our method on a noisy 2D curve to recover its sharp features. In fact, for this simple synthetic model we can accomplish a perfect reconstruction, since it was sampled from a perfect piecewise linear "V" curve. ℓ_1 minimization generates an exact piecewise linear result with one singular

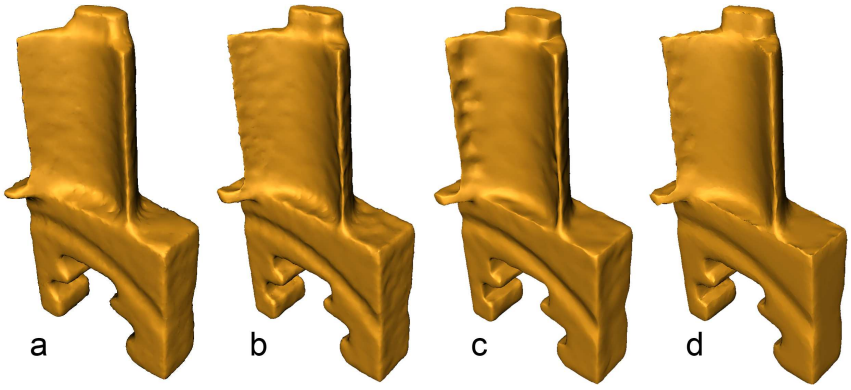


FIGURE 5.5.2. Comparison between our ℓ_1 with state-of-the-art reconstruction methods applied to a noisy blade model that originally contains sharp edges and smooth parts. We show the reconstruction results of LOP (a), DDMLS (b), RIMLS (c) and our ℓ_1 method (d).

point at the apex. For natural, real-world objects, we cannot expect such precise reconstruction. Nevertheless, using our global sparse method we obtain high quality results even for fairly complex sharp objects with relative high noise level (see Figure 5.4.2). The resulting models are piecewise smooth, and the error concentrates on edges, as can be seen in the corresponding zoomed regions and cross sections.

In Figures 5.5.1 and 5.5.2 we provide a comparison with state-of-the-art feature-aware reconstruction methods applied to the classical fandisk and blade models. In both models, we insert noise in the point positions in the amount of 1% of the bounding box size. In Figure 5.5.1, we show the results of LOP [**148**], DDMLS [**147**], RIMLS [**163**] and our method. The superiority of ℓ_1 over the other methods is evident; sharp features are recovered to a high level of accuracy, while smoothness is preserved in other regions.

Figure 5.5.2 shows a similar example where local feature-aware reconstruction methods (DDMLS, RIMLS) are able to faithfully recover sharp features (blade edges), however at the price of erroneously detecting local noise as redundant features (blade facets). Note also the lack of sharpness of some of the corners for the first two methods. In contrast, our global method does not have this flaw, and it correctly locates sparse singularities on the edges in a much cleaner manner.

For all the comparisons, we followed the guidelines provided in the distributed code by the authors and the parameters specified in their publications. We explicitly provide the parameters that we used:

- **DDMLS**: using cubic piecewise polynomials $m = 3$, local neighborhood size $h = 0.35\%$ of bounding box.
- **LOP**: max influence radius size $h = 0.3\%$ of bounding box, repulsion parameter $\mu = 0.3$, 10 iterations.
- **RIMLS**: local weight radii $h = [5 - 7]$ times local point spacing, degree of sharpness $\sigma_n = 0.75$, 15 projection iterations.

Figures 5.4.2 and 5.7.1 further illustrate the effectiveness of the ℓ_1 approach in terms of recovering the fine features of the object, while leaving the other parts smooth.

We have also applied our method to objects with less evident sharpness in the features. We observe that our method could handle such objects well (Figures 5.7.2 and 5.7.3). Note

TABLE 1. Running times of our algorithm

Model	Size (points)	Time (min)	Model	Size (points)	Time (min)
fandisk	17,106	3.5	*face*	110,551	12
blade	24,998	4	*Buddha*	150,737	27
knot	25,455	7	*funnel*	201,655	21
armadillo	99,416	16	*Escher*	240,909	22

that the ability to reconstruct smooth surfaces with no evident sharp features is due to the fact that our norm is ℓ_1 and not purely ℓ_0. In other words, we balance between sparsity and local smoothness. In smooth parts with no evident singularities ℓ_1-minimizations acts much like ℓ_2-minimization (much like the way median acts like mean when no outliers are present). In the scanned human face, we compared our global ℓ_1-minimization with a global ℓ_2-minimization. We note that even in this case, where sharpness is low, ℓ_1 provides a result of higher visual quality than ℓ_2.

In Figures 5.7.2 and 5.7.4 we demonstrate the behavior of our method in cases where the signal-to-noise ratio was low in the proximity of sharp features. In Figure 5.7.2, we reconstruct a scanned funnel that shows a shallow edge across the model. Figure 5.7.4 shows a similar shallow feature along the hand of a scanned Buddha statue. In both examples our method was able to recover sharpness to a large extent without smoothing it.

In Figure 5.7.5 we show the result of running the method on a large (240K points) scanned model of Escher's statue. It took our solver roughly 20 minutes to compute the optimal global solution. Note that despite its large size, our method does not fall into local minima. Our performance timings are reasonably good, considering the size of the problems. They are in the range of minutes (see Table 1) and were measured on a 64-bit Intel Core2 2.1 GHz using MATLAB.

Sensitivity and Parameters. Sensitivity to parameters is always a concern for methods of our type. Penalty methods at large require the choice of parameters whose optimal values are often either unknown or prohibitively expensive to compute (e.g. the smallest singular value of a linear operator). This is an area of active research in which progress is constantly being made, but much is still not understood.

Our algorithm is not particularly sensitive to parameters, especially in the positions phase. Table 2 provides a detailed list of the parameters involved in our method, and the actual range of values used in our experiments. The parameter range is narrow, and use of default parameters along with adjustments of γ_n and η_x according to the perceived noise level yields high quality results in a consistent fashion. Specifically, to obtain the results in this chapter we start with the default parameters. Next, based on the amount of noise and sharpness in the results we fine tuned the parameters to gain further improvement (see Figure 5.7.3 (c), (d) for a comparison between default and fine tuned parameters).

We do observe that the parameter that affects the output the most is γ_n. Since γ_n is a measure of noise level, we require a value large enough to obtain correct piecewise smooth normals while avoiding a too high value that may cause oversmoothing. This is a classical signal-to-noise ratio issue, which we explore in Figure 5.4.1, which shows three excerpts of a series of shallow V-shaped models of different angles and noise level. As expected, our method can handle less noise as the angle becomes shallower (i.e. weaker

TABLE 2. Parameters

Parameter	Range
Neighborhood size for initial (PCA) normal estimation (k)	$10 - 30$
Neighborhood size in orientation phase (k)	$3 - 6$
Max correction size of normals in first iteration (γ_n)	$0.10 - 0.15$
Max correction size of normals in second iteration (γ_n)	$0.03 - 0.07$
Angle for reweighting, normals phase (σ_θ)	$5^\circ - 20^\circ$
Neighborhood size in positions phase (k)	$8 - 12$
Assumed noise level in positions (η_x)	$0.02 - 0.03$
Angle for reweighting, positions phase (σ_θ)	$4^\circ - 20^\circ$

signal). The value of γ_n corresponds to noise level: 0.15 for 160°, 0.18 for 150° and 0.30 for 140°. Like many methods, our method tends to be more sensitive to parameters if the signal-to-noise ratio approaches the method's limit.

Some of this sensitivity may be attributed to the simple approach we use to compute initial orientations, using PCA with constant neighborhoods. In the presence of highly non-uniform sampling densities and noise, using a more refined technique for initial normals approximation may be more adequate. Finally, global optimization methods tend to be in general more sensitive to parameters than local methods.

On the other hand, our global approach has some clear advantages. Consider again the shallow V-shaped models of Figure 5.4.1. With no global considerations, a local method will keep many local sharp features or drop all of them. Using our global ℓ_1 method a single feature that concentrates all the error at one point is a feasible solution even for relatively low signal-to-noise ratios.

Limitations and Future Work. In some of our synthetic examples, our method has difficulty in correctly projecting points lying exactly on edge singularities. This is because orientations are essentially not defined there. Our method tends to concentrate errors on those edge samples, leaving them as error spikes outside of the model. We observed this phenomenon only in artificially noised synthetic models. This limitation can be overcome in the postprocessing stage, by applying a simple low-pass filter with a very small kernel that removes those singular spikes. For fairness, in our comparisons on synthetic objects we have applied the same filter for any other result that was improved by it. We did not apply the filter to any reconstruction of the real scanned objects.

Another limitation of our method is its relatively high computational cost. Our convex formulation incurs a high cost in the normals computation step, which has been addressed only partially so far. We believe, however, that it is possible to design a very efficient solver. Our experiments show that it is sufficient to solve the nonlinear iterates to a crude tolerance, and hence accelerate convergence, while keeping a high quality of the output. Furthermore, the preconditioner, which is based on a rank-1 correction, preserves sparsity of the underlying operator, and the overall iteration count for the optimization problem is fixed. As part of our future work, we plan to fully optimize our presented convex solver, and we believe that its performance can be improved considerably.

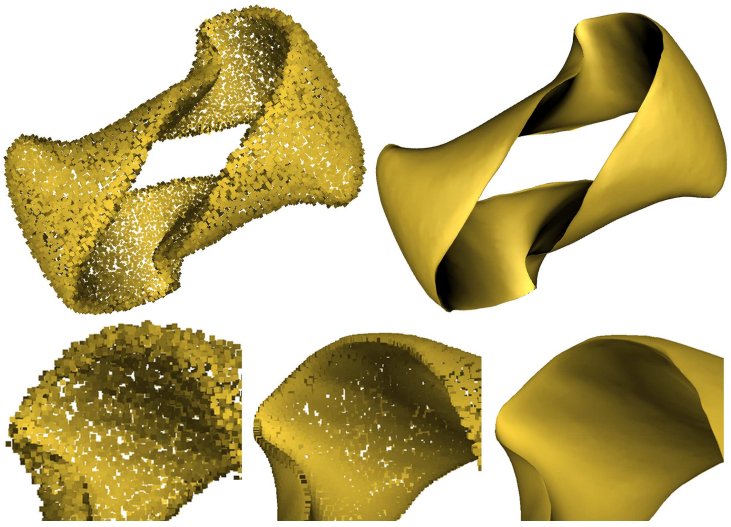


FIGURE 5.7.1. The ℓ_1 method applied to a sharp knot, with 1% noise added to the positions of the synthetic knot model (top left). The recovered knot appears on the top right. At the bottom, a zoom on a reconstructed sharp edge: from left to right – the initial noisy edge, point cloud after ℓ_1, and reconstruction.

5.7. Conclusions

We have introduced an ℓ_1-sparse approach for reconstruction of point set surfaces with sharp features. Our method is based on solving separately for the orientations and the positions and is formulated as a sequence of convex optimization problems. A key point here is that convexity allows for finding a global optimum and deriving efficient solvers. We incorporate a re-weighted technique to further enhance sparsity. A novel iterative solver tailored to the problem and based on a preconditioned iterative solver has been derived and implemented. As we demonstrate on several examples, the results are of high quality.

This work fits within a growing body of literature on methods whose principal goal is to enhance sparsity. We believe that the approach we use and our solution methodology may prove useful and effective for many problems in computer graphics. As part of our future work, we plan to look at extensions and other problems, and optimize the performance of our convex programming solver.

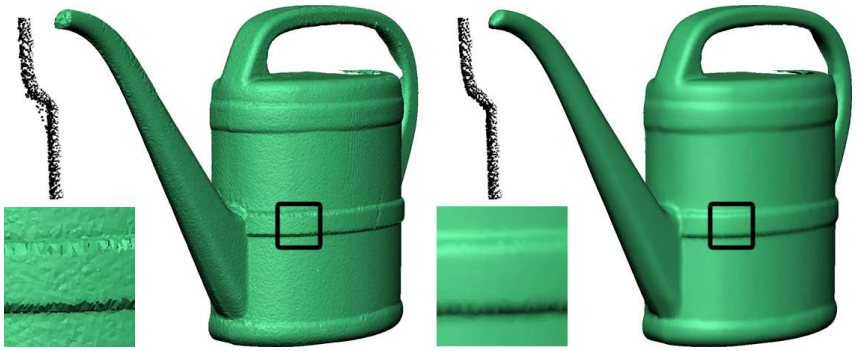


FIGURE 5.7.2. Result of applying our method to a scanned funnel (left). The scan contains shallow sharp features that are noisy.

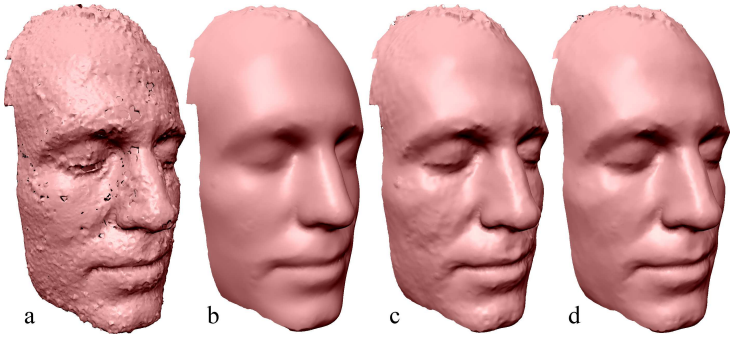


FIGURE 5.7.3. A comparison of minimization of ℓ_2-norm (b) vs. our ℓ_1 sparse minimization (c, d) on a scanned noisy human face (a). Although there are no pure sharp features, the method works well also on general models. In (c) we show the result of using our default parameters and (d) is obtained after fine tuning.

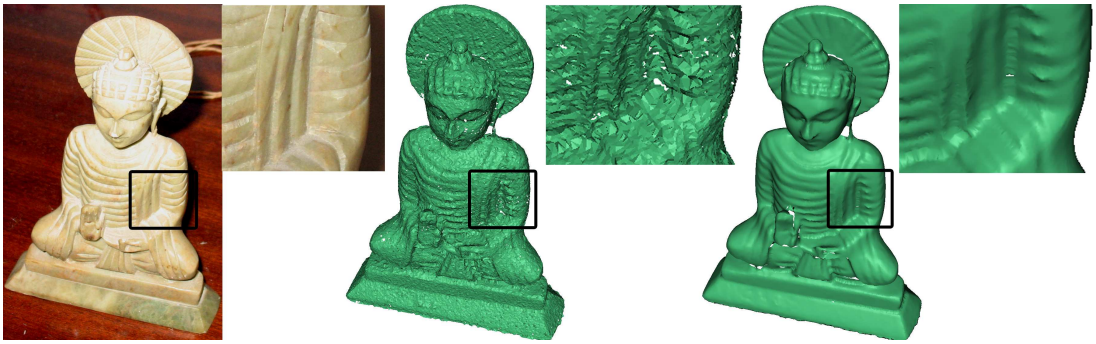


FIGURE 5.7.4. Result showing our method applied on a noisy scan of the Buddha statue (left). In the zoomed in regions of the hand, a small sharp feature is present in the original statue. Although hidden by the noise in the scan, our method was capable to pick it using a sparse global representation of the data.

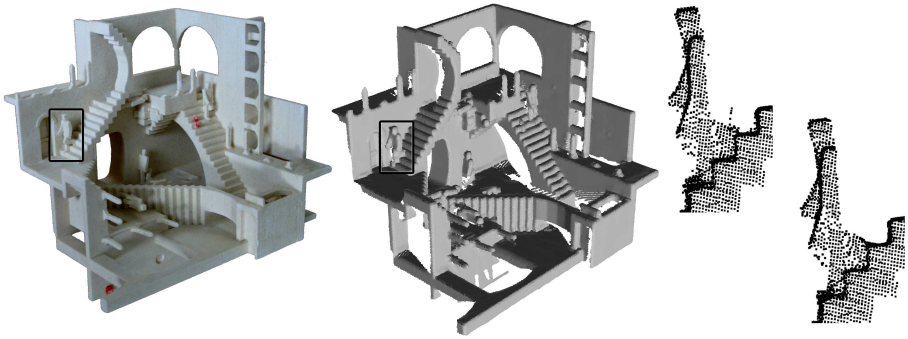


FIGURE 5.7.5. Result of applying our method to a large 240K noisy point cloud. Left is a highly complex physical model of an Escher statue, middle is the piecewise smooth point cloud after ℓ_1 optimization. On the right we show a zoomed cross-section comparison between the input (top) and our sharp result.

CHAPTER 6

Experimental study of solving HPD systems using indefinite incomplete factorizations

6.1. Introduction

Preconditioners based on incomplete factorization methods have long been used with Krylov subspace methods to solve large sparse systems of linear equations [**30, 184**]. While the Cholesky factorization LL^* of a Hermitian positive definite matrix is guaranteed to exist, there is no such guarantee of the existence of an incomplete factorization of this form. The reason is that the errors introduced due to dropping entries from the factor may result in zero or negative diagonal values.

The traditional approach to address this problem is to force positive definiteness by modifying the factorization process. Benzi's survey [**30**] of these methods notes that the various techniques tend to fall into two categories: simple and inexpensive fixes that often result in low-quality preconditioners, or sophisticated strategies yielding high-quality preconditioners that are expensive to compute. Some techniques to circumvent possible breakdown of incomplete Cholesky factorization involve using an LDL^* factorization, where D is diagonal; this can prevent breakdown in the construction of the preconditioner, but the preconditioner might be indefinite. One possibility, that has not been researched yet, is to compute an incomplete LDL^* factorization and force positive definiteness by perturbing tiny or negative entries in D after the factorization. A similar technique was used in chapter 3 (and [**20**]) to solve least-squares problems using perturbed QR factorizations. Gupta and George [**116**] propose switching from LL^* to LDL^* factorization upon encountering negative diagonals to complete the factorization without breakdown. Their approach does not require the preconditioner to be positive definite. An indefinite preconditioner can be problematic, even when the original matrix is positive definite, because it can result in a breakdown of the symmetric Krylov-subspace solvers like CG [**124**] and MINRES [**165**] . In CG, the breakdown is caused by a division by zero if the M^{-1}-norm of the residual becomes zero; In MINRES, the breakdown is caused when trying to compute the square root of a negative value, when the algorithm computes the M^{-1}-norm of the new basis vector. Furthermore, the correctness proof of both CG and MINRES rely on the existence of a Cholesky factor of the preconditioner [**184**].

As a result, the conventional wisdom has been that alternate Krylov-subspace methods, such as symmetric QMR [**101, 102**], GMRES [**185**], or BiCGStab [**217**], etc. must be used if the preconditioner is indefinite. However, using GMRES is expensive due to the long recurrence (expensive orthogonalization steps and a high memory requirement). Algorithms like QMR or BiCGStab do not minimize a norm of the residual or a norm of the error as GMRES, CG, and MINRES do. In general, it is not possible to get both optimality and a short recurrence with a non-symmetric method [**95**].

Algorithm 1 U-Conjugate Arnoldi Iteration

b = arbitrary, $q_1 = b/\|b\|_U$

for $n = 1, 2, 3, \ldots$

  $v = Aq_n$

  for $j = 1$ to n

    $h_{jn} = q_j^* U v$

    $v = v - h_{jn} q_j$

  end for

  $h_{n+1,n} = \|v\|_U$ (the algorithm fails if $h_{n+1,n} = 0$).

  $q_{n+1} = v/h_{n+1,n}$

Although not very well known, a there exists a variant of CG which allows an indefinite preconditioner[**14**]. We will refer to this variant as PCG-ODIR[1]. To the best of our knowledge this variant has not been experimentally compared to GMRES or QMR when an indefinite matrix is used to precondition an HPD system (the only implementation of PCG-ODIR that we are aware of is [**121**]). In this chapter[2] we experimentally explore this case and develop a new variant of PCG-ODIR that addresses the numerical problems demonstrated in the experiments. We also propose a new Krylov-subspace variant of MINRES that guarantee convergence and allow an indefinite preconditioner to be used.

6.2. The U-conjugate Arnoldi Iteration

The main tool that we use is a generalization of the classical Arnoldi iteration. The classical Arnoldi iteration forms, at step n, matrices Q_{n+1} and $\tilde{H}_n$ such that

$$AQ_n = Q_{n+1}\tilde{H}_n,$$

where $\tilde{H}_n$ is upper Hessenberg and Q_{n+1} is unitary. Instead of requiring Q_n to be unitary we require it to be unitary relative to the U-norm, where U is an Hermitian positive definite matrix. That is, we replace the condition

$$Q_n^* Q_n = I_{n\times n}$$

with the condition

$$Q_n^* U Q_n = I_{n\times n}\,.$$

To do so, all we need to do is replace dot-products with U inner-products, and 2-norms with U-norms. See Algorithm 1 for the pseudo-code. It is easy to see that the classical Arnoldi iteration is the U-conjugate iteration with $U = I_{N\times N}$ (where N is the number of rows in A).

[1] To be more precise, this variant is called simply PCG in [**14**]. In many cases the name PCG is used for the preconditioned version of the traditional CG, so we decided to use the name PCG-ODIR because this variants uses ODIR (unlike the the traditional preconditioned CG which uses OMIN).

[2] An earlier version of the results in this chapter were also reported in an IBM Technical Report, co-authored with Anshul Gupta and Sivan Toledo [**17**]. This chapter expands on the results of the technical report.

Like the classical Arnoldi iteration the U-conjugate Arnoldi iteration vectors span the the Krylov subspace. We omit the proof because it is identical to the proof that the classical Arnoldi iteration vectors span the Krylov subspace.

Theorem 6.2.1. *Let $q_1, \ldots, q_n$ be n vectors generated by a successful application of n iterations of Algorithm 1 on matrix A with initial vector b. Then,*

$$span\{q_1, q_2, \ldots, q_n\} = \mathcal{K}_n(A, b)\,.$$

The following theorem summarizes a few useful properties of the values generated by the U-conjugate Arnoldi iteration. Although the non-standard inner product Lanczos process (i.e., U-Conjugate Lanczos process) is present in the literature, the following theorem is new, to the best of our knowledge.

Theorem 6.2.2. *Let $\{q_i\}$and $\{h_{ji}\}$ be the values generated by the successful application of n iterations of Algorithm 1 on matrix A with initial vector b, where $1 \leq i, j \leq n$. Let*

$$Q_n = \begin{bmatrix} q_1 & q_2 & \cdots & q_n \end{bmatrix},$$

$$\tilde{H}_n = \begin{bmatrix} h_{11} & \cdots & h_{1n} \\ h_{21} & & \vdots \\ & \ddots & \vdots \\ & & h_{n+1} \end{bmatrix},$$

and

$$H_n = \left(\tilde{H}_n\right)_{1:n,1:n}.$$

Then,

(1) $AQ_n = Q_{n+1}\tilde{H}_n$,
(2) $Q_n^* U Q_n = I_{n\times n}$,
(3) $Q_n^* U A Q_n = H_n$.

Proof. The first two properties follow directly from the algorithm. Multiply the equation in property 1 by $Q_n^* U$ to get

$$Q_n^* U A Q_n = Q_n^* U Q_{n+1} \tilde{H}_n\,.$$

It is easy to see that

$$Q_n^* U Q_{n+1} = \begin{bmatrix} I_{n\times n} & 0_{n\times 1} \end{bmatrix},$$

so we have $Q_n^* U A Q_n = H_n$. □

The U-conjugate Arnoldi has a major disadvantage for a general A: the amount of work required to perform the nth iteration and amount of memory space needed is $O(nN + \mathrm{nnz}(A))$, where N is the number of rows in A. The classical Arnoldi reduces to a 3-term recurrence, and H_n is tridiagonal, if A is Hermitian. The U-conjugate Arnoldi iteration reduces to a three term recurrence, and H_n is tridiagonal, if $H_n = Q_n^* U A Q_n$ is Hermitian. This happens when UA is Hermitian. When this is the case, we call the resulting iteration *the U-Conjugate Lanczos Iteration* and we write T_n instead of H_n.

6.3. PCG-ODIR

At its core the Conjugate Gradients method generates at each iteration an A-conjugate basis for the Krylov subspace. That is

$$\text{span}\,\{q_1, q_2, \ldots, q_n\} = \mathcal{K}_n(A, b)$$

and

$$Q_n^* A Q_n = D_n$$

where

$$Q_n = \begin{bmatrix} q_1 & q_2 & \cdots & q_n \end{bmatrix}$$

and D_n is a diagonal matrix. Once we have found an A-conjugate basis the Conjugate Directions method can be used to produce an optimal A-norm approximation (see §7 in Shewchuk's tutorial [**192**]). The classical CG method couples the creation of the A-conjugate basis with the application of the conjugate directions method in a clever way. A preconditioner can be used, but it must be positive definite, otherwise the algorithm may fail (because of possible division by zero if the M^{-1}-norm of the residual becomes zero), and in any case the the correctness proof of CG rely on the existence of a Cholesky factor of the preconditioner [**184**].

It is well-known that the CG iteration can be formulated instead as a Lanczos process [**165**]. Using the Lanczos iteration we find an orthonormal basis U_n such that $U_n^* A U_n = T_n$ where T_n is tridiagonal and HPD. Let $T_n = R_n^* R_n$ be a Cholesky factorization of T_n and define

$$Q_n = U_n R_n^{-1}\,.$$

The columns of Q_n form an A-orthonormal basis. Unfortunately, we have not advanced towards a indefinitely-preconditioned version of CG: T_n must be positive definite, so the preconditioner must still be positive definite.

What is less known is that there is another, more straight-forward and robust, formulation of CG as a Lanczos process. It is based on the U-conjugate Lanczos process. If A is a Hermitian positive definite matrix, then we can use the U-conjugate Lanczos iteration to find an optimal A-norm approximate solution to $Ax = b$. We do so by applying the iteration on A, selecting $U = A$. After iteration n we have an A-conjugate basis to the Krylov subspace $\mathcal{K}_n(A, b)$. We can use the Conjugate Directions method to produce an optimal A-norm approximation (see §7 in Shewchuk's tutorial [**192**]). This version of CG appears in [**14**] under the name PCG-ODIR.

PCG-ODIR can be preconditioned quite easily. Suppose that we have formed an Hermitian preconditioner M. We can apply the A-conjugate Lanczos iteration to $M^{-1}A$ since $AM^{-1}A$ is Hermitian. Assuming we start our iteration with $M^{-1}b$, after the nth iteration we will find an n-dimensional A-conjugate basis to $\mathcal{K}(M^{-1}A, M^{-1}b)$. We can use that basis to find an optimal A-norm approximate solution $M^{-1}Ax = M^{-1}b$. The pseudo-code is listed in Algorithm 2.

If both M and A are Hermitian positive definite, then the classical CG algorithm produces an approximate in $\mathcal{K}_n(M^{-1}A, M^{-1}b)$ that minimizes the A-norm of the error; i.e., it finds an $x_n \in \mathcal{K}_n(M^{-1}A, M^{-1}b)$ such that $\|x_n - x\|_A$ is minimized. This minimizer is unique. This implies that under exact arithmetic, if the preconditioner is definite, then both classical CG and PCG-ODIR will produce the same vectors. Therefore, PCG-ODIR is indeed a different, and more robust, formulation of CG.

Algorithm 2 PCG-ODIR

Input: Hermitian positive definite A, a right hand side b and an Hermitian preconditioner M

$q_1 = M^{-1}b$
$l_1 = Aq_1$
$w = \sqrt{q_1^* l_1}$
$l_1 = l_1/w$
$q_1 = q_1/w$
$r^{(0)} = b$
$x^{(0)} = 0$
for $t = 1, 2, \ldots$
  $\gamma_t = q_t^* r^{(t-1)}$
  $x^{(t)} = x^{(t-1)} + \gamma_t q_t$
  $r^{(t)} = r^{(t-1)} - \gamma_t l_t$
  check for convergence
  $v_{t+1} = M^{-1} l_t$
  $H_{t,t} = l_t^* v_{t+1}$
  $H_{t-1,t} = H_{t,t-1} (= l_{t-1}^* v_{t+1})$
  $q_{t+1} = v_{t+1} - H_{t,t} q_t - H_{t-1,t} q_{t-1}$
  $l_{t+1} = Aq_{t+1}$
  $H_{t+1,t} = \sqrt{q_{t+1}^* l_{t+1}}$
  $l_{t+1} = l_{t+1}/H_{t+1,t}$
  $q_{t+1} = q_{t+1}/H_{t+1,t}$
end for

PCG-ODIR's advantage over classical CG is its ability to use an indefinite preconditioner and still to maintain the minimization properties. This advantage does not come without a price: while CG needs to store 5 vectors, and do 5 vector operations per iteration, PCG-ODIR needs to store 7 vectors, and do 13 vector operations per iteration.

PCG-ODIR's advantage over GMRES is the fact that it uses a Lanczos iteration, so it does not need to store all the bases. Its advantage over QMR and BiCGStab is that it minimizes a real norm of the error. Another potential advantage of PCG-ODIR over GMRES and QMR is the ability to base the stopping criteria on an estimate of the A-norm of the error. Indeed, the Hestenes-Stiefel estimate in classical CG can be easily incorporated in PCG-ODIR. More advanced methods have been proposed [**13, 107**], and some of them may be usable in PCG-ODIR.

6.4. Indefinitely Preconditioned CG

We have implemented PCG-ODIR and compared it to other algorithms (GMRES, QMR, BiCGStab) that work with indefinite preconditioners. The results of these comparisons appear in Section 6.6. Unfortunately, the performance is rather disappointing.

TABLE 1. Numerical stability: comparing full conjugation to local conjugation. In the OILPAN (NO PRECOND) instance, the convergence threshold was set to 10^{-5}.

Matrix	Droptol	Precond Definite?	PCG-ODIR	ORTHODIR	CG
CFD1	2×10^{-3}	NO	125 its	77 its	N/A
CFD1	4.5×10^{-4}	YES	85 its	69 its	85 its
CFD1	2×10^{-4}	YES	48 its	47 its	48 its
OILPAN	NO PRECOND	N/A	783 its	747 its	783 its
OILPAN	8×10^{-3}	NO	441 its	142 its	N/A
OILPAN	1.5×10^{-3}	NO	63 its	58 its	N/A
OILPAN	8×10^{-4}	YES	39 its	42 its	39 its
PWTK	4×10^{-3}	NO	149 its	103 its	N/A
PWTK	1×10^{-3}	NO	77 its	55 its	N/A
PWTK	8×10^{-4}	YES	61 its	55 its	61 its

PCG-ODIR is considerably faster than BiCGStab, but it is only slightly faster than QMR. It is faster than GMRES only when both use the same amount of memory, but not when both use the same preconditioner (in which case GMRES uses more memory). Theoretically, PCG-ODIR and GMRES should converge in about the same number of iterations, as both find approximate in the same Krylov-subspace. A natural suspect for the gap between the theoretical behavior and the actual behavior is the Lanczos process, which is known to lose orthogonality. Greenbaum [**111**] (§4) discusses the loss of orthogonality in the Lanczos process and its effect on CG and MINRES in detail.

A simple experiment verifies this hypothesis. Consider a version of PCG-ODIR where we use a long recurrence (which is more stable numerically) instead of a short recurrence (i.e., Arnoldi iteration instead of Lanczos iteration). Under exact arithmetic both the short recurrence and the long recurrence version of PCG-ODIR are equivalent. But, as Table 1 suggests, under inexact arithmetic this is not the case. We see that the long recurrence version of PCG-ODIR (labeled "ORTHODIR" the name used in [**14**]) does considerably fewer iterations then the short-recurrence version. We also see that CG sometimes performs many more iterations than ORTHODIR, but happens rarely. In particular, whenever CG performs well so does PCG-ODIR. This suggest that the numerical problems are due to short recurrence, and it is related to the quality of the preconditioner.

The formulation of CG as a Lanczos process was already helpful for allowing an indefinite preconditioner. We now use it to explain and deal with numerical stability issues. The basis vectors $q_1, q_2, \ldots$ are supposed to be A-conjugate, but due to rounding errors they lose conjugacy. As long as the loss of conjugacy is bounded, that is $\|I_{n\times n} - Q_n^* A Q_n\|_2 \leq \delta$ for some small δ, we will find iterates that are close to their ideal counterparts under exact arithmetic. Loss of conjugacy is not too severe if a long recurrence is used, but using a long recurrence is wasteful in memory and computation, and usually requires a restart at some stage. It is preferable to find a more economical method.

We propose the use of *selective orthogonalization* [**167**]. Instead of orthogonalizing the current iterate with respect to all previous basis vectors, we (incrementally) form a small set of vectors, say 5 vectors, and orthogonalize with respect to them (and with respect to the last two iterates, as in the short-recurrence form). This small set of vectors should be carefully selected so that it will restore A-conjugacy to Q_n as much as possible.

A celebrated result by Paige [**164**] shows how to find such vectors for the regular Lanczos process. Let $q_1, q_2, \ldots$ be the vectors formed by the Lanczos process on a Hermitian matrix A and let T_n be the tridiagonal matrix $T_n = Q_n^* A Q_n$. Let w_j $(j = 1, \ldots, n)$ be the eigenvectors of T_n and let $z_j = Q_n w_j$ be the corresponding Ritz vectors. Paige showed that under inexact arithmetic there are constants $\gamma_{j,n+1}$ of order of the rounding unit such that

$$z_j^* q_{n+1} = \frac{\gamma_{j,n+1}}{|\beta_{n+1} \mathrm{e}_j^T w_j|}$$

where e_j is the jth identity vector and β_{n+1} is a scalar computed during the Lanczos iteration. An iterate has a strong component in the direction of z_j only if $|\beta_{n+1}\mathrm{e}_j^T w_j|$ is small, which also means that the Ritz vector has converged or almost converged. Selective orthogonalization consists of saving converged and nearly converged Ritz vectors and orthogonalizing the Lanczos iterates against them.

We have formulated CG as a Lanczos process with a non-standard inner product, which opens the door for the use of selective orthogonalization.

To apply selective orthogonalization to PCG-ODIR, define $y_i = L^* q_i$ and $Y_i = L^* Q_i$ where $A = LL^*$is the Cholesky decomposition of A. Notice now that under exact arithmetic $\{y_i\}$ is the result of applying the regular Lanczos iteration on $L^* M^{-1} L$ and $L^* M^{-1} b$. Furthermore, Y_i is unitary and $Y_i^*(L^* M^{-1} L) Y_i = T_i$ for $i = 1, \ldots, t+1$. Assume that the rounding analysis of the Lanczos iteration applies to $\{y_i\}$ when viewed as the result of applying the Lanczos iteration on $L^* M^{-1} L$ (a non-trivial assumption, yet rounding analysis of PCG-ODIR is beyond the scope of this chapter), and let w_j $(j = 1, \ldots, t)$ be the eigenvectors of H_t and $z_j = Y_t w_j$ be the corresponding Ritz vectors. There exists constants $\gamma_{j,t+1}$ of order of the rounding unit such that

$$w_j Q_t^n A q_{t+1} = w_j Q_t^n L L^* q_{t+1} = z_j^* y_{t+1} = \frac{\gamma_{j,t+1}}{|H_{t+1,t} \mathrm{e}_j^T w_j|} .$$

Therefore, q_{t+1} has strong components in the A-direction of the converged or almost converged Ritz vectors (converged to the eigenvalues of $L^* M^{-1} L$).

IP-CG (Indefinitely Preconditioned CG), our variant of PCG-ODIR that uses selective orthogonalization, saves converged and nearly converged Ritz vectors and orthogonalizing the Lanczos iterates against them.

In practice, we found that in order to make the idea of selective orthogonalization practical, additional heuristics and compromises must be made. The following lists the set of heuristic and implementation details we used. This heuristics are based on careful engineering of the solver, and not on firm mathematical foundations.

(1) In order to limit the amount of memory used we do not keep more then a fixed number of converged Ritz vectors. In our experiments we kept a maximum of 8 converged Ritz vectors.
(2) Checking for convergence of Ritz vectors is expensive, and the cost grows as the iteration progresses. Therefore, we stop looking for additional Ritz vectors once

the iteration count is bigger then some parameter. In our experiments we do not check for convergence of the Ritz vectors after iteration 60.

(3) Checking for convergence of an eigenvalue can be done using only matrix H_t which is fairly small. Actually forming the converged Ritz vector requires all previous iterates. Keeping all iterates in memory will result in memory usage even larger than GMRES's. We noticed that convergence of a Ritz vector is not a frequent event. Therefore, we keep the previous iterates in secondary storage, and bring them into main memory only once convergence has occurred.

(4) It is not enough to keep only converged Ritz vectors. There can also be significant errors in the direction of almost converged Ritz vectors. Therefore, only mild convergence is required to keep a Ritz vector. In our experiments we keep a Ritz vector if $|H_{t+1,t}\mathrm{e}_j^T w_j| \leq 10^{-2}$.

(5) Even after a Ritz vector is kept, the next iterates may continue to improve its convergence. We therefore keep the most updated copy of every Ritz vector.

(6) Once the set of Ritz vectors has been updated it is important to correct the latest iterate of q_t, as well as the latest iterates of $x^{(t)}$ and $r^{(t)}$.

(7) Testing for convergence of the Ritz vectors is expensive, and it is advisable to avoid doing it in every iteration. We noticed that usually Ritz vectors converge only when loss of A-conjugacy starts to be significant. That occurs when $q_i^T A q_{t+1}$ is large for some $i = 1, \ldots, t$. Keeping all previous q iterates is memory demanding. Instead we inspect the value $|(\sum_{i=1}^{t} q_i^T) A q_{t+1}|$ instead. This value is a good indicator of the magnitude of $\max_i |q_i^T A q_{t+1}|$. We test for convergence once $|(\sum_{i=1}^{t} q_i^T) A q_{t+1}|$ is larger than some predetermined threshold. In our experiments we used the value of $1.49 \times 10^{-8} \approx \sqrt{\epsilon_{\text{machine}}}$.

6.5. Indefinitely Preconditioned MINRES

The MINRES algorithm can be used to solve $Ax = b$ for any Hermitian matrix, and a preconditioner can be used as long as it is Hermitian positive definite. In this section we will show a variant of MINRES that requires the opposite: any Hermitian preconditioner can be used as long as the matrix is positive definite.

Suppose that A is Hermitian positive definite, and that the preconditioner M is Hermitian. Like the algorithm used in Section 6.3, we use the A-conjugate Lanczos iteration on $M^{-1}A$ and $M^{-1}b$. We have found a matrix T_n and a basis Q_n to $\mathcal{K}_n = \mathcal{K}_n(M^{-1}A, M^{-1}b)$ with $M^{-1}AQ_n = Q_{n+1}\tilde{T}_n$ and $Q_n^* A Q_n = I_{n\times n}$. A is a Hermitian positive definite matrix, so there exists a lower triangular matrix L such that $A = LL^*$. We do not need to compute L, we use it only for the derivation of the algorithm. We will now show how Q_n and $\tilde{T}_n$ can be used to solve the equation $L^*M^{-1}Ax = L^*M^{-1}b$, which has exactly the same solution as $Ax = b$.

Algorithm 3 Indefinitely Preconditioned CG (IP-CG)

Input: Hermitian positive definite A, a right hand side b and an Hermitian preconditioner M

$q_1 = M^{-1}b$
$l_1 = Aq_1$
$w = \sqrt{q_1^* l_1}$
$l_1 = l_1/w$, $l_s = l_1$
$q_1 = q_1/w$
$r^{(0)} = b$
$x^{(0)} = 0$
for $t = 1, 2, \ldots$
  $\gamma_t = q_t^* r^{(t-1)}$
  $x^{(t)} = x^{(t-1)} + \gamma_t q_t$
  $r^{(t)} = r^{(t-1)} - \gamma_t l_t$
  check for convergence
  if (still looking for converged Ritz)
    save q_t and l_t to secondary storage
  $v_{t+1} = M^{-1} l_t$
  $H_{t,t} = l_t^* v_{t+1}$
  $H_{t-1,t} = H_{t,t-1} (= l_{t-1}^* v_{t+1})$
  $q_{t+1} = v_{t+1} - H_{t,t} q_t - H_{t-1,t} q_{t-1}$
  for each converged Ritz pair $(g, h = Ag)$
    $q_{t+1} \longleftarrow q_{t+1} - (h^* q_{t+1}) g$
  $l_{t+1} = Aq_{t+1}$
  $H_{t+1,t} = \sqrt{q_{t+1}^* l_{t+1}}$
  $l_{t+1} = l_{t+1}/H_{t+1,t}$
  $q_{t+1} = q_{t+1}/H_{t+1,t}$
  if (still looking for converged Ritz AND $\frac{1}{t} l_s^* q_{t+1} \geq$ threshold)
    find decomposition $H_t = VDV^T$
    for every column v of V such that $|v_t| \cdot H_{t+1,t} \leq$ threshold
      keep $(Q_t v, L_t v)$ as a Ritz pair (use q_t and l_t from secondary storage).
      $q_{t+1} \longleftarrow q_{t+1} - (v^* L_t^* q_{t+1}) Q_t v$
      $l_{t+1} \longleftarrow l_{t+1} - (v^* L_t^* q_{t+1}) L_t v$
      $x^{(t)} \longleftarrow x^{(t)} + (v^* Q_t^* r^{(t)}) Q_t v$
      $r^{(t)} \longleftarrow r^{(t)} - (v^* Q_t^* r^{(t)}) L_t v$
    end for
  end if
  $l_s \longleftarrow l_s + l_{t+1}$
end for

Let $\hat{Q}_n = L^* Q_n$. $Q_n^* A Q_n$ then reduces to $\hat{Q}_n^* \hat{Q}_n = I_{n \times n}$, so $\hat{Q}_n$ is a unitary matrix. Every $x \in \mathcal{K}_n$ can be written as $x = Q_n y$, so we have

$$\begin{aligned}
\min_{x \in \mathcal{K}_n} \|L^* M^{-1} A x - L^* M^{-1} b\|_2 &= \min_y \|L^* M^{-1} A Q_n y - L^* M^{-1} b\|_2 \\
&= \min_y \|L^* Q_{n+1} \tilde{T}_n y - L^* M^{-1} b\|_2 \\
&= \min_y \|\hat{Q}_{n+1} \tilde{T}_n y - L^* M^{-1} b\|_2 \\
&= \min_y \|\tilde{T}_n y - \hat{Q}_{n+1}^* L^* M^{-1} b\|_2 \\
&= \min_y \|\tilde{T}_n y - Q_{n+1}^* L L^* M^{-1} b\|_2 \\
&= \min_y \|\tilde{T}_n y - Q_{n+1}^* A M^{-1} b\|_2 \\
&= \min_y \|\tilde{T}_n y - \|M^{-1} b\|_A e_1\|_2.
\end{aligned}$$

We can iteratively find solutions y_n to $\min_y \|\tilde{T}_n y - \|M^{-1} b\|_A e_1\|_2$ and form $x_n = Q_n y_n$ in the same way as it is done in MINRES. As we can see we do not have to actually use L. We only rely on its existence. The pseudo-code is listed in Algorithm 4. We refer to this algorithm as IP-MINRES from here on.

A different and more technical way to derive IP-MINRES would be to write the equations for MINRES on $L^* M^{-1} L y = L^* M^{-1} b$ and multiply all vectors generated by the iteration by L^{-*}. The matrix L will disappear from the equations and we will get Algorithm 4. In order to streamline this chapter we do not give the details of this derivation.

IP-MINRES suffers from the same numerical problems as PCG-ODIR. Selective orthogonalization can be applied to IP-MINRES to produce a more robust version of IP-MINRES. In this study we restricted ourselves to study only the plain version of IP-MINRES (i.e., without selective orthogonalization), as it is new.

6.6. Numerical experiments and discussion

We have implemented the PCG-ODIR, IP-CG and IP-MINRES compared them to older algorithms (GMRES, QMR, BiCGStab, and CG). All the preconditioners (definite or indefinite) were built using WSMP [**115**]. We also used the implementation of GMRES, QMR, BiCGStab, and CG in that library. We stop the iterative method and declare convergence after the relative residual has dropped below 10^{-11}. We impose a limit of 1000 iterations and declare failure if the relative residual does not drop below 10^{-11} in 1000 iterations. Running times were measured on a 2.13 GHz Intel Core 2 Duo computer with 4 GB of main memory, running Linux 2.6. This computer has 2 processors, but our solver uses only one. All experiments are done in 64-bit mode.

Table 2 lists the SPD matrices used to test the indefinitely preconditioned solvers, along with their kind and sizes in terms of both dimension and the number of nonzeros. The matrices were obtained from the University of Florida sparse matrix collection [**72**].

6.6.1. Indefinite preconditioner. In this section, we list and analyze the results for instances where the preconditioner was indefinite. We compare PCG-ODIR, IP-CG and IP-MINRES to GMRES (without restarts and with restarts after 60 iterations), to the symmetric variant of QMR, and to BiCGStab. The results appear in Table 3. The

Algorithm 4 Indefinitely Preconditioned MINRES (IP-MINRES), without selective orthogonalization.

Input: Hermitian positive definite A, a right hand side b and an Hermitian preconditioner M
$q_1 = M^{-1}b$
$l_1 = Aq_1$
$w_1 = \sqrt{q_1^* l_1}$
$l_1 = l_1/w_1$
$q_1 = q_1/w_1$
$r^{(0)} = b - Ax$
$x^{(0)} = 0$
$s_{-2} = 0, s_{-1} = 0$
for $t = 1, 2, \ldots$ until convergence
  $v_{t+1} = M^{-1}l_t$
  $H_{t,t} = l_t^* v_{t+1}$
  $H_{t-1,t} = H_{t,t-1} (= l_{t-1}^* v_{t+1})$
  $q_{t+1} = v_{t+1} - H_{t,t}q_t - H_{t-1,t}q_{t-1}$
  $l_{t+1} = Aq_{t+1}$
  $H_{t+1,t} = \sqrt{q_{t+1}^* l_{t+1}}$
  $l_{t+1} = l_{t+1}/H_{t+1,t}$
  $q_{t+1} = q_{t+1}/H_{t+1,t}$
  $U_{t-2,t} = s_{t-2}H_{t-1,t}$
  if $(t > 2)$ $U_{t-1,t} = c_{t-2}H_{t-1,t}$ else $U_{t-1,t} = H_{t-1,t}$
  if $(t > 1)$ $U_{t,t} = -s_{t-1}U_{t-1,t} + c_{t-1}H_{t,t}$ else $U_{t,t} = H_{t,t}$
  $U_{t-1,t} = c_{t-1}U_{t-1} + s_{t-1}H_{t,t}$
  compute Givens rotation factors c_t and s_t on $\begin{bmatrix} U_{t,t} & H_{t+1,t} \end{bmatrix}^T$
  $U_{t,t} = c_t U_{t,t} + s_t H_{t+1,t}$
  $w_{t+1} = -s_t w_t$
  $w_t = c_t w_t$
  $m_t = (U_{t,t})^{-1}(q_t - U_{t-1,t}m_{t-1} - U_{t-2,t}m_{t-2})$
  $x^{(t)} = x^{(t-1)} + w_t m_t$
end for

results show that PCG-ODIR and our new algorithm converge when the preconditioner is indefinite, and that PCG-ODIR (and IP-CG) is indeed a more robust version of CG. As long as there are no restarts, in all but one instance, GMRES requires fewer iterations and converges faster than PCG-ODIR. IP-CG usually converges faster than PCG-ODIR, but not always (matrix LDOOR) and it sometimes fails (matrix ND24K). GMRES requires fewer iterations and converges faster than IP-CG as well, but only by a small margin. The comparison between IP-MINRES and GMRES is especially interesting: theoretically

TABLE 2. Test matrices

Matrix	N	NNZ	Kind
ROTHBERG/CFD1	70,656	1,825,580	CFD problem
ROTHBERG/CFD2	123,440	3,085,406	CFD problem
GHS_PSDEF/VANBODY	47,072	2,329,056	Structural problem
BOEING/PWTK	217,918	11,524,432	Structural problem
INPRO/MSDOOR	415,863	19,173,163	Structural problem
ND/ND24K	72,000	28,715,634	2D/3D problem
DNVS/X104	108,384	8,713,602	Structural problem
SCHENK_AFE/AF_SHELL7	504,855	17,579,155	Structural problem
GHS_PSDEF/BMWCRA_1	148,770	10,641,602	Structural problem
GHS_PSDEF/LDOOR	952,203	42,493,817	Structural problem
GHS_PSDEF/OILPAN	73,752	2,148,558	Structural problem
WISSGOTT/PARABOLIC_FEM	525,825	3,674,625	CFD problem
DNSV/SHIPSEC5	179,860	4,598,604	Structural problem
DNVS/SHIP_003	121,728	3,777,036	Structural problem

both algorithms are equivalent, but GMRES performs fewer iterations. This, again, shows that stability issues play a significant part when using a short recurrence. Table 3 also shows that running time is dominated by the cost of applying the preconditioner. PCG-ODIR and IP-MINRES do less operations-per-iteration than GMRES (and IP-CG), but GMRES is faster because it does less iterations.

PCG-ODIR and IP-MINRES are usually faster than QMR, but only marginally. IP-MINRES is theoretically superior to QMR since it minimizes the 2-norm of the residual, not a quasi-norm like QMR does. IP-CG is consistently faster than QMR, sometimes by a large margin, except for LDOOR, for which IP-CG did less iterations but took more time. It should also be noted that PCG-ODIR and IP-MINRES are more robust than QMR since they cannot breakdown (division by zero), like QMR can. A robust implementation of QMR needs to incorporate look ahead. The implementation of symmetric QMR that we use does not use look ahead. Both algorithms are faster than BiCGStab in all instances.

Our previous discussion revolved around running time. When we take into consideration memory consumption the picture is more complicated. PCG-ODIR, IP-CG and IP-MINRES use less memory than GMRES, so for memory-stressed scenarios (for example: solving a very large matrix, or solving several matrices concurrently) they allow a denser preconditioner. PCG-ODIR and IP-MINRES use less memory than IP-CG. The "Precond Density" column was added in order to explore the relation between running time and memory usage of different algorithms. The value in the density column is the ratio between the number of non-zeros in the incomplete factor and the number of rows in the matrix, that is the number of non-zeros required to store the incomplete factor is density $\times$ #columns. For a restart value of k, GMRES needs to store and additional k complete vectors, which is equivalent to an additional $k \times$ #columns non-zeros. IP-CG is more economical, since it stores only a small number of additional vectors (the "selected" vectors). For r converged Ritz vectors, IP-CG stores an additional $2r$ complete vectors, which is equivalent to an additional $2k \times$ #columns non-zeros. Recall that in our experiments IP-CG stored 8 Ritz vectors. Therefore, if we wish to compare the amount of

TABLE 3. Running time and number of iterations for instances in Table 2 in which the preconditioner is indefinite. Preconditioner density is the average number of non-zeros per column in the incomplete factor.

Matrix	Droptol	Precond Density	IP-MIN-RES	PCG-ODIR	IP-CG	GMRES(60)	GMRES	QMR	BiCGStab
CFD1	2×10^{-3}	197	127 its 23 sec	125 its 22 sec	83 its 20 sec	117 its 23 sec	77 its 19 sec	139 its 24 sec	165 its 43 sec
CFD2	2×10^{-3}	258	161 its 51 sec	160 its 51 sec	87 its 38 sec	87 its 37 sec	64 its 32 sec	174 its 53 sec	237 its 112 sec
VANBODY	2×10^{-3}	124	84 its 5.7 sec	85 its 5.8 sec	49 its 4.7 sec	48 its 5.5 sec	48 its 5.5 sec	87 its 5.8 sec	117 its 11.4 sec
PWTK	2×10^{-3}	177	97 its 38 sec	98 its 38 sec	73 its 35 sec	104 its 41 sec	71 its 34 sec	99 its 38 sec	94 its 57 sec
MSDOOR	8×10^{-4}	136	327 its 145 sec	336 its 146 sec	187 its 107 sec	358 its 179 sec	108 its 78 sec	338 its 146 sec	610 its 444 sec
MSDOOR	2×10^{-4}	139	36 its 41 sec	36 its 41 sec	28 its 42 sec	29 its 39 sec	29 its 39 sec	36 its 41 sec	40 its 55 sec
ND24K	4×10^{-4}	700	218 its 155 sec	217 its 154 sec	156 its 141 sec	179 its 145 sec	83 its 118 sec	270 its 169 sec	592 its 416 sec
X104	2×10^{-2}	168	67 its 22 sec	66 its 22 sec	44 its 22 sec	45 its 19 sec	45 its 19 sec	64 its 22 sec	90 its 38 sec
X104	2×10^{-3}	178	20 its 15 sec	20 its 15 sec	17 its 16 sec	18 its 15 sec	18 its 15 sec	20 its 15 sec	15 its 17 sec
LDOOR	2×10^{-3}	98	59 its 69 sec	62 its 69 sec	57 its 90 sec	59 its 75 sec	59 its 75 sec	66 its 71 sec	39 its 77 sec

memory used to store the preconditioner to the number of non-zeros to store the vectors in GMRES (and is not needed in PCG-ODIR) we need to compare k and density, and if we want to compare PCG-ODIR to IP-CG we need to compare $2r$ and density.

If we examine the results for the two instances of MSDOOR, we see that PCG-ODIR with the denser preconditioner (droptol $= 2 \times 10^{-4}$) uses less memory and is faster than IP-CG and GMRES (with any reasonable restart value) with a sparser preconditioner (droptol $= 8 \times 10^{-4}$). This is also true for the two instances of X104. IP-CG is with the denser preconditioner uses less memory and his faster than GMRES with a sparser preconditioner. IP-CG is more numerically accurate than PCG-ODIR, and GMRES is more numerically accurate than IP-CG. Each improvement in numerical accuracy requires additional memory, and this memory can instead be used to build a denser preconditioner. Our experiments indicate that in some cases it is better to forgo numerical accuracy and spend the additional memory to build a better preconditioner.

We explore this issue further in Table 4. In this set of experiments we have taken the largest matrix in our suite, ND24K, and solve it using different drop-tolerance values. The results show that GMRES is faster than PCG-ODIR, but if we want to examine what can happen on a memory-tight situation we should compare the "Precond Density" column to the restart value. From Table 4, we see that the minimum amount storage

TABLE 4. Detailed results for matrix ND24K.

Droptol	Precond Density	PCG-ODIR	IP-CG	GMRES	GMRES(120)	GMRES(200)
8×10^{-4}	574	FAIL	FAIL	439 its 188 sec	FAIL	FAIL
7×10^{-4}	528	FAIL	FAIL	338 its 146 sec	FAIL	2000 its 569 sec
6×10^{-4}	553	FAIL	FAIL	396 its 181 sec	FAIL	FAIL
5×10^{-4}	631	582 its 233 sec	320 its 169 sec	118 its 116 sec	118 its 116 sec	118 its 116 sec
4×10^{-4}	700	217 its 154 sec	156 its 140 sec	83 its 118 sec	83 its 118 sec	83 its 118 sec
3×10^{-4}	719	192 its 156 sec	124 its 142 sec	78 its 128 sec	78 its 128 sec	78 its 128 sec
2×10^{-4}	792	141 its 167 sec	78 its 150 sec	60 its 143 sec	60 its 143 sec	60 its 143 sec
1×10^{-4}	854	46 its 209 sec	35 its 211 sec	35 its 207 sec	35 its 207 sec	35 its 207 sec

by GMRES to solve the system in reasonable time is $749 \times$ #columns (drop-tolerance 5×10^{-4}). The minimum amount of memory needed by PCG-ODIR is $631 \times$ #columns. The difference $118 \times$#columns can be the difference between being able to solve the matrix on a given machine, or not. IP-CG is faster than PCG-ODIR, but in terms of memory it falls between PCG-ODIR and GMRES. The minimum amount of memory needed by PCG-ODIR is $649 \times$ #columns, more than PCG-ODIR but is faster (and still much less than GMRES). One can argue that a scenario where PCG-ODIR and IP-CG are possible but not GMRES is plausible.

6.6.2. Positive definite preconditioner. In this section, we list and analyze the results for instances where the preconditioner was definite. We compare PCG-ODIR, IP-CG and IP-MINRES to CG and to GMRES (without restart). Usually, when both the matrix and the preconditioner are positive definite CG is used. Under exact arithmetic PCG-ODIR is identical to CG. The goal of this set of experiments is to check whether PCG-ODIR's performance is similar to CG's under finite-accuracy arithmetic. We also wish to check, using the comparison to GMRES, IP-MINRES's sensitivity to numerical instabilities, and also investigate if selective orthogonalization help in this case too.

The results appear in Table 5. In all the instances listed in Table 5, the preconditioner is definite. The results show that indeed PCG-ODIR acts very similar to CG and converges at about the same number of iterations (with cases of slight advantage to both algorithms). CG performs fewer operations per iteration, so it is a bit faster. Nevertheless, PCG-ODIR is more robust, being able to handle an indefinite preconditioner, so the user can trade a few percents of performance for increased robustness.

The comparison of IP-MINRES and PCG-ODIR to GMRES show that the numerical instabilities encountered when using an indefinite preconditioner no longer appear when

TABLE 5. Running time and number of iterations for instances in Table 2 where the preconditioner is definite.

Matrix	Droptol	Precond Density	IP MINRES	PCG-ODIR	IP-CG	CG	GMRES
AF_SHELL7	2×10^{-3}	97	128 its 59 sec	137 its 60 sec	137 its 62 sec	137 its 57 sec	131 its 81 sec
BMWCRA_1	2×10^{-3}	215	128 its 59 sec	137 its 60 sec	FAIL	137 its 57 sec	147 its 61 sec
LDOOR	2×10^{-4}	122	17 its 57 sec	16 its 56 sec	17 its 58 sec	17 its 56 sec	18 its 58 sec
OILPAN	8×10^{-4}	89	39 its 3.8 sec	39 its 3.7 sec	37 its 4.5 sec	39 its 3.5 sec	39 its 3.6 sec
PARABOLIC_FEM	2×10^{-3}	19	68 its 13.7 sec	73 its 13.6 sec	73 its 15.1 sec	73 its 11.6 sec	70 its 19.3 sec
SHIPSEC5	2×10^{-3}	95	45 its 11.4 sec	46 its 11.3 sec	46 its 12.4 sec	45 its 10.7 sec	47 its 12.3 sec
SHIP_003	2×10^{-3}	108	84 its 13.1 sec	85 its 13.0 sec	84 its 14.5 sec	89 its 12.7 sec	87 its 15.5 sec

the preconditioner is definite. In most cases, IP-MINRES requires fewer iterations than GMRES and is faster. This explains why our experiments suggest that PCG-ODIR outperforms IP-CG when the preconditioner is definite. If the preconditioner is good (as often happens when it is definite) not much is gained from IP-CG complex orthogonalization strategy, while running time per iteration increases.

6.6.3. Using an indefinite preconditioner vs. forcing definiteness. An alternative to using an indefinite preconditioner is to somehow force the incomplete factorization to produce a definite preconditioner. A detailed experimental study of which strategy is better is beyond the scope of this chapter. The goal of this set of experiment is to show that there are cases where it would be preferable to use an indefinite preconditioner.

There are many methods by which definiteness can be forced [**30**]. We have chosen to test one of these methods. More specifically, we tried the method suggested by Manteuffel [**154**]. This method tries to find a value α such that the incomplete factorization of $\hat{A} = A + \alpha \text{diag}(A)$ is positive definite, and uses that factor as a preconditioner. The value of α is found using a trial-and-error method that can be expensive. Obviously, the quality of the preconditioner depends on the value of α that was used. For our comparison we decided not to use trial-and-error method due to its cost. Instead, we chose to try two values for α, a small value and a large value, for all three matrices in this set of experiments.

The results appear in Table 6. As can be seen from this table, forcing positive definiteness produced a better preconditioner in some cases, and a worse one in others. This demonstrates the effectiveness of PCG-ODIR and IP-CG methods, in that they provided reasonable results without a tuning parameter.

TABLE 6. Comparing strategies: using an indefinite preconditioner or forcing definiteness.

Matrix	Droptol	PCG-ODIR	IP-CG	CG, run 1 $\alpha = 0.01$	CG, run 2 $\alpha = 0.001$
CFD1	2×10^{-3}	125 its 22 sec	83 its 20 sec	112 its 17 sec	99 its 18 sec
MSDOOR	8×10^{-4}	336 its 146 sec	187 its 107 sec	FAIL: res = 1.1×10^{-10} after 1000 its	627 its 180 sec
X104	2×10^{-3}	20 its 15 sec	17 its 16 sec	FAIL: res = 1.6×10^{-10} after 1000 its	FAIL: res = 5.1×10^{-10} after 1000 its

TABLE 7. Numerical stability: comparing full conjugation to local conjugation. In the OILPAN (NO PRECOND) instance, the convergence threshold was set to 10^{-5}.

Matrix	Droptol	Precond Definite?	PCG-ODIR	ORTHODIR	IP-CG	IP-MINRES	GMRES	CG
CFD1	2×10^{-3}	NO	125 its	77 its	83 its	127 its	77 its	N/A
CFD1	4.5×10^{-4}	YES	85 its	69 its	69 its	84 its	69 its	85 its
CFD1	2×10^{-4}	YES	48 its	47 its	47 its	48 its	46 its	48 its
OILPAN	NO PRECOND	N/A	783 its	747 its	770 its	297 its	242 its	783 its
OILPAN	8×10^{-3}	NO	441 its	142 its	278 its	437 its	130 its	N/A
OILPAN	1.5×10^{-3}	NO	63 its	58 its	52 its	64 its	51 its	N/A
OILPAN	8×10^{-4}	YES	39 its	42 its	37 its	39 its	36 its	39 its
PWTK	4×10^{-3}	NO	149 its	103 its	104 its	149 its	103 its	N/A
PWTK	1×10^{-3}	NO	77 its	55 its	55 its	77 its	55 its	N/A
PWTK	8×10^{-4}	YES	61 its	55 its	54 its	61 its	54 its	61 its

6.6.4. Numerical stability: full conjugation vs. local conjugation. The results in Table 3 indicate that the new solvers often do not fulfill their theoretical potential when the preconditioner is indefinite and they tend to require more iterations than GMRES. We already explored this issue in section 6.4. We now explore it further, by adding IP-CG, IP-MINRES and GMRES to the comparison. The results appear in table 7.

From the results, we see that often a long recurrence needs considerably fewer iterations. Other times, the short recurrence works equally as well as the long recurrence. Adding selective orthogonalization helps, but there are still cases where ORTHODIR does less iterations. The experiments also show that numerical problems are not directly connected to the use of an indefinite preconditioner: we have cases where the problem

manifests for a definite preconditioner (CFD1-4.5×10^{-4}, OILPAN-NO PRECOND) and cases where manifests very weakly for an indefinite preconditioner (OILPAN-1.5×10^{-3}). There are cases where CG converges slower than it should even though the preconditioner is definite, so apparently both PCG-ODIR and CG suffer from the same numerical instability. In those cases IP-CG does less iterations than CG, reinforcing our conclusion that CG suffers from numerical instabilities too. There seems to be a connection between the quality of the preconditioner and numerical instability encountered. Indefinite incomplete factorization tend to be lower quality preconditioners because the indefiniteness in the incomplete factors indicates that incomplete factorization dropped non-zeros too aggressively.

6.7. Conclusions and Open Questions

We experimentally evaluated the performance of PCG-ODIR, a less well-known variant of CG. We also presented two new versions of CG and MINRES. Unlike classical CG and MINRES, both PCG-ODIR and the new algorithms accept a Hermitian indefinite preconditioner. The motivation for using these algorithms is the possible failure of incomplete factorization to produce a positive definite preconditioners inexpensively. We have conducted extensive numerical experiments and have compared the new solvers with CG, GMRES, symmetric QMR, and BiCGStab. We have demonstrated the robustness and the utility of this approach in many cases. Theoretically, GMRES is the optimal algorithm since it finds the minimum residual solution, but it does not use a short recurrence. Symmetric QMR and BiCGStab are sub-optimal (for example, QMR minimizes a quasi-norm and not the real norm), but they use a short recurrence. The algorithms we analyzed bridge the gap: they are theoretically optimal and they use a short recurrence.

The experiments show that PCG-ODIR (and IP-MINRES) does not always fulfill their full theoretical potential and GMRES usually converges in fewer iterations. Our analysis suggests that the problem is caused by numerical instabilities in the Lanczos process, and that CG too suffers from the same problem. We explored the strategy of selective orthogonalization to handle the numerical issues, and suggest the variant IP-CG based on this strategy. This variant does fewer iterations and is faster than PCG-ODIR. It uses more memory than PCG-ODIR, but less memory then GMRES.

Although GMRES usually converges faster than PCG-ODIR for the same preconditioner, PCG-ODIR often outperform GMRES by using a denser and more accurate incomplete factorization to compensate for the extra memory that GMRES requires. The same is true for IP-CG and IP-MINRES. A more detailed experimental study is required to compare the combination of a denser preconditioner and short recurrence solvers with that of a sparser preconditioner and GMRES. Another interesting question that arises from this chapter is whether it is better to use the incomplete factorization process as-is, even if the preconditioner turns out to be indefinite, or to use incomplete factorization methods that guarantee a positive definite preconditioner? A comprehensive experimental study would be required to answer this question, since there are many different methods to enforce positive definiteness [**30**]. Finally, we note that it is worth investigating whether the new methods have any advantages over their conventional counterparts in formulating communication avoiding Krylov-subspace methods (see [**127**] for details).

CHAPTER 7

Randomized algorithms for estimating the trace of an implicit symmetric positive semi-definite matrix

7.1. Introduction

Finding the trace of an explicit matrix is a simple operation. But there are application areas where one needs to compute the trace of an implicit matrix, that is, a matrix represented as a function. For example, in lattice Quantum Chromodynamics, one often needs to compute the trace of a function of a large matrix, trace($f(A)$). Explicitly computing $f(A)$ for large matrices is not practical, but computing the bilinear form $x^T f(A)x$ for an arbitrary x is feasible [**25, 26**]. Other examples include the regularized solution of least-squares problems using the Generalized Cross-Validation approach (see [**130**]) and computing the number of triangles in a graph [**214, 15**].

The standard approach for computing the trace of an implicit function is Monte-Carlo simulation, where the trace is estimated by $\frac{1}{M}\sum_{i=1}^{M} z_i^T A z_i$, where the z_i are random vectors. The original method is due to Hutchinson [**130**]. Although this method has been improved over the years ([**29, 131, 224**]), no paper to date has presented a theoretical bound on the number of samples required to achieve an ϵ-approximation of the trace; only the variance of estimators has been analyzed.

This chapter[1] makes four significant contributions to this area:

(1) We provide rigorous bounds on the number of Monte-Carlo samples required to achieve a maximum error ϵ with probability at least $1 - \delta$ in several trace estimators. The bounds are surprising: the method with the best bound is not the method with the smallest variance.
(2) We provide specialized bounds for the case of projection matrices, which are important in certain applications.
(3) We propose a new trace estimator in which the z_is are random columns of a unitary matrix with entries that are small in magnitude. This estimator converges slower than known ones, but it also uses fewer random bits.
(4) We experimentally evaluate the convergence of the three methods on a few interesting matrices.

7.2. Hutchinson's Method and Related Work

The standard Monte-Carlo method for estimating the trace of an implicit method is due to Hutchinson [**130**], who proved the following Lemma.

[1]The results of this chapter are also reported in a recently submitted paper, co-authored with Sivan Toledo [**24**].

Lemma 7.2.1. *Let A be an $n \times n$ symmetric matrix with* $\mathrm{trace}(A) \neq 0$. *Let z be a random vector whose entries are i.i.d Rademacher random variables (*$\Pr(z_i = \pm 1) = 1/2$*). $z^T Az$ is an unbiased estimator of* $\mathrm{trace}(A)$ *i.e.,*

$$\mathrm{E}(z^T Az) = \mathrm{trace}(A)$$

and

$$\mathrm{Var}(z^T Az) = 2\left(\|A\|_F^2 - \sum_{i=1}^{n} A_{ii}^2\right).$$

If we examine the variance term we see that intuitively it measures how much of the matrix's "energy" (i.e., the Frobenius norm) is on the diagonal. It is easy to see that for a general matrix Hutchinson's method can be ineffective because the variance can be arbitrarily large. Even for a symmetric positive definite the variance can be large: the variance for the matrix of all 1's, which is symmetric semi-definite, is $2(n^2 - n)$, whereas the trace is only n. This matrix can be perturbed to definiteness without a significant impact on the trace or variance. Such a large variance precludes the use of Chebyshev's inequality to bound the number of iterations required to obtain a given relative error in the trace. For such a bound to hold, the variance must be $o(\mathrm{trace}(A)^2)$.

Lemma 7.2.1 does not give a rigorous bound on the number of samples/matrix multiplications. This difficulty carries over to applications of this method, such as [**25, 26**]. Hutchinson's method has been improved over the years, but the improvements do not appear to have addressed this issue. Wong et al. [**224**] suggest using test vectors z that are derived from columns of an Hadamard matrix. Iitaka and Ebisuzaki [**131**] generalized Hutchinson's estimator by using *complex* i.i.d's with unit magnitude; they showed that the resulting estimator has lower variance than Hutchison's (but the computation cost is also higher). Silver and Röder [**196**] use Gaussian i.i.d variables, but without any analysis. Bekas et al. [**29**] focus on approximating the actual diagonal values, also using vectors derived from an Hadamard matrix.

In Section 7.7 below we show that it is possible to bound the number of samples required for Hutchinson's method. However, by the bound that we obtain is not as tight as the bound we obtain when the entries of z are i.i.d normal variables.

7.3. Three and an Half Estimators

In this section we describe the trace estimators that we analyze. We describe three estimators and a variant of one of them. All estimators follow the same basic pattern: a random vector z is drawn from some fixed distribution, and $z^T Az$ is used to estimate the trace. This procedure is repeated M times using i.i.d samples and the estimates are averaged.

The first estimator uses vectors whose entries are standard Gaussian (normal) variables.

Definition 7.3.1. A *Gaussian trace estimator* for a symmetric positive-definite matrix $A \in \mathbb{R}^{n \times n}$ is

$$G_M = \frac{1}{M}\sum_{i=1}^{M} z_i^T Az_i \,,$$

where the z_i's are M independent random vectors whose entries are i.i.d standard normal variables.

The Gaussian estimator does not constrain the 2-norm of the z_i's; it can be arbitrarily small or large. All the other estimators that we analyze normalize the quadratic forms by constraining $z^T z$ to be equal to n. This property alone allows us to prove below a general convergence bound.

Definition 7.3.2. A *normalized Rayleigh-quotient trace estimator* for a symmetric positive semi-definite matrix $A \in \mathbb{R}^{n \times n}$ is

$$R_M = \frac{1}{M} \sum_{i=1}^{M} z_i^T A z_i \ ,$$

where the z_i's are M independent random vectors such that $z_i^T z_i = n$ and $\mathrm{E}(z_i^T A z_i) = \mathrm{trace}(A)$.

The second estimator we analyze is Hutchinson's.

Definition 7.3.3. An *Hutchinson trace estimator* for a symmetric positive-definite matrix $A \in \mathbb{R}^{n \times n}$ is

$$H_M = \frac{1}{M} \sum_{i=1}^{M} z_i^T A z_i \ ,$$

where the z_i's are M independent random vectors whose entries are i.i.d Rademacher random variables.

The first two estimators use a very large sample spaces. The Gaussian estimator uses continuous random variables, and the Hutchinson estimator draws z from a set of 2^n vectors. Thus, the amount of random bits required to form a sample is $\Omega(n)$. Our third estimator samples from a set of n vectors, so it only needs $O(\log n)$ random bits per sample. We discuss the issue of randomness and it implications further in the next section. The third estimator samples from a smaller family by estimating the trace in a more direct way: it samples the diagonal itself. The average value of a diagonal element of A is $\mathrm{trace}(A)/n$. So we can estimate the trace by sampling a diagonal element and multiplying the result by n. This corresponds to sampling a unit vector from the standard basis and computing the Rayleigh quotient.

Definition 7.3.4. A *unit vector estimator* for a symmetric positive-definite matrix $A \in \mathbb{R}^{n \times n}$ is

$$U_M = \frac{n}{M} \sum_{i=1}^{M} z_i^T A z_i \ ,$$

where the z_i's are M independent uniform random samples from $\{e_1, \ldots, e_n\}$.

In contrast to previous methods, the quadratic forms in the unit-vector estimator do not depend in any way on the off-diagonal elements of A, only on the diagonal elements. Therefore, the convergence of U_M is independent of the off-diagonal elements. The distribution of diagonal elements does influence, of course, the convergence to $\mathrm{trace}(A)/n$. For some matrices, this method must sample all the diagonal elements for U_M to be close to $\mathrm{trace}(A)$. For example, if A has one huge diagonal element, the average is useless until

we sample this particular element. On the other hand, if all the diagonal elements are the same, the average converges to the exact solution after one sample.

Our last estimator is a variant of the unit vector estimator that uses randomization to address this difficulty. Instead of computing the trace of A, it computes the trace of $\mathcal{F}A\mathcal{F}^T$ where $\mathcal{F}$ is a unitary matrix. Since the *mixing matrix* $\mathcal{F}$ is a unitary, $\text{trace}(A) = \text{trace}(\mathcal{F}A\mathcal{F}^T)$. We construct $\mathcal{F}$ using a randomized algorithm that guarantees with high probability a relatively flat distribution of the diagonal elements of $\mathcal{F}A\mathcal{F}^T$. More precisely, we construct $\mathcal{F}$ in a way that attempts to flatten the distribution of all the elements of $\mathcal{F}A\mathcal{F}^T$, not just its diagonal elements. We use this strategy because we do not know how to flatten the diagonal elements alone. Our constructions are based on the random mixing matrices suggested in [4].

Definition 7.3.5. A *random mixing matrix* is a unitary $\mathcal{F} = FD$, where F and D are n-by-n unitary matrices. The matrix F is a fixed unitary matrix called the *seed* matrix. The matrix$D$ is a unitary random diagonal matrix with diagonal entries that are i.i.d Rademacher random variables: $\Pr(D_{ii} = \pm 1) = 1/2$.

Definition 7.3.6. A *mixed unit vector estimator* for a symmetric positive semi-definite matrix $A \in \mathbb{R}^{n\times n}$ is

$$T_M = \frac{n}{M}\sum_{i=1}^{M} z_i^T \mathcal{F}A\mathcal{F}^T z_i \ ,$$

where the z_i's are M independent uniform random samples from $\{e_1, \ldots, e_n\}$, and $\mathcal{F}$ is a random mixing matrix.

The mixing effectiveness of $\mathcal{F}$ depends on the quantity $\eta = \max |F_{ij}|^2$ [4]. A small η guarantees effective mixing. We discuss this further in section 7.8.

We choose the fixed seed matrix F so as to minimize $\eta = \max |F_{ij}|^2$. The minimal value of η for a unitary F is $1/n$. A normalized DFT matrix achieves the minimum, but applying it requires complex arithmetic. A normalized Hadamard matrix also achieves the minimum and its entries are real. However, Hadamard matrices do not exist for all dimensions, so they are more difficult to use (they require padding). The Discrete Cosine Transform (DCT) and the Discrete Hartley Transform (DHT), which are real, exist for any dimension, and can be applied quickly, but their η value is $2/n$, twice as large as that of the DFT and the Hadamard. All are valid choices. The decision should be based on the implementation cost of computing columns of F and applying $DAD^T$to them versus the value of η.

7.4. Comparing the Quality of Estimators

The easiest way to analyze the quality of trace estimators is to analyze their variance. For any Monte-Carlo estimator R_M we have $\text{Var}(R_M) = \text{Var}(R_1)/M$ so we only need to analyze the variance of a single sample. This type of analysis usually does not reveal much about the estimator, because the variance is usually too large to apply Chebyshev's inequality effectively.

A better way to analyze an estimator is to bound the number of samples required to guarantee that the probability of the relative error exceeding ϵ is at most δ.

Estimator	Variance of one sample	Bound on # samples for an (ϵ, δ)-approx	Random bits per sample
Gaussian	$2\|A\|_F$	$20\epsilon^{-2}\ln(2/\delta)$	infinite; $\Theta(n)$ in floating point
Normalized Rayleigh-quotient	-	$\frac{1}{2}\epsilon^{-2}n^{-2}\operatorname{rank}^2(A)\ln(2/\delta)\kappa_f^2(A)$	-
Hutchinson's	$2\left(\|A\|_F^2 - \sum_{i=1}^n A_{ii}^2\right)$	$6\epsilon^{-2}\ln(2\operatorname{rank}(A)/\delta)$	$\Theta(n)$
Unit Vector	$n\sum_{i=1}^n A_{ii}^2 - \operatorname{trace}^2(A)$	$\frac{1}{2}\epsilon^{-2}\ln(2/\delta)r_D^2(A)$ $r_D(A) = \frac{n\cdot\max_i A_{ii}}{\operatorname{trace}(A)}$	$\Theta(\log n)$
Mixed Unit Vector	-	$8\epsilon^{-2}\ln\left(4n^2/\delta\right)\ln(4/\delta)$	$\Theta(\log n)$

TABLE 1. Summary of results: quality of the estimators under different metrics. The proofs appear in sections 7.5-7.8.

Definition 7.4.1. Let A be a symmetric positive semi-definite matrix. A randomized trace estimator T is an (ϵ, δ)-*approximator* of $\operatorname{trace}(A)$ if

$$\Pr\left(|T - \operatorname{trace}(A)| \leq \epsilon \operatorname{trace}(A)\right) \geq 1 - \delta\,.$$

The third metric that we analyze is the number of random bits used by the algorithm, i.e. the randomness of the algorithm. The trace estimators are highly parallel; each Rayleigh quotient can be computed by a separate processor. If the number of random bits is small, they can be precomputed by a sequential random number generator. If the number is large (e.g., $O(n)$ per Rayleigh quotient), the implementation will need to use a parallel random-number generator. This concern is common to all Monte-Carlo methods.

Table 1 summarizes the results of our analyses. The proofs are in sections 7.5-7.8. The smallest variance is achieved by Hutchinson's estimator, but the Gaussian estimator has a better (ϵ, δ) bound. Unit vector estimators use the fewest random bits, but have an (ϵ, δ) bound that is worse than that of Gaussian and Hutchinson's estimators.

The (ϵ, δ) bounds are not necessarily tight. Our numerical experiments did not show a considerable difference in practice between the Gaussian, Hutchinson and mixed unit vector estimators. See section 7.9.

From a theoretical point of view, the (ϵ, δ) bound for the Gaussian estimator seems good; for fixed ϵ and δ, only $O(1)$ samples are needed. However, the ϵ^{-2} factor in the bound implies that the number of samples may need to scale exponentially with the number of bits of accuracy (the number of samples in the bound scales exponentially with $\log_{10}\epsilon^{-1}$). Therefore, in applications that require only a modest ϵ, say $\epsilon = 0.1$, the Gaussian estimator is good. But in applications that require a small ϵ, even $\epsilon = 10^{-3}$, the number of samples required may be too high.

Are these bounds tight? If they are not, the algorithms themselves may be useful even for small ϵ.

Although we do not have a formal lower-bound, we conjecture that our bound on G_M is almost asymptotically tight. Consider the order n all-ones matrix A. This matrix has a single non-zero eigenvalue n and $n-1$ zero eigenvalues. We see that $\frac{1}{n}z^TAz \sim \chi^2(1)$. Therefor $MG_M/n \sim \chi^2(M)$. This means that G_M has mean n and variance $2n^2/M$. The χ^2 distribution is the sum of independent random variables, so by the central limit theorem it converges to a normal distribution for large M. This convergence to normality

is rather fast, and $M \geq 50$ degrees of freedom is usually considered sufficient for the χ^2 distribution to be "approximately normal" [**46**]. We find that

$$\begin{aligned}\Pr(G_M - n \geq \epsilon n) &\approx \operatorname{erfc}(\epsilon\sqrt{M/2}) \\ &\geq \frac{2}{\sqrt{\pi}} \cdot \frac{\exp(-\epsilon^2 M/2)}{\epsilon\sqrt{M/2} + \sqrt{\epsilon^2 M/2 + 2}}\,,\end{aligned}$$

Let C_δ be the solution to

$$C_\delta\left(\sqrt{\ln(C_\delta/\sqrt{\pi}\delta)} + \sqrt{\ln(C_\delta/\sqrt{\pi}\delta) + 2}\right) = 2\,.$$

If $M < 2\epsilon^{-2}\ln(C_\delta/\pi\delta)$ where we find that

$$\begin{aligned}\Pr(G_M - n \geq \epsilon n) &\geq \frac{2}{\sqrt{\pi}} \cdot \frac{\exp(\ln(\sqrt{\pi}\delta/C_\delta))}{\sqrt{\ln(C_\delta/\sqrt{\pi}\delta)} + \sqrt{\ln(C_\delta/\sqrt{\pi}\delta) + 2}}\,, \\ &= \frac{2}{C_\delta\left(\sqrt{\ln(C_\delta/\sqrt{\pi}\delta)} + \sqrt{\ln(C_\delta/\sqrt{\pi}\delta) + 2}\right)} \cdot \delta \\ &= \delta\,.\end{aligned}$$

The bound is $\Omega(\epsilon^{-2})$ for a fixed δ, but it is not $\Omega(\epsilon^{-2}\ln(1/\delta))$ as $C_\delta \to 0$ if $\delta \to 0$. Nevertheless, this decay is slow and it appears that our bound is almost asymptotically tight.

The main difficulty in turning this argument into a formal proof is the approximation phase $\Pr(G_M - n \geq \epsilon n) \approx \operatorname{erfc}(\epsilon\sqrt{M/2})$. While it is true that the χ^2 distribution converges to the normal distribution, convergence can be very slow. Indeed, the Berry-Esseen Theorem [**96**, § 16.5] guarantees a convergence rate that is proportional only to $M^{-1/2}$. So for a fixed δ there exists an ϵ that is small enough such that the sample size will be so large that the tail bound on normal approximation kicks in. Indeed every Monte-Carlo i.i.d estimator with non-zero finite variance converges to a normal distribution, but the general wisdom on the χ^2 distribution is that it converges very quickly to the normal distribution.

A more direct way to prove a lower bound will be to use some lower bound on the tail of the chi-squared cumulative distribution function. Unfortunately, current bounds ([**220, 132**]) are too complex to provide a useful lower bound, and deriving a simple lower bound is outside the scope of this chapter.

In the next section we present experiments that show that convergence rate (in terms of digits of accuracy) on the all-ones matrix is indeed slow, supporting our conjecture that our bound is almost tight.

7.5. Analysis of the Gaussian Estimator

In this section we analyze the Gaussian estimator. We begin with the variance.

Lemma 7.5.1. *Let A be an $n \times n$ symmetric matrix. The single sample Gaussian estimator G_1 of A is an unbiased estimator of* $\operatorname{trace}(A)$ *i.e.,* $\mathrm{E}(G_1) = \operatorname{trace}(A)$ *and* $\operatorname{Var}(G_1) = 2\|A\|_F^2$.

PROOF. A is symmetric so it can be diagonalized. Let $\Lambda = UAU^T$ be the unitary diagonalization of A (its eigendecomposition), and define $y = Uz$, where $G_1 = z^T Az$. We can write $G_1 = \sum_{i=1}^n \lambda_i y_i^2$ where y_i is the ith entry of y_i. Since U is unitary, the entries of y are i.i.d Gaussian variables, like the entries of z, so $\mathrm{E}(y_i^2) = 1$ and $\mathrm{Var}(y_i^2) = 2$. We find that

$$\mathrm{E}(G_1) = \sum_{i=1}^n \lambda_i \,\mathrm{E}(y_i^2) = \sum_{i=1}^n \lambda_i = \mathrm{trace}(A)\,,$$

$$\mathrm{Var}(G_1) = \sum_{i=1}^n \lambda_i^2 \,\mathrm{Var}(y_i^2) = 2\sum_{i=1}^n \lambda_i^2 = 2\,\|A\|_F^2\ .$$

□

Next, we prove an (ϵ, δ) bound for the Gaussian estimator.

Theorem 7.5.2. *Let A be an $n \times n$ symmetric semidefinite matrix. The Gaussian estimator G_M is an (ϵ, δ)-approximator of* $\mathrm{trace}(A)$ *for $M \geq 20\epsilon^{-2}\ln(2/\delta)$.*

PROOF. A is symmetric so it can be diagonalized. Let $\Lambda = UAU^T$ be the unitary diagonalization of A (its eigendecomposition), and define $y_i = Uz_i$. Since U is unitary, the entries of y_i are i.i.d Gaussian variables. Notice that $G_M = \frac{1}{M}\sum_{i=1}^M \sum_{j=1}^n \lambda_j y_{ij}^2 = \frac{1}{M}\sum_{j=1}^n \lambda_j \sum_{i=1}^M y_{ij}^2$ where y_{ij} is the jth entry of y_i.

We prove the bound using a Chernoff-style argument. y_{ij} is a standard normal random variable so $\sum_{i=1}^M y_{ij}^2$ is χ^2 with M degrees of freedom. Therefore, the moment generating function of $Z = MG_M$ is

$$\begin{aligned} m_Z(t) &= \mathrm{E}(\exp(tZ)) \\ &= \prod_{i=1}^n (1 - 2\lambda_i t)^{-M/2} \\ &= (1 - 2\tau t + h(t))^{-M/2} \end{aligned} \tag{7.5.1}$$

where

$$\tau = \mathrm{trace}(A)$$

and

$$h(t) = \sum_{s=2}^n (-2)^s t^s \sum_{\substack{S \subseteq \Lambda \\ |S| = s}} \prod_{x \in S} x$$

as long as $|\lambda_i t| \leq \frac{1}{2}$ for all i (Λ is the set of A's eigenvalues).

It is easy to see if $\{x_1, \ldots, x_n\}$ is a set of non-negative real numbers, then for all $i = 1, \ldots, n$ we have

$$\sum_{\substack{S \subseteq [n] \\ |S| = i}} \prod_{j \in S} x_j \leq \left(\sum_{i=1}^n x_i\right)^i\ ,$$

where $[n] = \{1, \ldots, n\}$. Therefore, we can bound

$$|h(t)| \leq \sum_{j=2}^{n} (2\tau t)^j \, .$$

Set $t_0 = \epsilon/(4\tau(1+\epsilon/2))$. For all i we have $\lambda_i t_0 \leq \frac{1}{2}$, so (7.5.1) is the correct formula for $m_Z(t_0)$. We now have

$$\begin{aligned} |h(t_0)| &\leq \sum_{j=2}^{n} \left(\frac{\epsilon}{2(1+\epsilon/2)} \right)^j \\ &\leq \frac{\epsilon^2}{4(1+\epsilon/2)^2} \cdot \frac{1}{1 - \frac{\epsilon}{2(1+\epsilon/2)}}. \\ &= \frac{\epsilon^2}{4(1+\epsilon/2)} \end{aligned}$$

Markov's inequality asserts that

$$\begin{aligned} \Pr\left(G_M \geq \tau(1+\epsilon)\right) &= \Pr\left(Z \geq \tau M(1+\epsilon)\right) \, . \\ &\leq m_Z(t_0) \exp(-\tau M(1+\epsilon)t_0) \\ &\leq \left(1 - \epsilon/2\,(1+\epsilon/2) - \epsilon^2/4(1+\epsilon/2)\right)^{-M/2} \cdot \exp(-\frac{M}{2} \cdot \frac{\epsilon}{2} \cdot \frac{1+\epsilon}{1+\epsilon/2}) \\ &= \exp(-\frac{M}{2}(\ln(1 - \epsilon/2(1+\epsilon/2) - \epsilon^2/4(1+\epsilon/2)) + \frac{\epsilon}{2} \cdot \frac{1+\epsilon}{1+\epsilon/2})) \\ &= \exp(-\frac{M}{2}(\ln(1-\epsilon/2) + \frac{\epsilon}{2} \cdot \frac{1+\epsilon}{1+\epsilon/2})) \\ &= \exp\left(-\frac{M}{2}\left(\frac{\epsilon}{2} \cdot \frac{1+\epsilon}{1+\epsilon/2} - \sum_{i=1}^{\infty} \frac{(\epsilon/2)^i}{i})\right)\right) \\ &= \exp\left(-\frac{M}{2}\left(\frac{\epsilon}{2}\left(\frac{1+\epsilon}{1+\epsilon/2} - 1\right) - \frac{\epsilon^2}{8} - \frac{\epsilon^2}{4}\sum_{i=1}^{\infty} \frac{(\epsilon/2)^i}{(i+2)})\right)\right) \\ &\leq \exp\left(-\frac{M}{2}\left(\frac{\epsilon^2}{4} \cdot \frac{1}{1+\epsilon/2} - \frac{\epsilon^2}{8} + \frac{\epsilon^2}{4}\ln(1-\epsilon/2)\right)\right) \\ &= \exp\left(-\frac{M\epsilon^2}{8}\left(\frac{1}{1+\epsilon/2} - \frac{1}{2} + \ln(1-\epsilon/2)\right)\right) \\ &\leq \exp(-M\epsilon^2/20) \end{aligned}$$

for $\epsilon \leq 0.1$. We find that if $M \geq 20\epsilon^{-2}\ln(2/\delta)$ then $\Pr\left(G_M \leq \tau(1+\epsilon)\right) \leq \delta/2$. Using the same technique a lower bound can be shown, and combined with a union-bound we find that $\Pr\left(|G_M - \tau| \leq \tau(1+\epsilon)\right) \leq \delta$. □

In some cases it is possible to prove better bounds, or even the exact trace. For example, we show that using a Gaussian trace estimator we can compute the rank of a projection matrix (i.e., a matrix with only 0 and 1 eigenvalues) using only $O(\mathrm{rank}(A)\log(2/\delta))$ samples (where δ is a probability of failure; there is no dependence on ϵ). Finding the rank of a projection matrix is useful for computing charge densities (in electronic structures calculations) without diagonalization [**29**].

Lemma 7.5.3. *Let $A \in \mathbb{R}^{n\times n}$ be a projection matrix, and let $\delta > 0$ be a failure probability. For $M \geq 24\operatorname{rank}(A)\ln(2/\delta)$, the Gaussian trace estimator G_M of A satisfies*

$$\Pr(\operatorname{round}(G_M) \neq \operatorname{rank}(A)) \leq \delta\,.$$

PROOF. A projection matrix has only 0 and 1 eigenvalue, so the eigenvalue decomposition of A is of the form

$$A = U^T \begin{pmatrix} 1 & & & & & \\ & \ddots & & & & \\ & & 1 & & & \\ & & & 0 & & \\ & & & & \ddots & \\ & & & & & 0 \end{pmatrix} U\,.$$

If we write $y = Uz$ then $z^TAz = \sum_{i=1}^{\operatorname{rank}(A)} y_i^2$. Since U is unitary the entries of y_i are i.i.d Gaussian variables, so z^TAz is χ^2 with $\operatorname{rank}(A)$ degrees of freedom. The χ^2 distribution is additive, so MG_M is also χ^2 but with $M\operatorname{rank}(A)$ degrees of freedom. We now use a known tail-bounds on the χ^2 distribution [**145**]: if $X \sim \chi^2(k)$ then

$$\Pr(|X - k| \leq \epsilon k) \leq 2\exp(-k\epsilon^2/6)\,.$$

By applying this result to MG_M we find that

$$\begin{aligned}\Pr(|G_M - \operatorname{rank}(A)| \geq \operatorname{rank}(A)\epsilon) &= \Pr(|MG_M - M\operatorname{rank}(A)| \geq M\operatorname{rank}(A)\epsilon)\\ &\leq 2\exp(-M\operatorname{rank}(A)\epsilon^2/6)\,.\end{aligned}$$

If we set

$$M \geq 6\operatorname{rank}(A)^{-1}\epsilon^{-2}\ln(2/\delta) \tag{7.5.2}$$

we find that

$$\Pr(|G_M - \operatorname{rank}(A)| \geq \operatorname{rank}(A)\epsilon) \leq \delta\,.$$

If A is a projection matrix, then $\operatorname{trace}(A) = \operatorname{rank}(A)$ is an integer, so if the error is below $\frac{1}{2}$, then $\operatorname{round}(G_M) = \operatorname{rank}(A)$. We set $\epsilon = 1/(2\operatorname{rank}(A))$ and obtain

$$\Pr(\operatorname{round}(G_M) \neq \operatorname{rank}(A)) = \Pr(|G_M - \operatorname{rank}(A)| \geq \operatorname{rank}(A)\epsilon) \leq \delta\,.$$

If we plug ϵ into (7.5.2) we find that we require $M \geq 24\operatorname{rank}(A)\ln(2/\delta)$. □

7.6. General Bound for Normalized Rayleigh quotient Estimators

The sample vectors z in the Gaussian estimator are not normalized, and this can lead to a large z^TAz (but only with a small probability). Normalized estimators are somewhat easier to analyze because each sample is bounded. When A is well conditioned, we get a useful and very general bound.

Theorem 7.6.1. *A normalized Rayleigh estimator R_M is an (ϵ,δ)-approximator of* $\operatorname{trace}(A)$ *for $M \geq \frac{1}{2}\epsilon^{-2}n^{-2}\operatorname{rank}^2(A)\ln(2/\delta)\kappa_f^2(A)$, where $\kappa_f(A)$ is the ratio between the largest and smallest nonzero eigenvalue of A.*

PROOF. Let $0 = \lambda_1 = \cdots = \lambda_k \leq \cdots \leq \lambda_n$ be the eigenvalues of A where $k = n - \operatorname{rank}(A) + 1$, so $\kappa_f(A) = \lambda_n/\lambda_k$. It is easy to see that

$$\begin{aligned}\operatorname{trace}(A) \cdot \kappa_f(A) &= \sum_{i=1}^{n} \lambda_i \cdot \kappa_f(A) \\ &= \sum_{i=k}^{n} \frac{\lambda_i}{\lambda_k} \lambda_n \\ &\geq (n-k+1)\lambda_n \\ &= \operatorname{rank}(A)\lambda_n\end{aligned}$$

therefore for all i

$$0 \leq z_i^T A z_i \leq \lambda_n z_i^T z_i = n\lambda_n \leq \frac{n}{\operatorname{rank}(A)} \operatorname{trace}(A) \cdot \kappa_f(A)\,.$$

According to Hoeffding's inequality for any $t > 0$,

$$\Pr(|R_M - \operatorname{trace}(A)| \geq t) \leq 2\exp\left(-\frac{2M^2 \operatorname{rank}^2(A) t^2}{M \cdot n^2 \operatorname{trace}^2(A) \kappa_f^2(A)}\right).$$

If we set $t = \epsilon \operatorname{trace}(A)$ we find that

$$\Pr(|R_M - \operatorname{trace}(A)| \geq \epsilon \operatorname{trace}(A)) \leq 2\exp\left(-\frac{2M \operatorname{rank}^2(A)\epsilon^2}{n^2 \kappa_f^2(A)}\right).$$

We now set M so that the bound is smaller than δ:

$$\frac{2M \operatorname{rank}^2(A)\epsilon^2}{n^2 \kappa_f^2(A)} \geq \ln\left(\frac{2}{\delta}\right)$$

or

$$M \geq \frac{\ln(2/\delta) \cdot n^2 \kappa_f^2(A)}{2 \operatorname{rank}^2(A)\epsilon^2}\,.$$

□

7.7. Analysis of Hutchinson's Estimator

When A is ill conditioned, the (ϵ, δ) bound in Section 7.6 is weak. We can sharpen it for a specific normalized estimator, that of Hutchinson. However, the bound is still weaker than that of the Gaussian estimator. The bound here is of interest because (1) Hutchinson's estimator is widely used, (2) it uses fewer random bits than the Gaussian estimator, and (3) it requires only additions and subtractions, not multiplications. It is also possible that there is an even stronger bound for Hutchinson's method.

Theorem 7.7.1. *The Hutchinson estimator H_M is an (ϵ, δ)-approximator of* $\operatorname{trace}(A)$ *for* $M \geq 6\epsilon^{-2} \ln(2 \operatorname{rank}(A)/\delta)$.

To prove this theorem we use the following Lemma from [**1**, Lemma 5]:

Lemma 7.7.2. *Let $\alpha \in \mathbb{R}^n$ be an arbitrary unit vector. Define $Q = (\alpha^T z)^2$ where z is a random vector whose entries are i.i.d Rademacher random variables ($\Pr(z_i = \pm 1) = 1/2$). Let $Q_1, \ldots, Q_M$ be M i.i.d copies of Q (different zs but the same α), and define $S = \frac{1}{M}\sum_{i=1}^{M} Q_i$. Then, for any $\epsilon > 0$,*

$$\Pr(|S - 1| \geq \epsilon) \leq 2\exp\left(-\frac{M}{2}\left(\frac{\epsilon^2}{2} - \frac{\epsilon^3}{3}\right)\right).$$

Proof. *(of Theorem 7.7.1).* A is symmetric and semidefinite so it can be diagonalized. Let $\lambda_1, \ldots, \lambda_n$ be the eigenvalues of A and assume without loss of generality that the non-zero eigenvalues are $\lambda_1, \ldots, \lambda_{\mathrm{rank}(A)}$. Let $\Lambda = UAU^T$ be the unitary diagonalization of A (its eigendecomposition), and define $y_i = U^T z_i$. Notice that $H_M = \frac{1}{M}\sum_{i=1}^{M}\sum_{j=1}^{n}\lambda_j y_{ij}^2 = \sum_{j=1}^{n}\lambda_j \frac{1}{M}\sum_{i=1}^{M} y_{ij}^2$ where y_{ij} is the jth entry of y_i. The rows U_i^T of U^T are unit vectors so $S = \frac{1}{M}\sum_{i=1}^{M}\left(U_j^T z_i\right)^2$ satisfies the conditions of Lemma 7.7.2. But we also have $S = \frac{1}{M}\sum_{i=1}^{M} y_{ij}^2$, so

$$\Pr\left(\left|\frac{1}{M}\sum_{i=1}^{M} y_{ij}^2 - 1\right| \geq \epsilon\right) \leq 2\exp\left(-\frac{M}{2}\left(\frac{\epsilon^2}{2} - \frac{\epsilon^3}{3}\right)\right).$$

If $M \geq 6\epsilon^{-2}\ln(2\,\mathrm{rank}(A)/\delta)$ this implies that

$$\Pr\left(\left|\frac{1}{M}\sum_{i=1}^{M} y_{ij}^2 - 1\right| \geq \epsilon\right) \leq \frac{\delta}{\mathrm{rank}(A)}.$$

This bound holds for each specific j. Using the union-bound, we conclude that the probability that the error is larger than ϵ for some $j = 1, \ldots, \mathrm{rank}(A)$ is at most δ. Hence, the probability that the error is smaller than ϵ for all $j = 1, \ldots, \mathrm{rank}(A)$ is at least $1 - \delta$. So with probability $1 - \delta$ we also have

$$\begin{aligned}
|H_M - \mathrm{trace}(A)| &= \left|\sum_{j=1}^{n}\lambda_j \frac{1}{M}\sum_{i=1}^{M} y_{ij}^2 - \sum_{i=1}^{n}\lambda_i\right| \\
&= \left|\sum_{j=1}^{\mathrm{rank}(A)}\lambda_j\left(\frac{1}{M}\sum_{i=1}^{M} y_{ij}^2 - 1\right)\right| \\
&\leq \sum_{j=1}^{\mathrm{rank}(A)}\lambda_j\left|\frac{1}{M}\sum_{i=1}^{M} y_{ij}^2 - 1\right| \\
&\leq \epsilon\sum_{j=1}^{\mathrm{rank}(A)}\lambda_j \\
&= \epsilon\,\mathrm{trace}(A).
\end{aligned}$$

□

The bound is larger than the bound for the Gaussian estimator by a $\ln(\mathrm{rank}(A))$ factor. The main difficulty here is that, unlike the Gaussian estimator, the Hutchinson's estimator cannot be written as a weighted sum of i.i.d random variables. This forces us to use a union bound instead of using a global analysis. Nevertheless, given the better

variance term of Hutchinson's estimator we conjecture that this $\ln(\text{rank}(A))$ factor is redundant. In fact, there are some matrix classes for which Hutchinson's estimator is clearly better than the Gaussian estimator. For example, on diagonal or nearly diagonal matrices the Hutchinson's estimator will converge very fast, which is not true for the Gaussian estimator. Another interesting example is the all-ones matrix for which the bound for the Hutchinson estimator is the same as the bound for the Gaussian estimator (it is possible to show that for the all-ones matrix the Gaussian estimator is an (ϵ, δ)-approximator for $M \geq 6\epsilon^{-2}\ln(2/\delta)$).

7.8. Reducing Randomness: Analyzing Unit Vector Estimators

This section analyzes two unit vector estimators: the unit vector estimator and the mixed unit vector estimator. These estimators' main advantage is in restricting the sample space to n vectors. Thus, only $\lceil \log_2 n \rceil$ random bits are required per sample. This allows the samples to be generated in advance. We begin by analyzing the variance.

Lemma 7.8.1. *Let A be an $n \times n$ symmetric matrix. The single sample unit vector estimator U_1 of A is an unbiased estimator of* $\text{trace}(A)$ *i.e.,* $\text{E}(U_1) = \text{trace}(A)$ *and* $\text{Var}(U_1) = n\sum_{i=1}^{n} A_{ii}^2 - \text{trace}^2(A)$ *..*

PROOF. Let $U_1 = nz^T Az$. Because z is an identity vector $z^T Az$ just samples values from the diagonal. Every diagonal value is sampled with equal probability, so $\text{E}(z^T Az) = \text{trace}(A)/n$, from which $\text{E}(nz^T Az) = \text{trace}(A)$ follows immediately.

As for variance the following equality holds

$$\begin{aligned} \text{Var}(nz^T Az) &= \text{E}((nz^T Az)^2) - (\text{E}(nz^T Az))^2 \\ &= n^2\,\text{E}((z^T Az)^2) - \text{trace}^2(A) \end{aligned}$$

The random variable $(z^T Az)^2$ samples the square of the diagonal values of A so $\text{E}((z^T Az)^2) = \sum_{i=1}^{n} A_{ii}^2/n$ and the equality follows. □

We now turn to the more interesting analysis of the number of samples that guarantee an (ϵ, δ)-approximator. This quantity depends on the ratio between the largest possible estimate (when estimating the maximal diagonal value) and the trace.

Theorem 7.8.2. *The unit vector estimator U_M is an (ϵ, δ)-approximator of* $\text{trace}(A)$ *for $M \geq \frac{1}{2}\epsilon^{-2}\ln(2/\delta)r_D^2(A)$ where $r_D(A) = \frac{n \cdot \max_i A_{ii}}{\text{trace}(A)}$.*

PROOF. The unit vector estimator samples values from the diagonal and multiplies them by n, so a single samples takes values in the range $[0, n \cdot \max_i A_{ii}]$. According to Hoeffding's inequality

$$\Pr(|U_M - \text{trace}(A)| \geq t) \leq 2\exp\left(-\frac{2M^2t^2}{Mn^2 \cdot (\max_i A_{ii})^2}\right).$$

If we set $t = \epsilon\,\text{trace}(A)$ we find that

$$\Pr(|U_M - \text{trace}(A)| \geq \epsilon\,\text{trace}(A)) \leq 2\exp\left(-\frac{2M\epsilon^2}{r_D^2(A)}\right).$$

We now set M so that the bound is smaller than δ:

$$\frac{2M\epsilon^2}{r_D^2(A)} \geq \ln\left(\frac{2}{\delta}\right)$$

or

$$M \geq \frac{\ln(2/\delta) \cdot r_D^2(A)}{2\epsilon^2} .$$

□

We now analyze the mixed unit vector estimator. The unit vector estimator relies on the the mixing matrix $\mathcal{F}$. The analysis is based on the following lemma. The proof follows the beginning of the proof of Lemma 2.1 in [4] very closely, and we include it here mainly for completeness.

Lemma 7.8.3. *Let U be an $n \times m$ matrix with orthonormal columns, and let $\mathcal{F} = FD$ be a random mixing matrix. With probability of at least $1 - \delta$ ($\delta > 0$) we have for all i and j*

$$\left|(\mathcal{F}U)_{ij}\right| \leq \sqrt{2\eta \ln\left(\frac{2mn}{\delta}\right)} ,$$

where $\eta = \max |F_{ij}|^2$.

PROOF. We shall first bound $|(\mathcal{F}U)_{ij}|$ by bounding $\|\mathcal{F}x\|_\infty$ for any vector x such that $\|x\|_2 = 1$. Denote $u = \mathcal{F}x$ and let $u = \left[\begin{array}{ccc} u_1 & u_2 & \cdots \end{array}\right]^T$. We can write

$$u_1 = \sum_{i=1}^{m} b_i f_i x_i$$

where $b_i = \pm 1$ with equal probability, and $0 \leq f_i \leq \sqrt{\eta}$. Then,

$$\mathrm{E}(e^{t\eta^{-1}u_1}) = \prod_{i=1}^{m} \mathrm{E}(e^{t\eta^{-1}b_i f_i x_i}) = \prod_{i=1}^{m} \cosh(t\eta^{-1} f_i x_i) \leq \prod_{i=1}^{m} \cosh(t\eta^{-1/2} x_i) \leq e^{t^2\eta^{-1}\|x\|_2}$$

so by Markov's inequality (Chernoff type argument)

$$\begin{aligned} \Pr(|u_1| \geq s) &= \Pr(u_1 \geq s) + \Pr(u_1 \leq -s) \\ &= \Pr(e^{s\eta^{-1}u_1} \geq e^{s^2\eta^{-1}}) + \Pr(e^{-s\eta^{-1}u_1} \geq e^{s^2\eta^{-1}}) \\ &\leq 2\,\mathrm{E}(e^{s\eta^{-1}u_1})/e^{s^2\eta^{-1}} \\ &\leq 2e^{s^2\eta^{-1}\|x\|_2/2 - s^2\eta^{-1}} \\ &= 2e^{-s^2\eta^{-1}/2} \end{aligned}$$

If we set $s = \sqrt{2\eta \ln\left(\frac{2mn}{\delta}\right)}$ then $\Pr(|u_1| \geq s) \leq \delta/nm$. If we put in x the columns of U we see that we found a bound over all coordinates of $\mathcal{F}U$. By union bound of all nm coordinates we see that $|(\mathcal{F}U)_{ij}| \leq \sqrt{2\eta \ln\left(\frac{2mn}{\delta}\right)}$. □

The mixing matrix prevents entries from an orthonormal matrix to be too large. When applied from both sides to a symmetric positive semidefinite matrix it prevents the diagonal elements from being too big, i.e. $r_D(\mathcal{F}A\mathcal{F}^T)$ is not too big.

Theorem 7.8.4. *The mixed unit vector estimator T_M is an (ϵ, δ)-approximator of* trace(A) *for $M \geq 2n^2\eta^2\epsilon^{-2} \ln(4/\delta) \ln^2(4n^2/\delta)$.*

PROOF. A is symmetric so it can be diagonalized. Let $\Lambda = U^T AU$ be the unitary diagonalization of A (its eigendecomposition), and let $V = \mathcal{F}U$. It is easy to see that

$$\left(\mathcal{F}A\mathcal{F}^T\right)_{jj} = \sum_{k=1}^{n} \lambda_k V_{jk}^2 \,.$$

According to Lemma 7.8.3, with probability $1 - \delta/2$ we have

$$V_{jk}^2 = \left|\left(\mathcal{F}U\right)_{jk}\right|^2 \leq 2\eta \ln\left(\frac{2n^2}{\delta/2}\right) = 2\eta \ln\left(\frac{4n^2}{\delta}\right) \,. \tag{7.8.1}$$

The eigenvalues λ_i are non-negative, so we conclude that with probability $1 - \delta/2$ for all j,

$$\begin{aligned} 0 \leq \left(\mathcal{F}A\mathcal{F}^T\right)_{jj} &\leq 2\eta \ln\left(\frac{4n^2}{\delta}\right) \sum_{j=1}^{n} \lambda_j \\ &= 2\eta \ln\left(\frac{4n^2}{\delta}\right) \operatorname{trace}(A) \,. \end{aligned}$$

We find that

$$r_D(\mathcal{F}A\mathcal{F}^T) \leq 2n\eta \ln\left(\frac{4n^2}{\delta}\right) \,.$$

Therefore, according to Theorem 7.8.2for $M \geq 2n^2\eta^2\epsilon^{-2} \ln(4/\delta) \ln^2(4n^2/\delta)$ we have $\Pr\left(|T_M - \operatorname{trace}(A)| > \epsilon \operatorname{trace}(A)\right) \leq 1 - \delta/2$.

There can be failures of two kinds: with probability at most $\delta/2$ the bound on the diagonal elements of the mixed matrix may fail to hold, and even if it holds, with probability $\delta/2$ the ϵ bound on the estimation error may fail to hold. We conclude that with probability $1 - \delta$ the error bound does hold. □

Remark 7.8.5. For Fourier-type matrices, such as DFT and DCT, $\eta = \Theta(1/n)$, so the lower bound on M becomes simpler,

$$M \geq C \frac{\ln^2\left(4n^2/\delta\right) \ln(4/\delta)}{\epsilon^2} \,,$$

for some small C (8 for the case of DCT, 2 for DFT).

7.9. Experiments

We present the results of several computational experiments that compare the different estimators, and clarify the actual convergence rate.

Figure 7.9.1 shows the convergence of the various estimators on a matrix of order $n = 100,000$ whose elements are all 1. We have used this matrix as an example of the matrix with the largest variance possible for Hutchinson's and Gaussian estimator. The graphs show that all methods converge quite slowly. There is no significant difference in the convergence behavior of all three methods, although we presented different bounds. The graph also supports our conjecture that our bounds are almost tight, and that the cost is exponential in the number of required accuracy digits.

Figure 7.9.2 clarifies the convergence behavior of the estimators. The graph on the left shows the convergence all the way up to n iterations, with two variants of the mixed

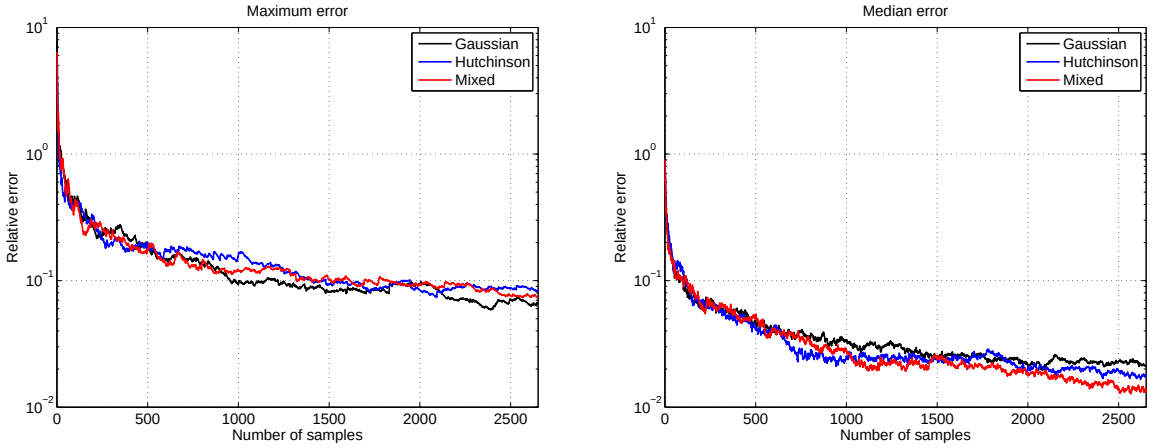


FIGURE 7.9.1. Convergence of the estimators on a matrix of order 100,000 whose elements are all 1. The graph on the left shows the maximum error during 100 runs of the algorithm, and the graph on the right the median of the 100 runs.

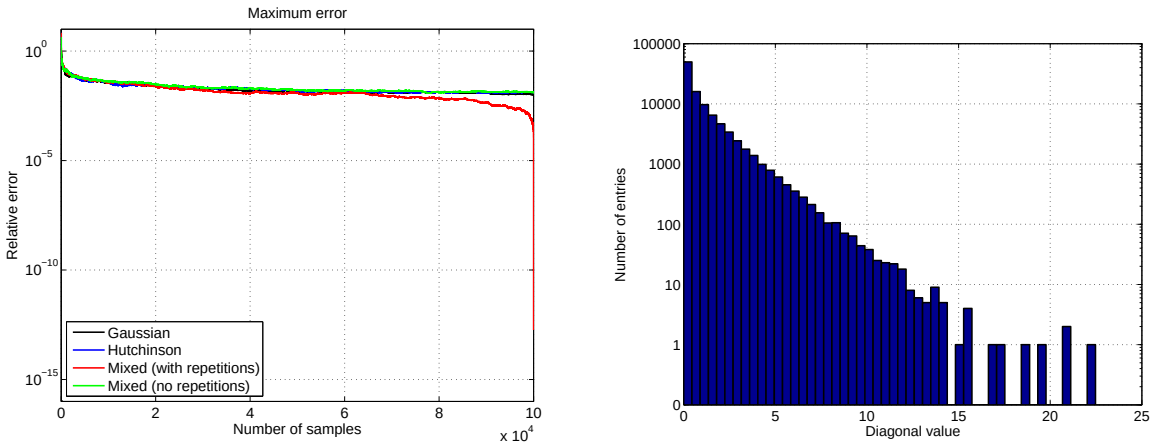


FIGURE 7.9.2. Details to clarify the behavior of the methods. The experiment is similar to the one in Figure 7.9.1. The graph on the left shows convergence all the way to n iterations, and the histogram on the right shows the distribution of diagonal values (relevant for the estimator presented in section 7.8).

estimator: with and without repetitions. Convergence stagnates and the error nears machine ϵ only very close to iteration n and only when sampling without repetitions. If we sample without repetitions, after we sample all the sample space we are guaranteed to have the exact trace (this is not possible for the Gaussian estimator and Hutchinson's estimator, but also not practical in our method). The histogram on the right show that in spite of the mixing that $\mathcal{F}$ performs, the diagonal elements of the mixed matrix $\mathcal{F}A\mathcal{F}^T$ are still highly skewed. In other words, there are some diagonal values that are important to sample; until they are sampled, the error remains large.

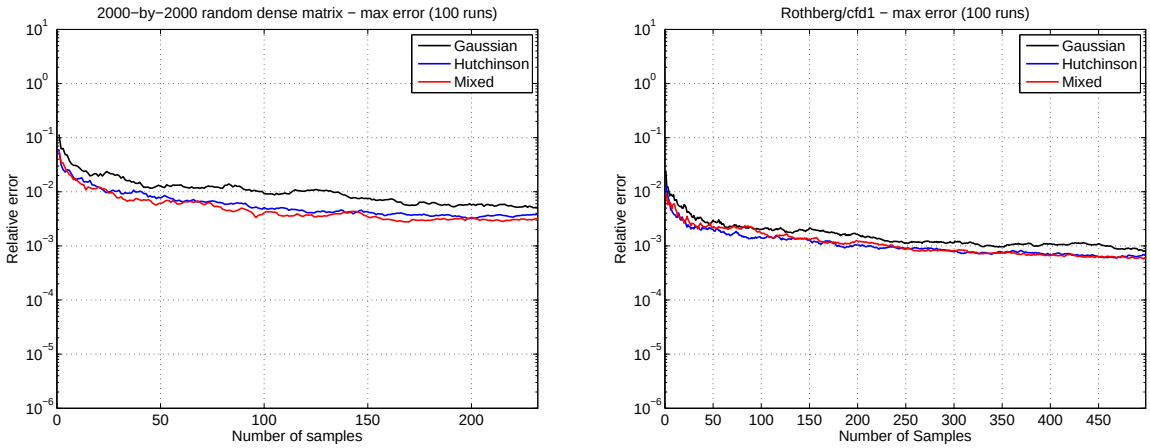


FIGURE 7.9.3. Convergence on two more matrices: a random matrix of order 2000 (left) and a sparse matrix of order 70,656.

Figure 7.9.3 shows that on other classes of matrices, the methods reach a smaller error before they stagnate. On a random dense matrix, the methods converge quickly to an error smaller than 10^{-2}, but then stagnate. On a sparse matrix from the University of Florida matrix collection, the methods reach an error of about 10^{-3} and then stagnate. There is again little difference between the convergence rates of the three methods, although it seems that Gaussian estimator is a little less accurate then the other two estimators.

7.10. Conclusions

In terms of the (ϵ, δ) bounds, the Gaussian estimator requires the smallest number of samples. The convergence bound for Hutchinson's estimator is the runner up: it requires more iterations than the Gaussian, but fewer than the mixed unit vector estimator.

In terms of the number of random bits that these estimators require, the ranking is the exact opposite: the Gaussian estimator requires the most bits, followed by Hutchinson's estimator, and the mixed unit vector estimator requires the least.

Convergence to a small error is slow, both in practice and in terms of the bounds. The ϵ^{-2} factor in all the bounds imply that the number of samples required to get close to, say, machine epsilon, is huge. The estimators quickly give a crude estimate of the trace (correct to within 0.1 or 0.01, say), but they require a huge number of samples to obtain a very accurate estimate.

The ϵ^{-2} factor in the bound is common to many Monte-Carlo algorithms in numerical linear algebra. When the Monte-Carlo method is used as an inexact solver within the context of an iterative solver, the overall algorithm can be both fast and accurate (chapter 8 and [**18**]). We are not aware of a suitable iterative algorithm for trace computations.

CHAPTER 8

Engineering a Random-Sampling Numerical Linear Algebra Algorithm

8.1. Introduction

Randomization is arguably the most exciting and innovative idea to have hit linear algebra in a long time. Several such algorithms have been proposed and explored in the past decade (see, e.g, [**186, 89, 80, 179, 162, 88, 178, 71, 45**] and the references therein). Some forms of randomization have been used for decades in linear algebra. For example, the starting vectors in Lanczos algorithms are always random. But recent research led to new uses of randomization: random mixing and random sampling, which can be combined to form random projections. These ideas have been explored theoretically and have found use in some specialized applications (e.g., data mining [**153, 45**]), but they have had little influence so far on mainstream numerical linear algebra.

This chapter[1] answers a simple question: can these new techniques beat state-of-the-art numerical linear algebra libraries *in practice*?

Through careful engineering of a new least-squares solver, which we call Blendenpik, and through extensive analysis and experimentation, we have been able to answer this question: yes.

Blendenpik beats LAPACK's direct dense least-squares solver by a large margin on essentially any dense tall matrix. Blendenpik is slower than LAPACK on tiny matrices, nearly square ones, and on some sparse matrices. But on a huge range of matrices of reasonable sizes, the answer is an unqualified yes. Figure 8.1.1 shows a preview of our experimental results. On large matrices, Blendenpik is about four times faster than LAPACK. We believe that these results show the potential of random-sampling algorithms, and suggest that random-projection algorithms should be considered for incorporation into future versions of LAPACK.

8.2. Overview of the algorithm

Let $x_{\text{opt}} = \arg\min_x \|Ax - b\|_2$ be a large highly overdetermined system, with $A \in \mathbb{R}^{m \times n}$ and $b \in \mathbb{R}^m$. Can we sample a small set of rows, $\mathcal{R}$, and use only those rows to find an approximate solution? That is, is the solution $x_{\mathcal{R}} = \arg\min_x \|A_{\mathcal{R},*}x - b_{\mathcal{R}}\|_2$ a good approximation of x_{opt}? The following simple experiment in MATLAB [**156**] illustrates that for random matrices $x_{\mathcal{R}}$ is indeed a good approximation in some sense as long as $\mathcal{R}$ is big enough:

[1]The results in this chapter also appear in a paper co-authored with Petar Maymounkov and Sivan Toledo which was published in the *SIAM Journal on Scientific Computing* [**18**]. This chapter also contains several proofs that were omitted from the paper, as long with a numerical experiment that was omitted from the paper (rightmost graph of Figure 8.5.8).

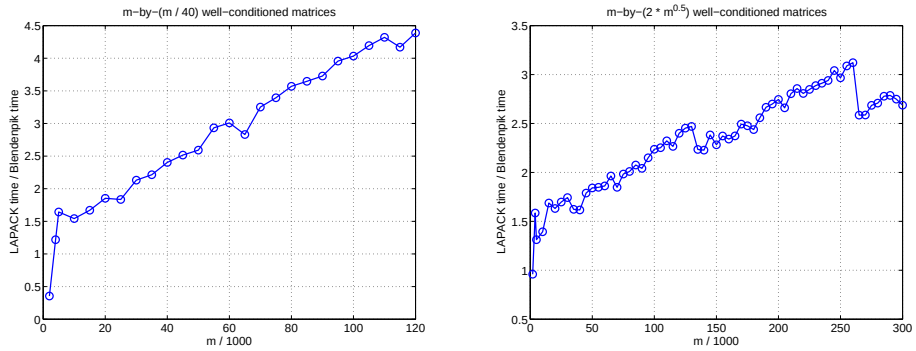


FIGURE 8.1.1. Comparison between LAPACK and the new solver for increasingly larger matrices. Graphs show the ratio of LAPACK's running time to Blendenpik's running time on random matrices with two kinds of aspect ratios.

```
>> rand('state', 2378)
>> randn('state', 23984)
>> m = 20000; n = 100;
>> A = rand(m, n); b = rand(m, 1);
>> [U, S, V] = svd(A, 0);
>> S = diag(linspace(1, 10^6, 100));
>> A = U * S * V';
>> sampled_rows = find(rand(m, 1) < 10 * n * log(n) / m);
>> A1 = A(sampled_rows, :); b1 = b(sampled_rows);
>> x = A \ b;
>> x1 = A1 \ b1;
>> norm(A * x1 - b) / norm(A * x - b)
ans =
    1.0084
```

The norm of residual is within 1.01 of the optimal residual. In general, under certain conditions on A, a sample of $\Omega(n \log(m) \log(n \log(m)))$ rows guarantee a residual that is within a factor of $1 + \epsilon$ of the optimal [**89**].

There are two problems with this approach. First, the analysis in [**89**] bounds the relative error in residual norm, that is

$$\|Ax_{\mathcal{R}} - b\|_2 / \|Ax_{\text{opt}} - b\|_2 \leq 1 + \epsilon$$

where x_{opt} is the true solution, and $x_{\mathcal{R}}$ is the computed solution. Drineas et al. show that this implies a bound on the forward error,

$$\frac{\|x_{\text{opt}} - x_{\mathcal{R}}\|_2}{\|x_{\text{opt}}\|_2} \leq \tan(\theta)\kappa(A)\sqrt{\epsilon}$$

where $\theta = \cos^{-1}(\|Ax_{\text{opt}}\|_2 / \|b\|_2)$. While such an analysis might be useful in some fields, it is difficult to relate it to standard stability analyses in numerical linear algebra. The standard stability analysis of least-squares algorithms is done in terms of *backward error*:

an approximate solution $\tilde{x}$ is shown to be the exact solution of a perturbed system

$$\tilde{x} = \arg\min_x \|(A + \delta A)x - b\|_2$$

where $\|\delta A\| \leq \tilde{\epsilon}\|A\|$. This implies a bound on the forward error

$$\frac{\|x_{\text{opt}} - \tilde{x}\|_2}{\|x_{\text{opt}}\|_2} \leq \left(\kappa(A) + \frac{\kappa(A)^2 \tan\theta}{\eta}\right)\tilde{\epsilon}$$

where $\eta = \|A\|_2\|x\|_2/\|Ax\|_2$. The two forward error bounds are not comparable in an obvious way. Moreover, the $\sqrt{\epsilon}$ appears to make it difficult to prove small forward error bounds in well conditioned cases.

Second, running time depends on ϵ^{-1}. The backward stability requirements (e.g., value of ϵ) of linear-algebra software may be a constant, but it is a tiny constant. So to achieve the required ϵ the constants in the asymptotic bounds in [**89**] might be too large.

Rokhlin and Tygert [**179**] use a different approach. They use the R factor of the sampled rows as a preconditioner in a Krylov-subspace method like LSQR [**166**]:

```
>> [Q, R] = qr(A1, 0);
>> x1 = lsqr(A, b, eps, 100, R);
lsqr converged at iteration 17 to a solution with relative residual 0.5
```

A uniformly sample of the rows is not always a good strategy. If A has a column j that is zero except for $A_{ij} \neq 0$, any subset of rows that excludes row i is rank deficient. If $m \gg n$, the algorithm needs to sample close to m rows in order to guarantee a high probability that row i is in the subset. If the row sample is too small, the preconditioner is rank deficient and LSQR fails.

```
>> A(1:end-1, end) = 0;
>> A1 = A(sampled_rows, :);
>> [Q, R] = qr(A1, 0);
>> x1 = lsqr(A, b, eps, 100, R);
Warning: Matrix is singular to working precision.
> In sparfun\private\iterapp at 33
  In lsqr at 217 In overview at 35
lsqr stopped at iteration 0 without converging to the desired
tolerance 2.2e-016
because the system involving the preconditioner was ill conditioned.
The iterate returned (number 0) has relative residual 1
```

Uniform random sampling works well only when the *coherence* of the matrix is small, which is equal to maximum norm of a row in Q, where Q forms an orthonormal basis for the column space of A (e.g., the leftmost factor in a reduced QR or singular-value decomposition; a formal definition of coherence appears in Section 8.3). The coherence of the matrix is between n/m and 1. The lower it is the better uniform sampling works.

```
>> [Q, R] = qr(A, 0);
>> coherence = max(sum(Q .^ 2, 2))
coherence =
    1.0000
```

The coherence of our matrix, after the insertion of zeros into column 1, is the worst possible.

The coherence of matrices with random independent uniform entries tends to be small, but as we have seen, other matrices have high coherence. We can use a randomized row-mixing preprocessing phase to reduce the coherence [**89, 179**].

```
>> D = spdiags(sign(randn(m, 1)), 0, m, m);
>> B = dct(D * A); B(1, :) = B(1, :) / sqrt(2);
>> [Q, R] = qr(B, 0);
>> coherence = max(sum(Q .^ 2, 2))
coherence =
    0.0083
```

First, we randomly multiply each row by $+1$ or -1, and then apply a discrete cosine transform (DCT) to each column. The first row is divided by $\sqrt{2}$ to make the transformation orthogonal. With high probability the coherence of B is small. In the example above, it is less than twice the minimal coherence (0.005). There are many ways to mix rows to reduce the coherence. We discuss other methods in Section 8.3.2.

With high probability, a uniform sample $B1$ of the rows of the row-mixed matrix B makes a good preconditioner. In the code below, we use the R factor of the sample, to allow LSQR to apply the preconditioner efficiently.

```
>> B1 = B(sampled_rows, :);
>> [Q, R] = qr(B1, 0);
>> x1 = lsqr(A, b, eps, 100, R);
lsqr converged at iteration 15 to a solution with relative residual 1
```

8.3. Theory

This section explains the theory behind the algorithms that this chapter investigates. The ideas themselves are not new; they have been proposed in several recent papers [**89, 179, 162**]. We do present some simple generalizations and improvements to existing results.

8.3.1. Uniform sampling preconditioners. The quality of uniform sampling preconditioners depend on how much the solution depends on specific rows. For example, if the sample is rank deficient unless row i is in it, then the size of a uniform sample must be too large to be effective. *Coherence* [**54**] is the key concept for measuring the dependence of the solution on specific rows.

Definition 8.3.1. Let A be an $m \times n$ full rank matrix and let U be an $m \times n$ matrix whose columns form an orthonormal basis for the column space of A. The *coherence* of A is defined as

$$\mu(A) = \max \|U_{i,*}\|_2^2 .$$

The coherence of a matrix is always smaller than 1 and bigger than n/m. If a row contains the only non-zero in one of the columns of A then the coherence of the matrix is 1. Coherence does not relate in any way to the condition number of A.

Uniform random sampling yields a good preconditioner on incoherent matrices (matrices with small coherence). For example, if $\mu(A) = n/m$, then a sample of $\Theta(n \log n)$ rows is sufficient to obtain a good preconditioner. The following theorem describes a relationship between the coherence, the sample size, and the condition number of the preconditioned system.

Theorem 8.3.2. *Let A be an $m \times n$ full-rank matrix, and let $\mathcal{S}$ be a random sampling operator that samples $r \geq n$ rows from A uniformly, such that $\mathcal{S}A$ is full rank. Let $\tau = C\sqrt{m\mu(A)\log(r)/r}$ where C is some constant defined in the proof. Assume that $\delta^{-1}\tau < 1$. With probability of at least $1-\delta$ the sampled matrix $\mathcal{S}A$ is full rank, and if $\mathcal{S}A = UR$ is a reduced QR factorization of $\mathcal{S}A$, we have*

$$\kappa(AR^{-1}) \leq \sqrt{\frac{1+\delta^{-1}\tau}{1-\delta^{-1}\tau}}\,.$$

To prove Theorem 8.3.2 we need the following simplified version of Lemma 4 in [**89**].

Lemma 8.3.3. *Let U be an $m \times n$ matrix whose columns are orthonormal, and let $\mathcal{S}$ be a random sampling operator that samples $r \leq m$ rows from U uniformly. Then*

$$\mathrm{E}\left(\left\|I_{n\times n} - (m/r)U^T\mathcal{S}^T\mathcal{S}U\right\|\right) \leq C\sqrt{m\mu(U)\log(r)/r}$$

for some constant C.

Proof. (of Theorem 8.3.2) First notice that if $\mathcal{S}U$ is full-rank then so is $\mathcal{S}A$ and $\kappa(\mathcal{S}U) = \kappa(AR^{-1})$. Indeed,

$$\begin{aligned}
\kappa(AR^{-1}) &= \sqrt{\kappa(A^TA, (\mathcal{S}A)^T(\mathcal{S}A))} \\
&= \sqrt{\kappa((\mathcal{S}A)^T(\mathcal{S}A), A^TA)} \\
&= \kappa((\mathcal{S}A)R^{-1}) \\
&= \kappa(\mathcal{S}U)
\end{aligned}$$

where $\kappa(X,Y)$ is the ratio between the largest and smallest determined generalized eigenvalues of SPD matrices X and Y.

According to Lemma 8.3.3

$$\mathrm{E}\left(\left\|I_{n\times n} - (m/r)U^T\mathcal{S}^T\mathcal{S}U\right\|\right) \leq \tau\,,$$

therefore according to Markov's inequality

$$\Pr\left(\left\|I_{n\times n} - (m/r)U^T\mathcal{S}^T\mathcal{S}U\right\| \geq \delta^{-1}\tau\right) \leq \delta\,.$$

With probability $1-\delta$ we have $\left\|I_{n\times n} - (m/r)U^T\mathcal{S}^T\mathcal{S}U\right\| < \delta^{-1}\tau < 1$, so $\mathcal{S}U$ is full rank. Every eigenvalue λ of $(m/r)U^T\mathcal{S}^T\mathcal{S}U$ is the Rayleigh quotient of some vector $x \neq 0$ so

$$\begin{aligned}
\lambda &= \frac{(m/r)x^Tx}{x^Tx} \\
&= \frac{x^Tx + x^T((m/r)U^T\mathcal{S}^T\mathcal{S}U - I_{n\times n})x}{x^Tx} \\
&= 1+\eta
\end{aligned}$$

where η is a Rayleigh quotient of $I_{n\times n} - (m/r)U^T\mathcal{S}^T\mathcal{S}U$. This matrix is symmetric its singular values are the absolute eigenvalues. This implies that $|\eta| < \delta^{-1}\tau$. So, all eigenvalues of $(m/r)U^T\mathcal{S}^T\mathcal{S}U$ must be between $1-\delta^{-1}\tau$ and $1+\delta^{-1}\tau$, and we have

$$\kappa(\mathcal{S}U) \leq \sqrt{\frac{1+\delta^{-1}\tau}{1-\delta^{-1}\tau}}\,.$$

□

Remark 8.3.4. Notice that the condition number of the original matrix A does not affect the bound on the condition number of the preconditioned matrix.

Remark 8.3.5. Theorem 8.3.2 describes a relationship between sample size (r), probability of failure (δ) and the condition number of the preconditioned system. With a small sample, the probability of obtaining a high condition number is high. A high condition number may lead to a large number of iterations in LSQR, but the number of iterations may also be small: the convergence of LSQR depends on the distribution of the singular values of AR^{-1}, not just on the extreme singular values. In fact, chapter 3 use the fact that a few very large or very small singular values do not effect convergence much.

If the coherence is high, uniform sampling produces poor preconditioners. One alternative is to use non-uniform sampling. Let $A = UR$ be a reduced QR factorization of A. Drineas et al. [**87**] suggest sampling row i with probability $p_i = \|U_i\|_2^2/m$, where U_i is row i of U. Computing these probabilities requires too much work (a QR factorization of A), so to make this approach practical probabilities should be, somehow, approximated; to the best of our knowledge no efficient approximation algorithm has been developed yet. Therefore, in the next subsection we turn to a simpler approach, the one used by our solver, which is based on mixing rows.

8.3.2. Row mixing. Theorem 8.3.2 implies that even if there are important rows, that is even if coherence is high, if we sample enough rows then with high probability the preconditioner is a good preconditioner. The higher $\mu(A)$ is the more rows should be sampled. This poses two problems. First, finding $\mu(A)$ is computationally hard. Second, if $\mu(A)$ is high too, then many rows need to be sampled. Indeed, if $\mu(A) = 1$ (the worst) then as many as $\mathcal{O}(m \log m)$ rows need to be sampled in order to get a bound on the condition number using Theorem 8.3.2. When $\mu(A) = 1$, there is a row in the matrix that must be included in the sample for R to be full rank. We do not know which row it is, so no row can be excluded from the sample; this is not useful. If $\mu(A) = n/m$ (minimal coherence), on the other hand, then only $\Theta(n \log n)$ rows need to be sampled to get $\kappa = \mathcal{O}(1)$ with high probability.

In general, we cannot guarantee a bound on $\mu(A)$ in advance. The solution is to perform a *preprocessing* step in which rows are mixed so that their importance is nearly equal. The crucial observation is that a unitary transformation preserves condition number, but changes coherence. If $\mathcal{F}$ is a unitary transformation and R is a preconditioner $\mathcal{F}A$, then R is an equally good preconditioner for A because the singular values of AR^{-1} and $\mathcal{F}AR^{-1}$ are the same. But $\mu(A)$ and $\mu(\mathcal{F}A)$ are not necessarily the same; if we select $\mathcal{F}$ so that $\mu(\mathcal{F}A)$ is small, then we can construct a good preconditioner by uniformly random sampling the rows of $\mathcal{F}A$.

Any fixed unitary transformation $\mathcal{F}$ leads to a high $\mu(\mathcal{F}A)$ on some A's, so we use a random unitary transformation. We use the *random mixing matrices* which we introduced in chapter 7. Recall that a random mixing matrix $\mathcal{F}$ from a product of a fixed *seed unitary transformation* F and a random diagonal matrix D with ± 1 diagonal entries. The diagonal entries of D are random, unbiased, independent random variables. The following lemma shows that with high probability, the coherence of $\mathcal{F}A$ is small, as long as the maximum value in F is not too large.

Lemma 8.3.6. *Let A be an $m \times n$ full rank matrix where $m \geq n$. Let F be an $m \times m$ unitary matrix, let D be a diagonal matrix whose diagonal entries are i.i.d Rademacher random variables ($\Pr(D_{ii} = \pm 1) = 1/2$), and let $\mathcal{F} = FD$. With probability of at least 0.95 we have*

$$\mu(\mathcal{F}A) \leq Cn\eta \log m \ ,$$

where $\eta = \max |F_{ij}|^2$ and some constant C.

Remark 8.3.7. A must be full rank for the μ to be well defined. The theorem can be generalized to success guarantees other than 0.95. A higher probability leads to a higher constant C.

Proof. Let $A = UR$ be a reduced QR factorization of A. We have $\mathcal{F}A = (\mathcal{F}U)R$ which is a reduced QR factorization of $\mathcal{F}A$, so

$$\mu(\mathcal{F}A) = \max \|(\mathcal{F}U)_{i,*}\|_2^2 \ .$$

According to lemma 7.8.3 (from chapter 7) with probability of at least $1 - \delta$ ($\delta > 0$) we have for all i and j

$$\left|(\mathcal{F}U)_{ij}\right| \leq \sqrt{2\eta \ln\left(\frac{2mn}{\delta}\right)} \ .$$

In particular for $\delta = 0.05$ and since $m \geq n$ there exist a constant C such that $|(\mathcal{F}U)_{ij}| \leq C\eta^{1/2}\sqrt{\log m}$ for all i and j with probability of at least 0.95. The bound on μ can now be achieved by summing the maximum value over the coordinates of a row. □

A seed matrix F is effective if it is easy to apply to A and if $\eta = \max |F_{ij}|^2$ is small. The minimal value of η is $1/m$. If η is $1/m$, then all the entries of F must have squared absolute values of $1/m$. A normalized DFT matrix has this property, and it it can be applied quickly, but it involves complex numbers. A normalized Hadamard matrix has entries that are all $\pm 1/\sqrt{m}$, and in particular are all real. Hadamard matrices do not exist for all dimensions, but they do exist for powers of two, and they can be applied quickly at powers of two. The Walsh-Hadamard series

$$H_l = \frac{1}{\sqrt{2}} \begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix}, \ H_{l+1} = \frac{1}{\sqrt{2}} \begin{pmatrix} H_l & H_l \\ H_l & -H_l \end{pmatrix} \ ,$$

enables the Walsh-Hadamard Transform (WHT). Two other options for F are the Discrete Cosine Transform (DCT) and Discrete Hartley Transform (DHT), which are real, exist for every size, and can be applied quickly. Their η value is $2/m$, twice as large as that of the WHT.

If we use one of the transformation described above, we need a sample of $\Theta(n \log(m) \log(n \log(m)))$ rows to obtain $\kappa = \mathcal{O}(1)$ with high probability. In practice, smaller samples are sufficient. In Section 8.4 discuss implementation issues and considerations for selecting the seed unitary transformation.

A possible alternative mixing strategy is a Kac random walk [**136**]. We define

$$\mathcal{F} = G_{T(m,n)} G_{T(m,n)-1} \cdots G_3 G_2 G_1,$$

where each G_t is a random Givens rotation. To construct G_t, we select two random indices i_t and j_t and a random angle θ_t, and apply the corresponding Givens rotation. The number of rotations is chosen to make the coherence of $\mathcal{F}A$ sufficiently small with high probability.

How small can we make $T(m,n)$? Ailon et al. [4] conjecture that $T(m,n) = O(m \log m)$ will suffice, but they do not have a proof, so we do not currently use this approach. We propose an even simpler random walk where instead of using a random angle θ_t we fix $\theta_t = \pi/4$. We conjecture that still $T(m,n) = O(m \log m)$ will suffice and we have verified this conjecture experimentally.

8.3.3. High coherence due to a few rows. The coherence is the maximal row norm of U, an orthonormal basis for the column space of A. If all the rows of U have low coherence, a uniform random sample of the rows of A leads to a good preconditioner. We now show that even if a few rows in U have a large norm, a uniform random sample still leads to an effective preconditioner. The fundamental reason for this behavior is that a few rows with a large norm may allow a few singular values of the preconditioned system AR^{-1} to be very large, but the number of large singular values is bounded by the number of large rows. A few large singular vectors cause the condition number of AR^{-1} to become large, but they do not affect much the convergence of LSQR (see chapter 3).

Lemma 8.3.8. *Let A be an $m \times n$ full rank matrix where $m \geq n$, and suppose we can write $A = \left[\begin{smallmatrix} A_1 \\ A_2 \end{smallmatrix}\right]$ where A_2 has $l \leq \min(m-n,\, n)$ rows. Let $\mathcal{S} \in \mathbb{R}^{k \times (m-l)}$ be a matrix such that $\mathcal{S}A_1$ is full rank. Let $\mathcal{S}A_1 = QR$ be the QR factorization of $\mathcal{S}A_1$. Then at least $n-l$ singular values of AR^{-1} are between the smallest singular value of A_1R^{-1} and the largest singular value of A_1R^{-1}.*

PROOF. (of Lemma 8.3.8) The singular values of A_1R^{-1} are the square root of the generalized eigenvalues of $(A_1^TA_1, (\mathcal{S}A_1)^T(\mathcal{S}A_1))$. The singular values of AR^{-1} are the square root of the generalized eigenvalues of $(A_1^TA_1 + A_2^TA_2, (\mathcal{S}A_1)^T(\mathcal{S}A_1))$. The matrix $A^TA = A_1^TA_1 + A_2^TA_2$ is a l-rank perturbation of $A_1^TA_1$ so according to Theorem 3.4.3 at least $n-l$ generalized eigenvalues of $(A_1^TA_1 + A_2^TA_2, (\mathcal{S}A_1)^T(\mathcal{S}A_1))$ are between the smallest and largest generalized eigenvalues of $(A_1^TA_1, (\mathcal{S}A_1)^T(\mathcal{S}A_1))$. □

Suppose that A_1 is incoherent but A is coherent. In this case, coherency can be attributed to only a small number of rows (l rows). If A_1 is incoherent and full rank than random sampling will produce a good preconditioner without row mixing. Lemma 8.3.8 implies that the same preconditioner will be a good preconditioner for A as long l is small. In practice, we do not know the partition of A to A_1 and A_2. We simply sample from all the rows of A. But if m is large and the sample is small, the probability of missing any specific row is large; in particular, if l is small then rows from A_2 are likely to be missed. The lemma shows that R is still a good preconditioner. If rows from A_2 are in the sample, the preconditioner is even better.

The lemma assumes that the row sample is full rank. In fact, almost the same result applies even if the sample is rank deficient, as long as we perturb R to make it full rank; see Chapter 3 for details.

8.4. Algorithm and Implementation

In this section we summarize the three major steps of the algorithm: *row mixing (preprocessing)*, *row sampling and QR factorization*, and *iterative solution*. We also discuss how we handle random-sampling failures. The overall solver is presented in Algorithm 5.

Implementation. Our solver currently runs under MATLAB 7.7 [**156**], but it is implemented almost entirely in C. The C code is called from MATLAB using MATLAB's CMEX interface.

Row mixing. In 8.3.2 we suggest five row mixing strategies: DFT, DCT, DHT, WHT and Kac. We chose not to implement DFT and Kac. The DFT of a vector is a complex vector even if the vector is real. Thus, using DFT entails operation-count and memory penalties on subsequent phases when applied on real matrices. Therefore, it is unlikely that an FFT-based algorithm would outperform one based on DCT or DHT. Kac's random walk appears to suffer from poor cache locality due to random index selection.

WHT is theoretically optimal, in the sense that its η value is $1/m$, but it can be applied only if the number of rows is a power of two. By padding the matrix with zeros we can apply WHT to smaller matrices. This causes discontinuous increases in the running time and memory usage as m grows.. We use SPIRAL WHT [**133**] to apply WHT. To get good performance it is essential to use the package's self-optimization feature, which incurs a small one time overhead.

Instead of using WHT, any Hadamard matrix can be used. If H_1 and H_2 are Hadamard matrices then so is $H_1 \otimes H_2$, so using kernels of small Hadamard transforms efficient large Hadamard transforms can be implemented. But to the best of our knowledge, there is currently no efficient implementation of this idea.

DCT and DHT are near optimal alternative (their η value is $2/m$). Their advantages over WHT is that they exist for all vector size and that in principle, they can be always applied in $O(m \log m)$ operations. However, in practice these transforms are quite slow for some sizes. The performance of fast transforms (DCT and DHT) depends on how the input size m can be factored into integers. The performance is not monotone in m. Also, the fast-transform library that we use (FFTW [**103**]) requires tuning for each input size; the tuning step also takes time. To address these issues, we used the following strategy. During the installation of our solver, we generate tuned DCT and DHT solvers for sizes of the form $m = 1000k$ where k is an integer. The information used by FFTW to generate the tuned solvers (called "wisdom" in FFTW) is kept in a file. Before the solver uses FFTW to compute DHT or DCT, this information is loaded into FFTW, so no additional tuning is done at solve time. Before applying DCT or DHT to a matrix, we pad the matrix to the next multiple of 1000, or to a slightly higher multiple if the tuning step suggested that the higher multiple would result in higher performance. One can imagine more sophisticated strategies, based on knowing what kernel sizes FFTW has fast building blocks, and using sizes that are multiple of those building block. The method that we used is not optimal, but it does deliver good performance while keeping tuning time reasonable.

We tune FFTW using aggressive settings, so tuning takes a long time (hours). We also experimented with milder tuning setting. If FFTW's weakest tuning is used, the tuning time of DHT reduces to about 11 minutes, but the time spent in computing DHTs is sometimes doubled. As we shall see in Section 8.5.6, this slows our solver, relative to aggressive setting, by at most 15% (usually less).

Sampling rows and QR factorization. We sample rows by generating a size $\tilde{m}$ vector with random uniform entries in $[0, 1]$, where $\tilde{m}$ is the number of rows after padding. We use MATLAB's RAND function to generate the vector. A row is sampled if the corresponding entry in the vector is smaller than $\gamma n/\tilde{m}$, where γ is a parameter. The expected

number of rows that are sampled is γn, but the actual value can be higher or smaller. This is the same strategy that was suggested in [**89**]. Once the rows are sampled we compute their QR factorization using LAPACK's DGEQRF function.

Row sampling can be combined with row mixing to improve the asymptotic running time. Any k indices of the FFT of a m element vector can be computed using only $\mathcal{O}(m \log k)$ operations [**197**]. This is also true for WHT [**5**]. If we select the sampled rows before the row-mixing we can compute only the mixed rows that are in the sample. We do not use this strategy because the libraries that we used do not have this option.

Iterative solution. We use LSQR to find the solution. Given an iterate x_j with a corresponding residual $r_j = b - Ax_j$, stopping the algorithm when

$$\frac{\|A^T r_j\|_2}{\|A\|_F \|r_j\|_2} \leq \rho \; . \tag{8.4.1}$$

guarantees that x_j is an exact solution of

$$x_j = \arg\min_x \|(A + \delta A)x - b\|_2$$

where $\|\delta A\|_F \leq \rho \|A\|_F$. That is, the solution is backward stable [**61**]. The value of ρ is a parameter that controls the stability of the algorithm. To use this stopping criterion, we need to compute r_j and $A^T r_j$ in every iteration. It is therefore standard practice in LSQR codes to estimate $\|r_j\|_2$ and $\|A^T r_j\|_2$ instead of actually computing them. The estimate formulas used are accurate in exact arithmetic, and in practice they are remarkably reliable [**166**]. If a preconditioner R is used, as in our algorithm, $\|A^T r_j\|_2$ cannot be estimated but $\| \left(AR^{-1}\right)^T r_j\|_2$ can be. Preconditioned LSQR codes estimate $\|AR^{-1}\|_F$ as well, and use the stopping criterion

$$\frac{\| \left(AR^{-1}\right)^T r_j\|_2}{\|AR^{-1}\|_F \|r_j\|_2} \leq \rho \; .$$

that guarantees a backward stable solution to

$$y_j = \arg\min_x \left\|AR^{-1}y - b\right\|_2 \; ,$$

and return $x_j = R^{-1}y_j$. We use the same strategy in our solver. We set $\rho = 10^{-14}$, which is close to $\epsilon_{\text{machine}}$, but not close enough to risk stagnation of LSQR . This setting results in a solver that is about as stable as a QR-based solver.

Handling failures. The bounds in Section 8.3 hold with some probability bounded from below. With some probability, the algorithm can fail to produce an effective preconditioner, in one of two ways: (1) the preconditioner can be rank deficient or highly ill conditioned, or (2) the condition number $\kappa(AR^{-1})$ can be high. When the condition number is high, LSQR converges slowly, but the overall algorithm does not fail. But a rank deficient preconditioner cannot be used with LSQR. To address this issue, we estimate the condition number of the preconditioner R using LAPACK's DTRCON function. If the condition number is too high (larger than $\epsilon_{\text{machine}}^{-1}/5$) we perform another row mixing phase and re-sample. If we repeat this three times and still do not get a full rank preconditioner we give up, assume that the matrix itself is rank deficient, and use LAPACK. This never happened in our experiments on full-rank matrices, but on some matrices we had to mix and sample more than once.

Algorithm 5 Blendenpik's algorithm

x=**blendenpik**($A \in \mathbb{R}^{m \times n}$,$b \in \mathbb{R}^n$)

▷ $m \geq n$, A is non-singular

▷ parameters: γ and transform type

$$\tilde{m} \longleftarrow \begin{cases} 2^{\lceil \log_2 m \rceil} & \text{, WHT} \\ \lceil m/1000 \rceil \times 1000 & \text{, DCT or DHT} \end{cases}$$

$$M \longleftarrow \begin{bmatrix} A \\ 0 \end{bmatrix} \in \mathbb{R}^{\tilde{m} \times n}$$

while not returned

$M \longleftarrow F_{\tilde{m}}(DM)$

▷ D is a diagonal matrix with ± 1 on its diagonal with equal probability

▷ $F_{\tilde{m}}$ is the seed unitary transform (WHT/DCT/DHT), $\Theta(mn \log m)$ operations

Let $\mathcal{S} \in \mathbb{R}^{\tilde{m} \times \tilde{m}}$ be a random diagonal matrix:

$$\mathcal{S}_{ii} = \begin{cases} 1 & \text{, with probability } \gamma n/\tilde{m} \\ 0 & \text{, with probability } 1 - \gamma n/\tilde{m} \end{cases}$$

Factorize: $\mathcal{S}M = QR$, reduced QR factorization ($R \in \mathbb{R}^{n \times n}$)

$\tilde{\kappa} \longleftarrow \kappa_{\text{estimate}}(R)$, condition number estimation (LAPACK's DTRCON)

if $\tilde{\kappa}^{-1} > 5\epsilon_{\text{machine}}$

$x \longleftarrow \text{LSQR}(A, b, R, 10^{-14})$

return

else

if #iterations > 3

failure: solve using LAPACK and return

end if

end if

end while

8.5. Numerical experiments

We experimented with the new algorithm extensively in order to explore its behaviors and to understand its performance. This section reports the results of these experiments (Figure 8.1.1 above shows additional results).

8.5.1. Experimental Setup. We compare the new solver, which we call Blendenpik, to a high-performance dense QR solver and to LSQR with no preconditioning. The dense QR solver is LAPACK's DGELS: a high performance, high quality, portable code. We call LAPACK from MATLAB using a special CMEX interface that measures only LAPACK's running time. No MATLAB-related overheads are included; MATLAB is used here only as a scripting tool.

Running times were measured on a machine with two AMD Opteron 242 processors (we only used one) running at 1.6 GHz with 8 GB of memory. We use GOTO BLAS 1.30 and LAPACK 3.2.1 for basic matrix operations and FFTW 3.2.1 for the DCT and DHT.

The measured running times are wall-clock times that were measured using the FTIME Linux system call.

We evaluated our solver on several classes of random matrices. Random matrices were generated using MATLAB's RAND function (random independent uniform numbers). Ill-conditioned matrices are obtained by generating their SVD decomposition: two random orthonormal matrices and an equally spaced diagonal matrix with the appropriate condition number.

Our solver relies on automatic tuning of the fast-transform libraries that it uses (FFTW and SPIRAL). This is an installation time overhead that is not included in our running-time measurements. Automatic tuning is a technique of growing importance in various numerical libraries, such as the ATLAS [**223**] implementation of the BLAS.

Theoretical bounds relate to the coherence, which is the maximum row norm in the orthogonal factor of the matrix. Our experiments suggest that in practice running time is related to the number of rows that have a large norm in the orthogonal factor. Therefore, we experimented with three types of matrices: *incoherent* matrices, *semi-coherent* matrices and *coherent* matrices. Incoherent matrices $X_{m\times n}$, either well conditioned or ill conditioned, are generated using the `rand` function with no restriction on the structure. Semi-coherent matrices are of the form

$$Y_{m\times n} = \begin{bmatrix} \tilde{B} & \\ & I_{n/2} \end{bmatrix} + 10^{-8} \begin{bmatrix} 1 & \cdots & 1 \\ \vdots & & \vdots \\ 1 & \cdots & 1 \end{bmatrix}$$

where $\tilde{B}$ is a $(m - n/2) \times n/2$ rectangular random matrix and $I_{n/2}$ is a square identity of dimension $n/2$. $B_{m\times n}$ is, in fact, coherent ($\mu(B_{m\times n}) = 1$), but only $n/2$ rows have a large norm in the orthogonal factor. Our coherent matrices have the form

$$Z_{m\times n} = \begin{bmatrix} D_{n\times n} \\ 0_{(m-n)\times n} \end{bmatrix} + 10^{-8} \begin{bmatrix} 1 & \cdots & 1 \\ \vdots & & \vdots \\ 1 & \cdots & 1 \end{bmatrix}$$

where $D_{n\times n}$ is a random diagonal matrix. The orthogonal factors of these matrices have n rows with a large norm. In both semi-coherent and coherent matrices, the constant 10^{-8} matrix is added to make the matrices dense. Some solvers, including LAPACK's (in version 3.2.1), exploit sparsity. Our solver does not. We added the constant matrix to avoid this source of variance; we acknowledge the fact that for some sparse matrices LAPACK's dense solver is faster than our solver.

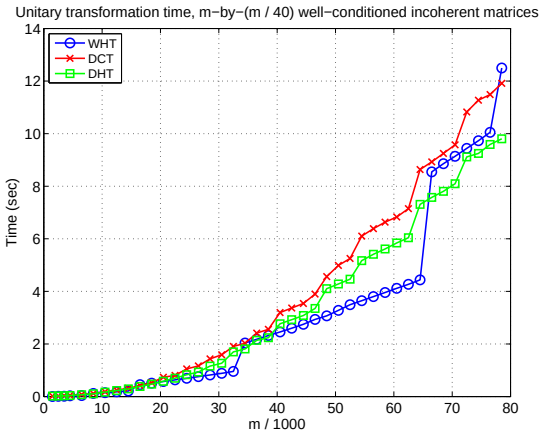


FIGURE 8.5.1. Time spent on the fast unitary transformation (row mixing) for increasingly larger matrices. We tested all three implemented transforms: WHT, DCT and DHT.

8.5.2. Tuning experiments. The behavior of our solver depends on the seed unitary transformation that mixes the rows, on the number of row-mixing steps, and on the sample size. These parameters interact in complex ways, which is not fully captured by asymptotic analyses. We begin with experiments that are designed to help us choose these parameters in the algorithm.

8.5.2.1. *Unitary transformation type.* The row mixing phase uses a fixed seed unitary matrix that only depends on the row dimension of the problem. In 8.3.2 we suggested five different seed unitary matrices. As explained in Section 8.4, we implemented only three of them, all using external libraries: the Walsh-Hadamard transform (WHT), the discrete cosine transform (DCT) and the discrete Hartley transform (DHT). Figures 8.5.1 shows the running time of each transformation time on increasingly larger matrices. WHT is the fastest, but DHT and DCT comes close.

Different unitary transforms improve coherence in different ways. Figure 8.5.2 examines the overall running time of the solver on incoherent, semi-coherent and coherent matrices. For incoherent and semi-coherent matrices there does not seem to be a significant difference between the different mixing methods. WHT's overall time is smaller because it is faster than other methods. On coherent matrices, WHT exhibits poor and erratic performance. In some cases two WHT phases were necessary to obtain a full-rank preconditioner. DHT and DCT continue work well on coherent matrices; the two methods behave the same. It is interesting to note that from a theoretical standpoint WHT is superior, but in practice DHT and DCT work better.

Clearly, WHT's advantage (fast application and a low η) are offseted by it's disadvantages (reduced robustness and a large memory footprint). We therefore decided to use DHT (which is faster than DCT) for all subsequent experiments except for the right graph in Figure 8.1.1, where we used WHT for experimental reasons.

8.5.2.2. *Sample size and number of row mixing steps.* The theoretical analysis shows that sampling $\Omega(n \log(m) \log(n \log(m)))$ rows is sufficient with high probability, but we do not know the constants in the asymptotic notation. The analysis may give bounds in the probability of failure, but even if there is failure (e.g., the condition number is bigger

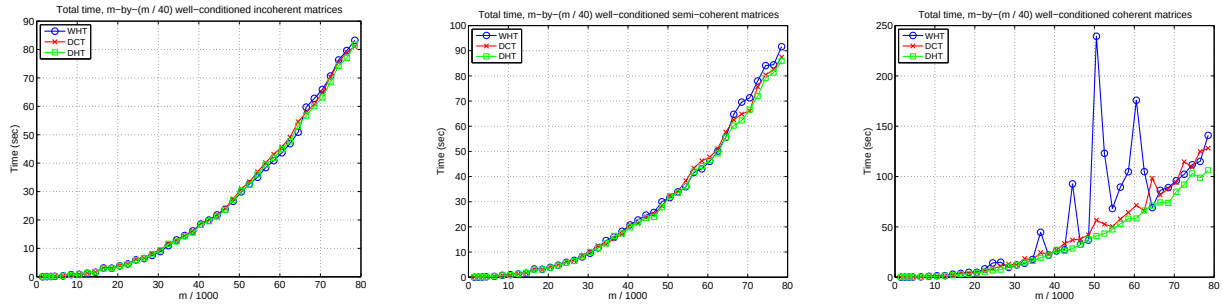


FIGURE 8.5.2. Overall running time of the algorithm with different fast unitary transforms (row mixing) on increasingly larger matrices. We tested on incoherent matrices (left graph), semi-coherent matrices (middle graph) and coherent matrices (right graph).

than the bound) running time might still be good. Convergence behavior is governed by the distribution of singular values, and it is not fully captured by condition number. The contributions of each phase to the running time interact in a complex way that is not fully predictable by worst case asymptotic analysis. Therefore, we performed experiments whose goal is to determine a reasonable sampling size.

We also need to decide on the number of row-mixing steps. Row-mixing steps reduce the coherence and improve the outcome of random sampling. Theoretical bounds state that after a single row mixing step, the coherence is within a $O(\log m)$ factor of the optimal with high probability. Therefore, after the first row mixing step there is still room for improvement. Although there is no theoretical results that states so, it reasonable to assume that additional row mixing steps will reduce coherence further, which will cause LSQR to converge faster, perhaps offsetting the cost of the extra mixing steps.

Figure 8.5.3 present the results of these experiments. We ran experiments with two matrix sizes, $30,000 \times 750$ (top graphs) and $40,000 \times 2,000$ (bottom graphs), and all matrix types, incoherent (left graphs), semi-coherent (middle graphs) and coherent (right graphs). All the matrices were ill-conditioned.

We used sample size γn, where γ ranges from 1.5 to 10. Although the theoretical bound is superlinear, it is not necessarily tight. As the results show, for the range of matrices tested in our experiments the best sample size displays a *sub*linear (in n) behavior (which might change for larger matrices).

For $30,000 \times 750$ matrices the best sample size is around $\gamma = 6$. For $40,000 \times 2,000$ it is $\gamma = 3$. Apparently, for larger matrices a smaller sample is needed (relative to n), contrary to the theoretical analysis. A sample size with$\gamma = 4$ is close to optimal for all matrices. For incoherent and semi-coherent matrices there is a (small) advantage for using only one preprocessing phase. For coherent matrices the best results are achieved when using two preprocessing phases. In fact, using only one preprocessing phase can be disastrous when combined with a sample size that is too small. But with sample size $\gamma = 4$ near optimal results can be achieved with only one preprocessing phase.

Following these experiments we decided to fix $\gamma = 4$ and to use one preprocessing phase. We used these setting for the rest of the experiments These parameters are not optimal in all cases, but they seem to be nearly optimal for most cases. The rest of the experiments in this chapter use these values.

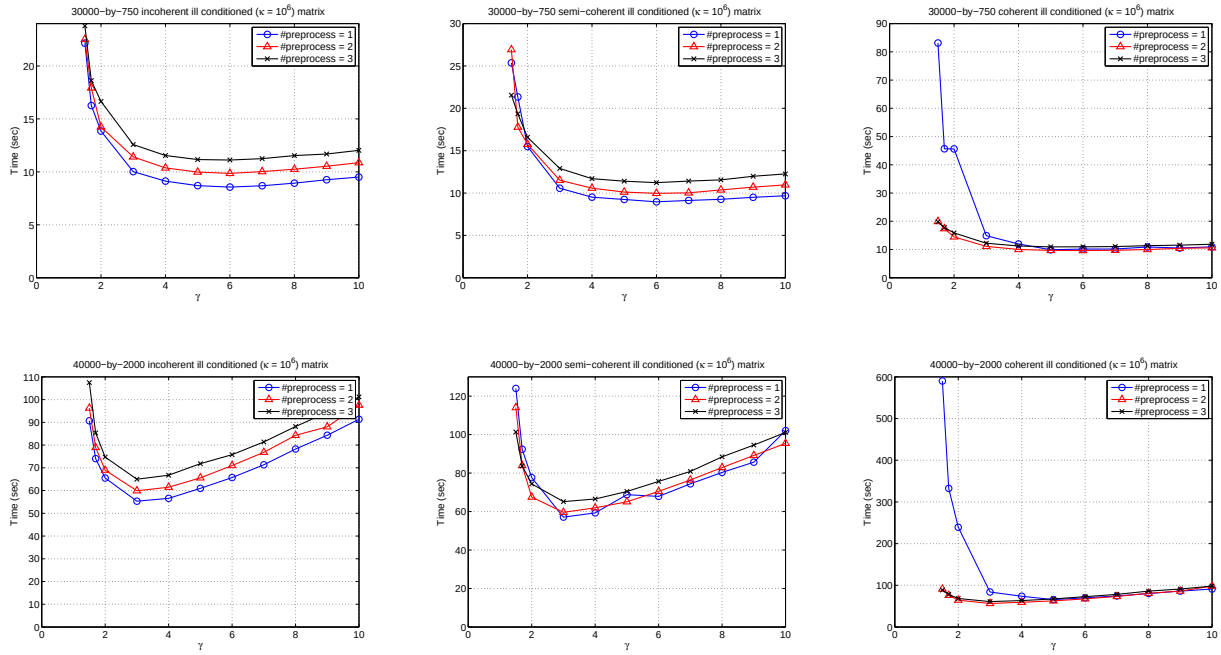


FIGURE 8.5.3. Running time as a function of sample size and number of row mixing steps for $30,000 \times 7,500$ matrices (top graphs) and a $40,000 \times 2,000$ matrices (bottom graphs). We run the same experiment on incoherent matrices (left graphs), semi-coherent matrices (middle graph) and coherent matrices (right graphs).

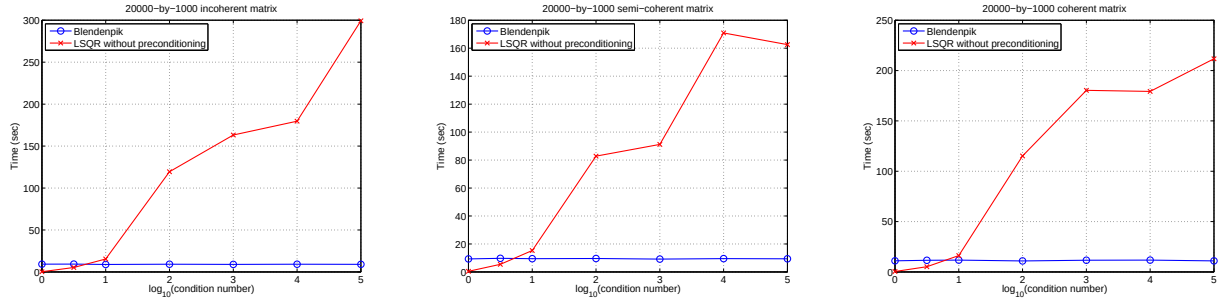


FIGURE 8.5.4. Running time on increasingly ill conditioned matrices.

8.5.3. Ill conditioned matrices. Figure 8.5.4 shows that the condition number of A does not affect our new solver at all, but it does affect unpreconditioned LSQR. On very well conditioned matrices, unpreconditioned LSQR is faster, but its performance deteriorates quickly as the condition number grows.

8.5.4. Easy and hard cases. Figure 8.5.5 compares the performance of our solver and of LAPACK on incoherent, semi-coherent and coherent matrices of four different aspect ratios. The number of elements in all matrices is the same. (LAPACK's running time depends only on the matrix's dimensions, not on its coherence, so the graph shows only one LAPACK running time for each size.) Our solver is slower on matrices with high coherence than on matrices with low coherence, but not by much. Even when the coherence is high,

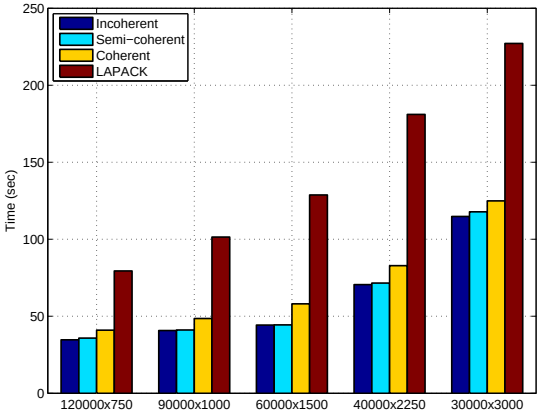


FIGURE 8.5.5. Running time on different coherence profiles.

our solver is considerably faster than LAPACK. Hard cases (high coherence) run slower because LSQR converges slower, so more LSQR iterations are performed (the other phases of the algorithm are oblivious to coherence). It appears that a single row mixing phase does not remove the coherence completely.

8.5.5. Convergence rate. In the experiments whose results are shown in the left graph in Figure 8.5.6, we examine the LSQR convergence rate on a single matrix. The graph shows the norm of the residual after each iteration. Except for the final iterations, where the solver stagnates near convergence, the convergence rate is stable and predictable. This is a useful property that allows us to predict when the solver will converge and to predict how the convergence threshold affects the running time. The rate itself is slower on coherent matrices than on incoherent and semi-coherent ones This is the same issue we saw in Figure 8.5.5.

The graph on the right examines the number of iterations required for LSQR to converge as a function of problem size. On incoherent and semi-coherent matrices the number of iterations grows very slowly. On coherent matrices the number of iterations grows faster.

8.5.6. The cost of the different phases. Figure 8.5.7 shows a breakdown of the running time of our solver for incoherent matrices (left graph) and coherent matrices (right graph) of increasingly larger size. The row mixing preprocessing phase is not a bottleneck of the algorithm. Most of the time is spent on factoring the preconditioner and on LSQR iterations. The most expensive phase is the LSQR phase. The asymptotic running time of the row mixing phase is $\Theta(mn \log m)$, and for the QR phase it is $\Theta(n^3)$. Each LSQR iteration takes $\Theta(mn)$ time and the number of iterations grows slowly. In both graphs $n = m/40$, so the QR phase is asymptotically the most expensive.

The dominance of the LSQR phase implies that considerable speedup can be achieved by relaxing the convergence threshold. In our experiments the convergence threshold was set to 10^{-14}. If a convergence threshold of 10^{-6} is acceptable, for example, we can roughly halve the number of iterations of the LSQR phase, thereby accelerating our solver considerably.

The row mixing phase takes about 15% of overall solver time. Even if we double row mixing time, our solver will still be faster than LAPACK on nearly all of the matrices used in our experiments.

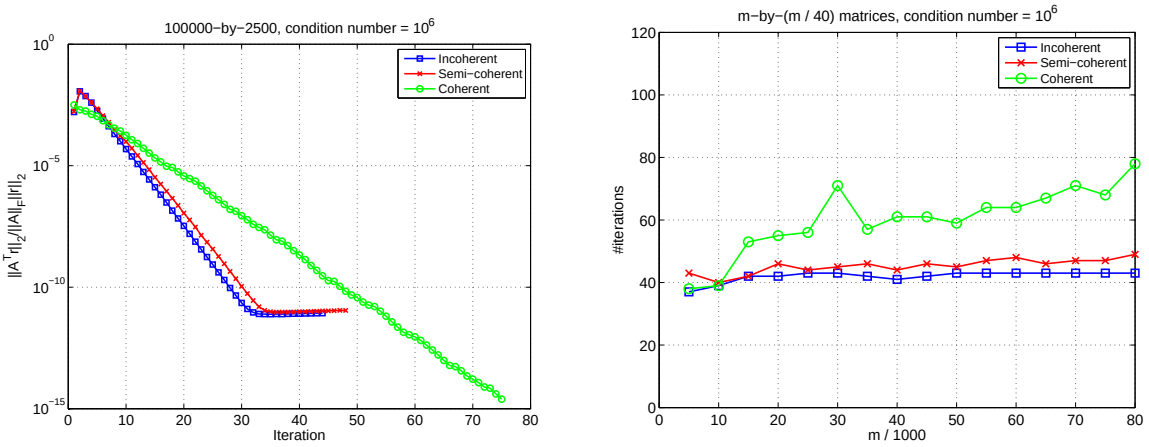


FIGURE 8.5.6. Convergence rate experiments. The left graph shows $\|A^T r^{(i)}\|_2 / \|A\|_F \|r^{(i)}\|_2$, where $r^{(i)}$ is the residual after the ith iteration of LSQR, on three $100,000 \times 2,500$ matrices of different coherence profiles. The right graph shows the number of LSQR iterations needed for convergence on increasingly larger matrices.

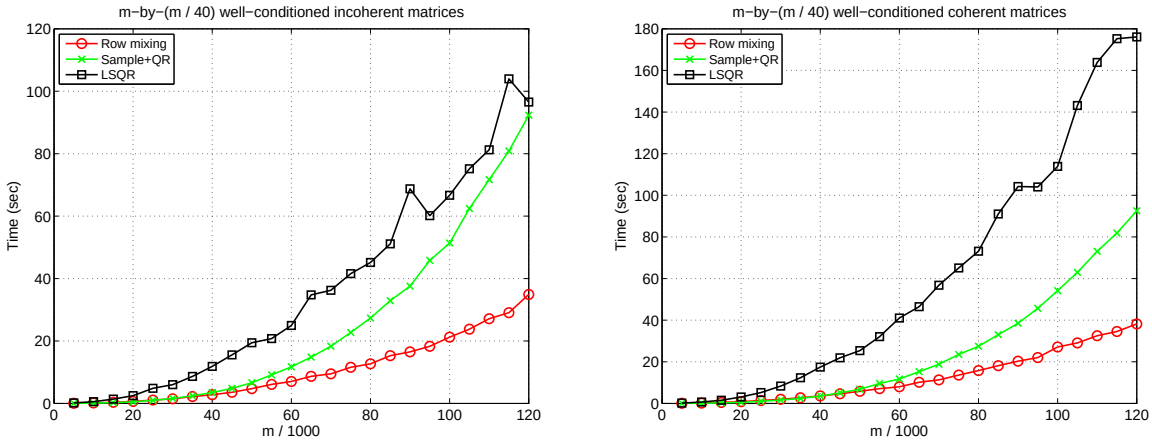


FIGURE 8.5.7. Breakdown of running time on increasingly larger matrices. The plotted series shows the running time of each phase. The left graph shows the breakdown for incoherent matrices, while the right graph shows the breakdown for coherent matrices.

8.5.7. No row mixing. If a matrix is completely incoherent to begin with, we do not need to mix its rows. On such matrices, row mixing takes time but does not reduce the running time of subsequent phases. The left graph in Figure 8.5.8 shows that this is essentially true on random matrices, which have low (but not minimal) coherence; the algorithm runs faster without mixing at all.

The middle graph in Figure 8.3.6 examines performance on coherent matrices whose coherence can be attributed to a small number c of rows. The matrices are of the form

$$S_{(m+c)\times(n+c)} = \begin{bmatrix} S_0 & S_1 \\ 0 & 10^3 \times I_c \end{bmatrix}$$

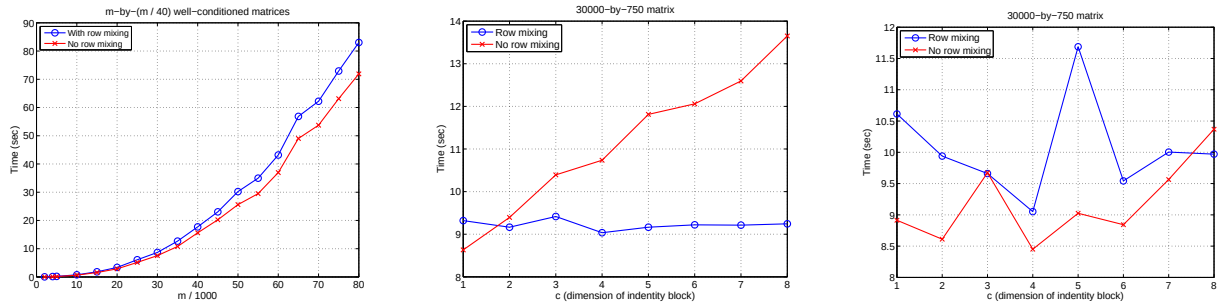


FIGURE 8.5.8. Experiments examining strategies with no row mixing vs. the regular strategy. The left graph compares the solver without a row mixing phase to the solver with a row mixing phase on incoherent matrices. The middle graph compares the same two solvers on matrices with a few important rows. The right graph repeats the experiment of the middle graph, but with an LSQR with full orthogonalization for the solver without row mixing.

where $S_0 \in \mathbb{R}^{m\times(n-c)}$and $S_1 \in \mathbb{R}^{m\times c}$ are random rectangular matrix, and I_c is a c-by-c identity. When c is tiny (1 and 2), row mixing does not improve the running time substantially. But for $c > 2$, with row mixing the running time remains constantly low, while a performance of random sampling without mixing deteriorates as the size of the $10^3 \times I_c$ block grows.

The reason for the deterioration is numerical inaccuracies, and not poor preconditionning. The basis vectors generated by LSQR loses orthogonality because a short recurrence (Lanczos recurrence) is used. A celebrated result of Paige [**164**] shows that loss of orthogonality is large only in the directions of converged or nearly converged Ritz vectors. As long as no Ritz has converged a satisfactory level of orthogonality is maintained. This result explains why isolated singular values in the preconditioned matrix cause numerical problems: the algorithm tends to converge fast for the isolated eigenvalues. Possible solution for this problem are full orthogonalization (expensive), selective orthogonalization [**167**] and others (see §5.3 in [**205**]). We have verified this observation by running LSQR with full orthogonalization (the rightmost graph).

8.6. Discussion and related work

Experiments show that our solver is faster than LAPACK and faster than LSQR with no preconditioning. The algorithm is robust and predictable. The algorithm is competitive in the usual metric of numerical linear algebra, and it demonstrates that randomized algorithms can be effective in numerical linear algebra software. We have not encountered cases of large dense matrices where the solver fails to beat LAPACK, even in hard test cases, and we have not encountered large variance in running time of the algorithm on a given matrix. Even the convergence rate in the iterative phase is stable and predictable (unlike many algorithms that use an iterative method).

Although, the numerical experiment demonstrate the validity of the worst-case theoretical analysis, they also demonstrate that actual performance is not fully described

by it. In some issues actual performance acts differently than suggested by theoretical bounds, or the observed behavior is not explained by the analysis:

- The theoretical analysis suggest that WHT is better in reducing coherence. In practice DHT and DCT work better, even though it takes longer to compute them. In fact, on highly coherent matrices, WHT sometimes fails to mix rows well enough (so we need to apply it again), while this never happened for DHT and DCT.
- The algorithm may fail with some small probability. It may fail to produce an incoherent matrix after row sampling, and important rows may be left out of the random sample (thereby producing a poor preconditioner). Some failures may slow down the solver considerably (for example, when the preconditioner is rank deficient and another row mixing phase is necessary), but it is practically impossible for the algorithm not to finish in finite time on full rank matrices. Current theory does not guarantee that the probability of slowdown is negligible. When using WHT for row mixing, the solver did slow down sometimes due to such failures. When the DHT is used for row mixing, we have not encountered such failures, running time was always good, with a small variance. Apparently the actual probability of failure is much smaller than the theoretical bounds.
- Theoretical bounds require a superlinear sample size. In practice, a linear sample works better. It is unclear whether the reason is that the bounds are not tight, or whether constants come into play.
- The theory relates performance to the coherence of the matrix. Coherence uses the maximum function, which from our experiment, is too crude for analyzing random sampling. Actual performance depends on the distribution of row norms in the orthogonal factor, not just the maximum values. In a sense, the role coherence is similar to the role of condition number in Krylov methods: it provides bounds using extreme values (easy to handle) while actual performance depends on intern values (hard to handle).

The algorithm used by our solver is new, but its building blocks are not. We chose building blocks that are geared toward an efficient implementation. Using WHT for row mixing (and padding the matrix by zeros) was suggested by Drineas et al. [**89**]. Their complete method is not suitable for a general-purpose solver because sample size depends on the required accuracy. Using DCT or DHT for row mixing in low-rank matrix approximations was suggested by Nguyen et al. [**162**]. Their observation carries to least-squares solution. DHT has a smaller memory footprint than WHT, and it works better than WHT and DCT, so we decided to use it. Using the sampled matrix as a preconditioner for an iterative Krylov-subspace method was suggested by Rohklin et al. [**179**]. They use CGLS; we decided to use LSQR because it often works better. The row mixing method in [**179**] uses FFT, which forces the solver to work on complex numbers. Furthermore, their analysis require two FFT applications.

Our observation that the solver can work well even if the post-mixing coherence is high, as long as the number of high-norm rows in U is small, is new.

Unlike previous work in this area, we compared our solver to a state-of-the-art direct solver (LAPACK), showed that it is faster, and explored its behavior on a wide range of matrices. Drineas et al. [**89**] do not implement their algorithm. Rokhlin et al. [**179**] implemented their algorithm, but they compared it to a direct solver that they implemented,

which is probably slower than LAPACK's. They they also experimented only with a small range of matrices (incoherent matrices whose number of rows is a power of two).

A possible future work is to compare Blendenpik to the new communication avoiding least-squares solver of Demmel et al. [**81**], and perhaps devise a communication avoiding variant of Blendenpik. Communication avoiding QR is currently not incoporated into LAPACK, but from private communications we understood that a prototype demonostrates speedups similar to the ones demonstrated by Blendenpik. It is worth noting that Blendenpik uses LAPACK's QR routine so it too might benefit from the new communication avoiding routine.

Bibliography

[1] Achlioptas, D. Database-friendly random projections. In *PODS '01: Proceedings of the twentieth ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database systems* (New York, NY, USA, 2001), ACM, pp. 274–281.

[2] Adamson, A., and Alexa, M. Point-sampled cell complexes. *ACM Trans. Graph. 25*, 3 (2006), 671–680.

[3] Adlers, M. Computing sparse orthogonal factor in matlab. Tech. Rep. LiTH-MAT-R-98-19.

[4] Ailon, N., and Chazelle, B. Approximate nearest neighbors and the fast Johnson-Lindenstrauss transform. In *STOC '06: Proceedings of the thirty-eighth annual ACM Symposium on Theory of Computing* (New York, NY, USA, 2006), ACM, pp. 557–563.

[5] Ailon, N., and Liberty, E. Fast dimension reduction using rademacher series on dual BCH codes. In *SODA '08: Proceedings of the nineteenth annual ACM-SIAM symposium on Discrete algorithms* (Philadelphia, PA, USA, 2008), Society for Industrial and Applied Mathematics, pp. 1–9.

[6] Alexa, M., and Adamson, A. On normals and projection operators for surfaces defined by point sets. In *Eurographics Symp. on Point-Based Graphics* (2004), pp. 149–155.

[7] Alexa, M., Behr, J., Cohen-Or, D., Fleishman, S., Levin, D., and Silva, C. T. Computing and rendering point set surfaces. *IEEE Transactions on Visualization and Computer Graphics 9*, 1 (January 2003), 3–15.

[8] Amenta, N. Defining point-set surfaces. In *ACM Transactions on Graphics (SIGGRAPH) '04* (2004), pp. 264–270.

[9] Amenta, N., and Kil, Y. J. The domain of a point set surfaces. *Eurographics Symposium on Point-based Graphics 1*, 1 (2004), 139–147.

[10] Amestoy, P., Duff, I., and L'Excellent, J.-Y. MUMPS: a MUltifrontal Massively Parallel sparse direct Solver. *http://mumps.enseeiht.fr/*.

[11] Amestoy, P. R., Duff, I. S., and Puglisi, C. Multifrontal **QR** factorization in a multiprocessor environment. *Numerical Linear Algebra with Applications 3*, 4 (1998).

[12] Anderson, E., Bai, Z., Bischof, C., Blackford, S., Demmel, J., Dongarra, J., Croz, J. D., Greenbaum, A., Hammarling, S., McKenney, A., and Sorensen, D. *LAPACK User's Guide*, 3rd ed. SIAM, Philadelphia, PA, 1999. Also available online from `http://www.netlib.org`.

[13] Arioli, M. A stopping criterion for the conjugate gradient algorithm in a element method framework. Tech. rep., Numerische Mathematik, 2000.

[14] Ashby, S. F., Manteuffel, T. A., and Saylor, P. E. A taxonomy for conjugate gradient methods. *SIAM J. Numer. Anal. 27*, 6 (1990), 1542–1568.

[15] Avron, H. Counting triangles in large graphs using randomized matrix trace estimation. In *Proceedings of KDD-LDMTA'10* (2010).

[16] Avron, H., Chen, D., Shklarski, G., and Toledo, S. Combinatorial preconditioners for scalar elliptic finite-element problems. *SIAM Journal on Matrix Analysis and Applications 31*, 2 (2009), 694–720.

[17] Avron, H., Gupta, A., and Toledo, S. New Krylov-Subspace solvers for hermitian positive definite matrices with indefinite preconditioners. Tech. Rep. RC 24698 (W0812-001), IBM T. J. Watson Research Center, Yorktown Heights, NY, December 1, 2008.

[18] Avron, H., Maymounkov, P., and Toledo, S. Blendenpik: Supercharging LAPACK's least-squares solver. *SIAM Journal on Scientific Computing 32*, 3 (2010), 1217–1236.

[19] Avron, H., Ng, E., and Toledo, S. A generalized Courant-Fischer minimax theorem. Tech. rep., Tel-Aviv University, Israel, Aug. 2008.

[20] Avron, H., Ng, E., and Toledo, S. Using perturbed QR factorizations to solve linear least-squares problems. *SIAM Journal on Matrix Analysis and Applications 31*, 2 (2009), 674–693.

[21] Avron, H., Sharf, A., Greif, C., and Cohen-Or, D. ℓ_1-sparse reconstruction of sharp point set surfaces. *ACM Trans. Graph. 29* (November 2010), 135:1–135:12.

[22] Avron, H., Shklarski, G., and Toledo, S. On element SDD approximability. Tech. rep., Tel-Aviv University, Israel, Apr. 2008.

[23] Avron, H., Shklarski, G., and Toledo, S. Parallel unsymmetric-pattern multifrontal sparse lu with column preordering. *ACM Trans. Math. Softw. 34*, 2 (2008), 1–31.

[24] Avron, H., and Toledo, S. Randomized algorithms for estimating the trace of an implicit symmetric positive semi-definite matrix. *Journal of the ACM 59*, 2 (2011).

[25] Bai, Z., Fahey, M., and Golub, G. Some large scale matrix computation problems. *J. Comput. Appl. Math 74* (1996), 71–89.

[26] Bai, Z., Fahey, M., Golub, G., Menon, M., and Richter, E. Computing partial eigenvalue sum in electronic structure calculations. Tech. Rep. SCCM-98-03, Stanford University, Jan 1998.

[27] Barlow, J. L., and Vemulapati, U. B. Rank detection methods for sparse matrices. *SIAM Journal on Matrix Analysis and Applications 13*, 4 (1992), 1279–1297.

[28] Bauer, F. L. Optimally scaled matrices. *Numerische Mathematik 5* (1963), 73–87.

[29] Bekas, C., Kokiopoulou, E., and Saad, Y. An estimator for the diagonal of a matrix. *Appl. Numer. Math. 57*, 11-12 (2007), 1214–1229.

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

[31] Benzi, M., and Tůma, M. A comparative study of sparse approximate inverse preconditioners. *Appl. Numer. Math. 30* (June 1999), 305–340.

[32] Bern, M., Gilbert, J. R., Hendrickson, B., Nguyen, N., and Toledo, S. Support-graph preconditioners. *SIAM Journal on Matrix Analysis and Applications 27* (2006), 930–951.

[33] Bernardini, F., Mittleman, J., Rushmeier, H., Silva, C., and Taubin, G. The ball-pivoting algorithm for surface reconstruction. *IEEE Transactions on Visualization and Computer Graphics 5*, 4 (1999), 349–359.

[34] Bernstein, D. S. *Matrix Mathematics: Theory, Facts, and Formulas with Applications to Linear Systems Theory*. Princeton University Press, 2005.

[35] Bischof, C., and Tang, P. LAPACK Working Note 33: Robust incremental condition estimation. Tech. rep., University of Tennessee, Knoxville, TN, USA, 1991.

[36] Bischof, C. H. Incremental condition estimation. *SIAM Journal on Matrix Analysis and Applications 11*, 2 (1990), 312–322.

[37] Bischof, C. H., and Hansen, P. C. Structure-preserving and rank-revealing QR-factorizations. *SIAM Journal on Scientific and Statistical Computing 12*, 6 (1991), 1332–1350.

[38] Bischof, C. H., Lewis, J. G., and Pierce, D. J. Incremental condition estimation for sparse matrices. *SIAM Journal on Matrix Analysis and Applications 11*, 4 (1990), 644–659.

[39] Bjorck, A. A general updating algorithm for constrained linear least squares problems. *SIAM Journal on Scientific and Statistical Computing 5*, 2 (1984), 394–402.

[40] Björck, Å. *Numerical Methods for Least Squares Problems*. SIAM, 1996.

[41] Boman, E. G., Chen, D., Hendrickson, B., and Toledo, S. Maximum-weight-basis preconditioners. *Numerical Linear Algebra with Applications 11* (2004), 695–721.

[42] Boman, E. G., and Hendrickson, B. On spanning tree preconditioners. Unpublished manuscript, Sandia National Laboratories, 2001.

[43] Boman, E. G., and Hendrickson, B. Support theory for preconditioning. *SIAM Journal on Matrix Analysis and Applications 25*, 3 (2004), 694–717.

[44] Boman, E. G., Hendrickson, B., and Vavasis, S. Solving elliptic finite element systems in near-linear time with support preconditioners. *SIAM Journal on Numerical Analysis 46*, 6 (2008), 3264–3284.

[45] Boutsidis, C., and Drineas, P. Random projections for the nonnegative least-squares problem. *Linear Algebra and its Applications 431*, 5-7 (2009), 760 – 771.

[46] Box, G. E. P., Hunter, W. G., and Hunter, J. S. *Statistics for Experimenters: An Introduction to Design, Data Analysis, and Model Building*. John Wiley & Sons, June 1978.

[47] Boyd, S., and Vandenberghe, L. *Convex Optimization*. Cambridge University Press, 2004.

[48] Braatz, R. D. Response to Chen and Toledo on "minimizing the Euclidean condition number". Private communication, Sept. 2005.

[49] Braatz, R. D., and Morari, M. Minimizing the Euclidean condition number. *SIAM Journal on Control and Optimization 32* (1994), 1763–1768.

[50] Brainman, I., and Toledo, S. Nested-dissection orderings for sparse LU with partial pivoting. In *Proceedings of the 10th SIAM Conference on Parallel Processing for Scientific Computing* (Norfolk, Virginia, Mar. 2001). 10 pages on CDROM.

[51] Brezina, M., Cleary, A. J., Falgout, R. D., Henson, V. E., Jones, J. E., Manteuffel, T. A., McCormick, S. F., and Ruge, J. W. Algebraic multigrid based on element interpolation (amge). *SIAM Journal on Scientific Computing 22*, 5 (2000), 1570–1592.

[52] Candeès, E. J., and Wakin, M. B. An introduction to compressive sampling [a sensing/sampling paradigm that goes against the common knowledge in data acquisition]. *Signal Processing Magazine, IEEE 25*, 2 (2008), 21–30.

[53] Candès, E., and Romberg, J. l1-Magic : Recovery of sparse signals via convex programming. In *http://www.acm.caltech.edu/l1magic/* (October 2005).

[54] Candès, E. J., and Recht, B. Exact matrix completion via convex optimization. *CoRR abs/0805.4471* (2008).

[55] Candès, E. J., Romberg, J., and Tao, T. Robust uncertainty principles: exact signal reconstruction from highly incomplete frequency information. *Information Theory, IEEE Transactions on 52*, 2 (2006), 489–509.

[56] Candès, E. J., Wakin, M. B., and Boyd, S. P. Enhancing sparsity by reweighted ℓ_1 minimization. *Journal of Fourier Analysis and Applications 14* (2008), 877–905.

[57] Chan, T. F. Rank revealing QR factorizations. *Linear Algebra Appl. 88/89* (1987), 67–82.

[58] Chan, T. F., and Hansen, P. C. Some applications of the rank revealing QR factorization. *SIAM Journal on Scientific and Statistical Computing 13*, 3 (1992), 727–741.

[59] Chan, T. F., Osher, S., and Shen, J. The digital TV filter and nonlinear denoising. *IEEE Trans. Image Process 10* (2001), 231–241.

[60] Chandrasekaran, S., and Ipsen, I. C. F. On rank-revealing factorisations. *SIAM Journal on Matrix Analysis and Applications 15*, 2 (1994), 592–622.

[61] Chang, X.-W., Paige, C. C., and Titley-Peloquin, D. Stopping criteria for the iterative solution of linear least squares problems. *SIAM Journal on Matrix Analysis and Applications 31*, 2 (2009), 831–852.

[62] Chartier, T., Falgout, R. D., Henson, V. E., Jones, J., Manteuffel, T., McCormick, S., Ruge, J., and Vassilevski, P. S. Spectral AMGe (ρAMGe). *SIAM Journal on Scientific Computing 25*, 1 (Jan. 2003), 1–26.

[63] Chen, D., and Toledo, S. Vaidya's preconditioners: Implementation and experimental study. *Electronic Transactions on Numerical Analysis 16* (2003), 30–49.

[64] Chen, D., and Toledo, S. Combinatorial characterization of the null spaces of symmetric H-matrices. *Linear Algebra and its Applications 392* (2004), 71–90.

[65] Chen, S. S., Donoho, D. L., and Saunders, M. A. Atomic decomposition by basis pursuit. *SIAM Rev. 43*, 1 (2001), 129–159.

[66] Chen, Y., Davisa, T. A., Hager, W. W., and Rajamanickam, S. Algorithm 8xx: Cholmod, supernodal sparse cholesky factorization and update/downdate. Tech. Rep. TR-2006-005. Submitted to ACM Trans. Math. Software.

[67] Claerbout, J., and Muir, F. Robust modeling of erratic data. *Geophysics 38*, 5 (1973), 826–844.

[68] Concus, P., Golub, G. H., and O'Leary, D. P. A generalized conjugate gradient method for the numerical solution of elliptic partial differential equations. In *Sparse Matrix Computations*, J. R. Bunch and D. J. Rose, Eds. Academic Press, New York, 1976, pp. 309–332.

[69] Daitch, S. I., and Spielman, D. A. Support-graph preconditioners for 2-dimensional trusses. *CoRR abs/cs/0703119* (2007). informal publication.

[70] Daniels, J. I., Ha, L. K., Ochotta, T., and Silva, C. T. Robust smooth feature extraction from point clouds. In *SMI '07: Proceedings of the IEEE International Conference on Shape Modeling and Applications 2007* (2007), pp. 123–136.

[71] Dasgupta, A., Drineas, P., Harb, B., Kumar, R., and Mahoney, M. W. Sampling algorithms and coresets for ℓ_p regression. *SIAM Journal on Computing 38*, 5 (2009), 2060–2078.

[72] Davis, T. The University of Florida Sparse Matrix Collection. *http://www.cise.ufl.edu/research/sparse/matrices.*

[73] Davis, T. A. SuiteSparse: a Suite of Sparse matrix packages. *http://www.cise.ufl.edu/research/sparse/SuiteSparse/.*

[74] Davis, T. A. *Direct Methods for Sparse Linear Systems.* SIAM, Philadelphia, 2006.

[75] Davis, T. A. Multifrontal multithreaded rank-revealing sparse qr factorization. Submitted to *ACM Transactions on Mathematical Software*, 22 pages, 2008.

[76] Davis, T. A., Gilbert, J. R., Larimore, S. I., and Ng, E. G. A column approximate minimum degree ordering algorithm. Tech. Rep. TR-00-005, Department of Computer and Information Science and Engineering, University of Florida, 2000.

[77] Davis, T. A., and Hager, W. W. Dynamic supernodes in sparse cholesky update/downdate and triangular solves. Tech. Rep. TR-2006-004. Submitted to ACM Trans. Math. Software.

[78] Davis, T. A., and Hager, W. W. Modifying a sparse cholesky factorization. *SIAM Journal on Matrix Analysis and Applications 20*, 3 (1999), 606–627.

[79] Davis, T. A., and Hager, W. W. Row modifications of a sparse cholesky factorization. *SIAM J. Matrix Anal. Appl. 26*, 3 (2005), 621–639.

[80] Demmel, J., Dumitriu, I., and Holtz, O. Fast linear algebra is stable. *Numerische Mathematik 108*, 1 (2007), 59–91.

[81] Demmel, J., Grigori, L. amd Hoemmen, M., and Langou, J. Communication-optimal parallel and sequential qr and lu factorizations. Tech. Rep. UCB/EECS-2008-89, University of California, Berkeley, 2008.

[82] Demmel, J. W., Eisenstat, S. C., Gilbert, J. R., Li, X. S., and Liu, J. W. H. A supernodal approach to sparse partial pivoting. *SIAM Journal on Matrix Analysis and Applications 20* (1999), 720–755.

[83] Dey, T. K., and Sun, J. An adaptive mls surface for reconstruction with guarantees. In *SGP '05: Proceedings of the third Eurographics symposium on Geometry processing* (2005), p. 43.

[84] Dey, T. K., and Sun, J. Normal and feature approximations from noisy point clouds. In *FST & TCS 2006, LNCS 4337* (2006), pp. 21 – 32.

[85] Dongarra, J. J., Cruz, J. D., Hammarling, S., and Duff, I. A set of level 3 basic linear algebra subprograms. *ACM Transactions on Mathematical Software 16*, 1 (1990), 1–17.

[86] Donoho, D. L., Elad, M., Temlyakov, V. N., and A. Stable recovery of sparse overcomplete representations in the presence of noise. *IEEE Trans. Inform. Theory 52* (2006), 6–18.

[87] Drineas, P., Mahoney, M. W., and Muthukrishnan, S. Sampling algorithms for ℓ_2 regression and applications. In *SODA '06: Proceedings of the seventeenth annual ACM-SIAM symposium on Discrete algorithm* (New York, NY, USA, 2006), ACM, pp. 1127–1136.

[88] Drineas, P., Mahoney, M. W., and Muthukrishnan, S. Relative-error *cur* matrix decompositions. *SIAM Journal on Matrix Analysis and Applications 30*, 2 (2008), 844–881.

[89] Drineas, P., Mahoney, M. W., Muthukrishnan, S., and Sarlós, T. Faster least squares approximation. *CoRR abs/0710.1435* (2007).

[90] Duff, I., and Scott, J. The HSL mathematical software library. *http://www.hsl.rl.ac.uk/contact.html.*

[91] Duff, I. S., and Vømel, C. Incremental norm estimation for dense and sparse matrices. *BIT Numerical Mathematics 42*, 2 (2002), 300–322.

[92] Elad, M., Starck, J., Querre, P., and Donoho, D. Simultaneous cartoon and texture image inpainting using morphological component analysis (mca). *Applied and Computational Harmonic Analysis 19*, 3 (November 2005), 340–358.

[93] Elkin, M., Emek, Y., Spielman, D. A., and Teng, S.-H. Lower-stretch spanning trees. In *Proceedings of the 37th annual ACM Symposium on Theory of Computing (STOC)* (Baltimore, MD, 2005), ACM Press, pp. 494–503.

[94] Elsey, M., and Esedoglu, S. Analogue of the total variation denoising model in the context of geometry processing. *Multiscale Modeling & Simulation 7*, 4 (2009), 1549–1573.

[95] Faber, V., and Manteuffel, T. Necessary and sufficient conditions for the existence of a conjugate gradient method. *SIAM J. Numer. Anal. 21*, 2 (1984), 352–362.

[96] Feller, W. *An Introduction to Probability Theory and Its Applications, Vol. 2*, 3rd. ed. Wiley, January 1971.

[97] Fleishman, S., Cohen-Or, D., and Silva, C. T. Robust moving least-squares fitting with sharp features. *ACM Trans. Graph. 24*, 3 (2005), 544–552.

[98] Fleishman, S., Drori, I., and Cohen-Or, D. Bilateral mesh denoising. *ACM Trans. Graph. 22*, 3 (2003), 950–953.

[99] Foster, L. V. The probability of large diagonal elements in the QR factorization. *SIAM Journal on Scientific and Statistical Computing 11*, 3 (1990), 531–544.

[100] Frangioni, A., and Gentile, C. New preconditioners for KKT systems of network flow problems. *SIAM Journal on Optimization 14* (2004), 894–913.

[101] Freund, R. W., and Nachtigal, N. M. QMR: a quasi-minimal residual method for non-Hermitian linear systems. *Numerische Mathematik 60*, 1 (Dec 1991), 315–339.

[102] Freund, R. W., and Nachtigal, N. M. An implementation of the QMR method based on coupled two-term recurrences. *SIAM J. Sci. Comput. 15*, 2 (1994), 313–337.

[103] Frigo, M., and Johnson, S. G. FFTW: An adaptive software architecture for the FFT. In *1998 ICASSP Conference Proceedings* (1998), vol. 3, p. 1381.

[104] George, A., and Heath, M. T. Solution of sparse linear least squares problems using Givens rotations. *Linear Algebra Appl. 34* (1980), 69–83.

[105] Gill, P. E., Murray, W., Saunders, M. A., Tomlin, J. A., and Wright, M. H. On projected Newton barrier methods for linear programming and an equivalence to Karmarkar's projective method. *Math. Program. 36*, 2 (1986), 183–209.

[106] Golub, G. H. Numerical methods for solving linear least squares problems. *Numer. Math. 7* (1965), 206–216.

[107] Golub, G. H. Matrices, moments and quadrature II; how to compute the norm of the error in iterative methods. *BIT 37* (1997), 687–705.

[108] Golub, G. H., and Kahan, W. Calculating the singular values and pseudo-inverse of a matrix. *Journal of the Society for Industrial and Applied Mathematics: Series B, Numerical Analysis 2* (1965), 205–224.

[109] Golub, G. H., and Loan, C. F. V. *Matrix Computations (3rd ed.)*. Johns Hopkins University Press, Baltimore, MD, USA, 1996.

[110] Grant, M., and Boyd, S. CVX: Matlab software for disciplined convex programming (web page and software). http://stanford.edu/ boyd/cvx, 2009.

[111] Greenbaum, A. *Iterative Methods for Solving Linear Systems*. SIAM, 1997.

[112] Gremban, K., Miller, G., and Zagha, M. Performance evaluation of a parallel preconditioner. In *9th International Parallel Processing Symposium* (Santa Barbara, April 1995), IEEE, pp. 65–69.

[113] Gremban, K. D. *Combinatorial Preconditioners for Sparse, Symmetric, Diagonally Dominant Linear Systems*. PhD thesis, School of Computer Science, Carnegie Mellon University, Oct. 1996. Technical Report CMU-CS-96-123.

[114] Guennebaud, G., and Gross, M. Algebraic point set surfaces. In *ACM Transactions on Graphics (SIGGRAPH) '07* (2007).

[115] Gupta, A. WSMP: Watson sparse matrix package (Part-III: iterative solution of sparse systems). Tech. Rep. RC-24398, IBM T.J. Watson Research Center, Yorktown Heights, NY, Nov. 2007.

[116] Gupta, A., and George, T. Adaptive techniques for improving the performance of incomplete factorization preconditioning. Tech. Rep. RC 24598 (W0807-036), IBM T. J. Watson Research Center, Yorktown Heights, NY, July 7, 2008. To appear in *SIAM Journal on Scientific Computing*.

[117] Gupta, R. Support graph preconditioners for linear systems. Master's thesis, Texas A&M University, December 2004.

[118] Haase, G., Langer, U., Reitzinger, S., and Schicho, J. Algebraic multigrid methods based on element preconditioning. *International Journal of Computer Mathematics 78*, 4 (2001), 575–598.

[119] Hansen, P. C. *Rank-Deficient and Discrete Ill-Posed Problems: Numerical Aspects of Linear Inversion*. SIAM, Philadelphia, 1998.

[120] Heath, M. T. Some extensions of an algorithm for sparse linear least squares problems. *SIAM Journal on Scientific and Statistical Computing 3*, 2 (1982), 223–237.

[121] Hegland, M., and Saylor, P. E. Block Jacobi preconditioning of the conjugate gradient method on a vector processor. *International Journal of Computer Mathematics*, 1 (1992).

[122] Hénon, P., Ramet, P., and Roman, J. PaStiX: A high-performance parallel direct solver for sparse symmetric definite systems. *Parallel Computing 28*, 2 (2002), 301–321.

[123] Henson, V. E., and Yang, U. M. BoomerAMG: a parallel algebraic multigrid solver and preconditioner. *Applied Numerical Mathematics 41* (2002), 155–177.

[124] Hestenes, M., and Stiefel, E. Methods of conjugate gradients for solving linear systems. *National Bureau of Standards Jounal of Research 49* (1952), 409–436.

[125] Higham, N. J. *Accuracy and Stability of Numerical Algorithms*. SIAM, 1996.

[126] Higham, N. J. *Accuracy and Stability of Numerical Algorithms*, 2nd ed. SIAM, 2002.

[127] Hoemmen, M. *Communication-Avoiding Krylov Subspace Methods*. PhD thesis, University of California, Berkeley. Spring 2010.

[128] Holland, P., and Welsch, R. Robust regression using iteratively reweighted least-squares. *Communications in Statistics - Theory and Methods 6*, 9 (1977), 813 – 827.

[129] Hughes, T. J. R., Levit, I., and Winget, J. An element–by–element solution algorithm for problems of structural and solid mechanics. *Comp. Meth. Appl. Mech. Engng. 36* (1983), 241–254.

[130] Hutchinson, M. F. A stochastic estimator of the trace of the influence matrix for Laplacian smoothing splines. *Communications in Statistics, Simulation and Computation*, 18 (1989), 1059–1076.

[131] Iitaka, T., and Ebisuzaki, T. Random phase vector for calculating the trace of a large matrix. *Physical Review E 69* (2004), 057701–1–057701–4.

[132] Janssen, A. J. E. M., van Leeuwaarden, J. S. H., and Zwart, B. Gaussian expansions and bounds for the Poisson distribution applied to the Erlang B formula. *Advances in Applied Probability 40*, 1 (2008), 122–143.

[133] Johnson, J., and Puschel, M. In search of the optimal Walsh-Hadamard transform. In *ICASSP '00: Proceedings of the Acoustics, Speech, and Signal Processing, 2000. on IEEE International Conference* (Washington, DC, USA, 2000), IEEE Computer Society, pp. 3347–3350.

[134] Jones, T. R., Durand, F., and Desbrun, M. Non-iterative, feature-preserving mesh smoothing. In *SIGGRAPH '03: ACM SIGGRAPH 2003 Papers* (2003), pp. 943–949.

[135] Júdice, J. J., Patricio, J., Portugal, L. F., Resende, M. G. C., and Veiga, G. A study of preconditioners for network interior point methods. *Computational Optimization and Applications 24* (2003), 5–35.

[136] Kac, M. Probability and related topics in physical science. *Wiley Interscience*.

[137] Kambadur, P., Gupta, A., Ghoting, A., Avron, H., and Lumsdaine, A. PFunc: Modern task parallelism for modern high performance computing. In *Proceedings of the 2008 ACM/IEEE conference on Supercomputing (SC)* (Portland, Oregon, November 2009).

[138] Karypis, G., and Kumar, V. A fast and high quality multilevel scheme for partitioning irregular graphs. *SIAM Journal on Scientific Computing 20* (1998), 359–392.

[139] Kolluri, R. Provably good moving least squares. In *SODA '05: Proceedings of the sixteenth annual ACM-SIAM symposium on Discrete algorithms* (2005), pp. 1008–1017.

[140] Koutis, I., and Miller, G. L. A linear work, $O(n^{1/6})$ time, parallel algorithm for solving planar Laplacians. In *Proceedings of the Eighteenth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2007, New Orleans, Louisiana, USA, January 7-9, 2007* (2007), N. Bansal, K. Pruhs, and C. Stein, Eds., SIAM, pp. 1002–1011.

[141] Koutis, I., Miller, G. L., and Peng, R. Approaching optimality for solving sdd systems. *CoRR abs/1003.2958* (2010).

[142] Langer, U., Reitzinger, S., and Schicho, J. Symbolic methods for the element preconditioning technique. Tech. rep., Johannes Kepler Universität (JKU) Linz, Jan. 2002.

[143] Levin, A., Fergus, R., Durand, F., and Freeman, W. T. Image and depth from a conventional camera with a coded aperture. In *SIGGRAPH '07: ACM SIGGRAPH 2007 papers* (2007), ACM.

[144] Levin, D. Mesh-independent surface interpolation. In *Geometric Modeling for Scientific Visualization* (2003), pp. 37–49.

[145] Li, P., Hastie, T., and Church, K. Nonlinear estimators and tail bounds for dimension reduction in l_1 using Cauchy random projections. In *Learning Theory*, N. H. Bshouty and C. Gentile, Eds., vol. 4539 of *Lecture Notes in Computer Science*. Springer Berlin Heidelberg, Berlin, Heidelberg, 2007, ch. 37, pp. 514–529.

[146] Li, X. S., and Demmel, J. W. SuperLU_DIST: A scalable distributed memory sparse direct solver for unsymmetric linear systems. *ACM Transactions on Mathematical Software 29* (2003), 110–140.

[147] Lipman, Y., Cohen-Or, D., and Levin, D. Data-dependent mls for faithful surface approximation. In *SGP '07: Proceedings of the fifth Eurographics symposium on Geometry processing* (2007), pp. 59–67.

[148] Lipman, Y., Cohen-Or, D., Levin, D., and Tal-Ezer, H. Parameterization-free projection for geometry reconstruction. *ACM Trans. Graph. 26*, 3 (2007), 22.

[149] Lipman, Y., Sorkine, O., Levin, D., and Cohen-Or, D. Linear rotation-invariant coordinates for meshes. In *Proceedings of ACM SIGGRAPH 2005* (2005), pp. 479–487.

[150] Lobo, M. S., Vandenbergheb, L., Boyd, S., and Lebret, H. Applications of second-order cone programming. *Linear Algebra and its Applications 284* (1998), 193–228.

[151] Lu, S.-M., and Barlow, J. L. Multifrontal computation with the orthogonal factors of sparse matrices. *SIAM Journal on Matrix Analysis and Applications 17*, 3 (1996), 658–679.

[152] Maggs, B. M., Miller, G. L., Parekh, O., Ravi, R., and Woo, S. L. M. Finding effective support-tree preconditioners. In *SPAA '05: Proceedings of the seventeenth annual ACM symposium on Parallelism in algorithms and architectures* (2005), ACM Pres, pp. 176–185.

[153] Mahoney, M. W., and Drineas, P. *cur* matrix decompositions for improved data analysis. *Proceedings of the National Academy of Sciences 106*, 3 (2009), 697–702.

[154] Manteuffel, T. An incomplete factorization technique for positive definite linear systems. *Mathematics of Computation 34* (1980), 473–497.

[155] MathWorks, T. Matlab version 7.2. software package, Jan. 2006.

[156] MathWorks, T. MATLAB version 7.7. software package, 2008.

[157] Matstoms, P. Sparse qr factorization in matlab. *ACM Trans. Math. Softw. 20*, 1 (1994), 136–159.

[158] Mederos, B., Velho, L., and De Figueiredo, L. H. Robust smoothing of noisy point clouds. In *Proc. of the SIAM Conf. on Geometric Design and Computing* (2003).

[159] Meijerink, J. A., and van der Vorst, H. A. An iterative solution method for linear systems of which the coefficient matrix is a symmetric M-matrix. *Mathematics of Computation 31* (1977), 148–162.

[160] Ng, E. A scheme for handling rank-deficiency in the solution of sparse linear least squares problems. *SIAM J. Sci. Stat. Comput. 12*, 5 (1991), 1173–1183.

[161] Ng, M. K. *Iterative Methods for Toeplitz Systems (Numerical Mathematics and Scientific Computation)*. Oxford University Press, Inc., New York, NY, USA, 2004.

[162] Nguyen, N. H., Do, T. T., and Tran, T. D. A fast and efficient algorithm for low-rank approximation of a matrix. In *41st ACM Symposium on Theory of Computing (STOC 2009)* (2009).

[163] Oztireli, C., Guennebaud, G., and Gross, M. Feature preserving point set surfaces based on non-linear kernel regression. *Computer Graphics Forum 28*, 2 (2009), 493–501.

[164] Paige, C. C. *The Computation of Eigenvalues and Eigenvectors of Very Large Sparse Matrices*. PhD thesis, University of London, 1971.

[165] Paige, C. C., and Saunders, M. A. Solution of sparse indefinite systems of linear equations. *SIAM Journal on Numerical Analysis 12* (1975), 617–629.

[166] Paige, C. C., and Saunders, M. A. LSQR: An algorithm for sparse linear equations and sparse least squares. *ACM Trans. Math. Softw. 8*, 1 (1982), 43–71.

[167] Parlett, B. N., and Scott, D. S. The Lanczos algorithm with selective orthogonalization. *Mathematics of Computation 33*, 145 (Jan 1979), 217–238.

[168] Pauly, M., Keiser, R., and Gross, M. H. Multi-scale feature extraction on point-sampled surfaces. *Computer Graphics Forum 22*, 3 (2003), 281–290.

[169] Pauly, M., Mitra, N., and Guibas, L. Uncertainty and variability in point cloud surface data. In *SGP '04: Symposium on Geometry processing* (2004).

[170] Persson, P.-O., and Strang, G. A simple mesh generator in MATLAB. *SIAM Review 46* (2004), 329–345.

[171] Pierce, D. J., and Lewis, J. G. Sparse multifrontal rank revealing QR factorization. *SIAM Journal on Matrix Analysis and Applications 18*, 1 (1997), 159–180.

[172] Portugal, L., Bastos, F., Júdice, J., Paixao, J., and Terlaky, T. An investigation of interior-point algorithms for the linear transportation problem. *SIAM Journal on Scientific Computing 17* (1996), 1202–1223.

[173] Portugal, L. F., Resende, M. G. C., Veiga, G., and Júdice, J. J. A truncated primal-infeasible dual-feasible interior point network flow method. *Networks 35* (2000), 91–108.

[174] Reitzinger, S. Algebraic multigrid and element preconditioning I. Tech. rep., Johannes Kepler Universität (JKU) Linz, Dec. 1998.

[175] Resense, M., and Veiga, G. An efficient implementation of the network interior-point method. In *Network Flows and Matching: the First DIMACS Implementation Challenge*, D. Johnson and C. McGeoch, Eds., vol. 12 of *DIMACS Series in Discrete Mathematics and Computer Science*. AMS, 1993.

[176] Reuter, P., Joyot, P., Trunzler, J., Boubekeur, T., and Schlick, C. Point set surfaces with sharp features. Tech. rep., LaBRI, 2005.

[177] Robert, Y. Regular incomplete factorizations of real positive definite matrices. *Linear Algebra and its Applications 48* (1982), 105–117.

[178] Rokhlin, V., Szlam, A., and Tygert, M. A randomized algorithm for principal component analysis. To appear in *SIAM Journal on Matrix Analysis and Applications*, 2009.

[179] Rokhlin, V., and Tygert, M. A fast randomized algorithm for overdetermined linear least-squares regression. *Proceedings of the National Academy of Sciences 105*, 36 (2008), 13212–13217.

[180] Rudelson, M., and Vershynin, R. Sampling from large matrices: An approach through geometric functional analysis. *Journal of the ACM 54* (July 2007).

[181] Rudin, L., Osher, S., and Fatemi, E. Nonlinear total variation based noise removal algorithms. *Physica D 60* (1992), 259–268.

[182] Rudin, L. I. Images, numerical analysis of singularities and shock filters. Tech. rep., 1987.

[183] Ruge, J. W., and Stüben, K. Algebraic multigrid (AMG). In *Multigrid Methods*, S. F. McCormick, Ed., vol. 3 of *Frontiers in Applied Mathematics*. SIAM, Philadelphia, PA, 1987, pp. 73–130.

[184] Saad, Y. *Iterative Methods for Sparse Linear Systems*. Society for Industrial and Applied Mathematics, Philadelphia, PA, USA, 2003.

[185] Saad, Y., and Schultz, M. H. GMRES: A generalized minimal residual algorithm for solving nonsymmetric linear systems. *SIAM Journal on Scientific and Statistical Computing 7*, 3 (1986), 856–869.

[186] Sarlos, T. Improved approximation algorithms for large matrices via random projections. In *FOCS '06: Proceedings of the 47th Annual IEEE Symposium on Foundations of Computer Science* (Washington, DC, USA, 2006), IEEE Computer Society, pp. 143–152.

[187] Schenk, O., and Gartner, K. PARDISO solver project. *http://www.pardiso-project.org/*.

[188] Shapiro, A. Upper bounds for nearly optimal diagonal scaling of matrices. *Linear and Multilinear Algebra 29* (1991), 145–147.

[189] Sharf, A., Alcantara, D. A., Lewiner, T., Greif, C., Sheffer, A., Amenta, N., and Cohen-Or, D. Space-time surface reconstruction using incompressible flow. In *SIGGRAPH '08: ACM SIGGRAPH Asia 2008 papers* (2008), vol. 27, ACM, pp. 1–10.

[190] Shen, C., O'brien, J. F., and Shewchuk, J. R. Interpolating and approximating implicit surfaces from polygon soup. In *ACM Transactions on Graphics (SIGGRAPH) '04* (2004), pp. 896–904.

[191] Shepard, D. A two-dimensional interpolation function for irregularly-spaced data. In *Proceedings of the 1968 23rd ACM national conference* (1968), pp. 517–524.

[192] Shewchuk, J. An introduction to the conjugate gradient method without the agonizing pain. Tech. Rep. CMU-CS-94-125, School of Computer Science, Carnegie Mellon University, 1994.

[193] Shklarski, G. *Combinatorial Preconditioners for Finite Element Problems and Other Contributions to Numerical Linear Algebra.* PhD thesis, Tel-Aviv University. August 2008.

[194] Shklarski, G., and Toledo, S. Rigidity in finite-element matrices: Sufficient conditions for the rigidity of structures and substructures. *SIAM Journal on Matrix Analysis and Applications 30*, 1 (2008), 7–40.

[195] Si, H. *TetGen, A Quality Tetrahedral Mesh Generator and Three-Dimensional Delaunay Triangulator: Users's Manual for Version 1.4*, Jan. 2006. available online from `http://tetgen.berlios.de`.

[196] Silver, R. N., and Röder, H. Calculation of densities of states and spectral functions by chebychev recursion and maximum entropy. *Physical Review E 56* (1997), 4822–4829.

[197] Sorensen, H. V., and Burrus, C. S. Efficient computation of the DFT with only a subset of input or output points. *IEEE Trans. Signal Processing 41*, 3 (1993), 1184–1200.

[198] Sorkine, O., Cohen-Or, D., Irony, D., and Toledo, S. Geometry-aware bases for shape approximation. *IEEE Transactions on Visualization and Computer Graphics 11*, 2 (2005), 171–180.

[199] Spielman, D. A., and Srivastava, N. Graph sparsification by effective resistances. In *Proceedings of the 40th annual ACM Symposium on Theory of Computing* (New York, NY, USA, 2008), STOC '08, ACM, pp. 563–568.

[200] Spielman, D. A., and Teng, S.-H. Solving sparse, symmetric, diagonally-dominant linear systems in time $0(m^{1.31})$. In *Proceedings of the 44th Annual IEEE Symposium on Foundations of Computer Science* (Oct. 2003), pp. 416–427.

[201] Spielman, D. A., and Teng, S.-H. Nearly-linear time algorithms for graph partitioning, graph sparsification, and solving linear systems. In *STOC '04: Proceedings of the thirty-sixth annual ACM symposium on Theory of computing* (New York, NY, USA, 2004), ACM Press, pp. 81–90.

[202] Spielman, D. A., and Teng, S.-H. Nearly-linear time algorithms for preconditioning and solving symmetric, diagonally dominant linear systems. Unpublished manuscript available online at http://arxiv.org/abs/cs/0607105, 2006.

[203] Spielman, D. A., and Teng, S.-H. Nearly-linear time algorithms for preconditioning and solving symmetric, diagonally dominant linear systems. *CoRR abs/cs/0607105* (2009).

[204] Starck, J.-L., Elad, M., and Donoho, D. L. Image decomposition via the combination of sparse representations and a variational approach. *IEEE Transactions on Image Processing 14*, 10 (2005), 1570–1582.

[205] Stewart, G. W. *Matrix algorithms.* Society for Industrial and Applied Mathematics, Philadelphia, PA, USA, 2001.

[206] Strassen, V. Gaussian elimination is not optimal. *Numerische Mathematik 14*, 3 (1969), 354–356.

[207] Tasdizen, T., Whitaker, R., Burchard, P., and Osher, S. Geometric surface smoothing via anisotropic diffusion of normals. In *VIS '02: Proceedings of the conference on Visualization '02* (Washington, DC, USA, 2002), IEEE Computer Society, pp. 125–132.

[208] Tasdizen, T., Whitaker, R., Burchard, P., and Osher, S. Geometric surface processing via normal maps. *ACM Trans. Graph. 22*, 4 (2003), 1012–1033.

[209] The MathWorks, Inc. Matlab *Reference Guide.* Natick, MA, Aug. 1992.

[210] Toledo, S., and Avron, H. Combinatorial preconditioners. In *Combinatorial Scientific Computing*, U. Naumann and O. Schenk, Eds. Computational Science series, Chapman & Hall / CRC Press, 2010.

[211] Toledo, S., Chen, D., and Rotkin, V. TAUCS: A library of sparse linear solvers. *http://www.tau.ac.il/ stoledo/taucs/*.

[212] Tomasi, C., and Manduchi, R. Bilateral filtering for gray and color images. In *ICCV '98: Proceedings of the Sixth International Conference on Computer Vision* (1998), p. 839.

[213] Tropp, J. A. Just relax: convex programming methods for identifying sparse signals in noise. *IEEE Transactions on Information Theory 52*, 3 (2006), 1030–1051.

[214] Tsourakakis, C. E. Fast counting of triangles in large real networks without counting: Algorithms and laws. IEEE Computer Society, pp. 608–617.

[215] Vaidya, P. M. Solving linear equations with symmetric diagonally dominant matrices by constructing good preconditioners. Unpublished manuscript. A talk based on this manuscript was presented at the IMA Workshop on Graph Theory and Sparse Matrix Computations, Minneapolis, October 1991.

[216] van der Sluis, A. Condition numbers and equilibration of matrices. *Numerische Mathematik 14* (1969), 14–23.

[217] van der Vorst, H. A. BI-CGSTAB: a fast and smoothly converging variant of BI-CG for the solution of nonsymmetric linear systems. *SIAM J. Sci. Stat. Comput. 13*, 2 (1992), 631–644.

[218] Varga, R. S., B., S. E., and Mehrmann, V. Incomplete factorizations of matrices and connections with H-matrices. *SIAM Journal on Numerical Analysis 17* (1980), 787–793.

[219] Vavasis, S. Private communication, 2007.

[220] Wallace, D. L. Bounds on normal approximations to student's and the chi-square distributions. *The Annals of Mathematical Statistics 30*, 4 (1959), 1121–1130.

[221] Wang, M., and Sarin, V. Constructing m-matrix preconditioners for finite element problems. Submitted to *SIAM Journal on Scientific Computing*, 2008.

[222] Wedin, P. Perturbation theory for pseudo-inverses. *BIT Numerical Mathematics 13* (1973), 217–232.

[223] Whaley, R. C., and Petitet, A. Minimizing development and maintenance costs in supporting persistently optimized BLAS. *Software: Practice and Experience 35*, 2 (February 2005), 101–121. `http://www.cs.utsa.edu/~whaley/papers/spercw04.ps`.

[224] Wong, M. N., Hickernell, F. J., and Liu, K. I. Computing the trace of a function of a sparse matrix via Hadamard-like sampling. 2004.