ALGORITHMS FOR THE EQUILIBRATION OF MATRICES AND THEIR APPLICATION TO LIMITED-MEMORY QUASI-NEWTON METHODS

A DISSERTATION
SUBMITTED TO THE INSTITUTE FOR
COMPUTATIONAL AND MATHEMATICAL ENGINEERING
AND THE COMMITTEE ON GRADUATE STUDIES
OF STANFORD UNIVERSITY
IN PARTIAL FULFILLMENT OF THE REQUIREMENTS
FOR THE DEGREE OF
DOCTOR OF PHILOSOPHY

Andrew Michael Bradley

May 2010

Abstract

Diagonally scaling a matrix often reduces its condition number. Equilibration scales a matrix so that the row and column norms are equal. We review the existence and uniqueness theory for exact equilibration. Then we introduce a formalization of approximate equilibration and develop its existence and uniqueness theory. Next we develop approximate equilibration algorithms that access a matrix only by matrix-vector products. We address both the nonsymmetric and symmetric cases.

Limited-memory quasi-Newton methods may be thought of as changing the metric so that the steepest-descent method works effectively on the problem. Quasi-Newton methods construct a matrix using vectors of two types involving the iterates and gradients. The vectors are related by an approximate matrix-vector product. Using our approximate matrix-free symmetric equilibration method, we develop a limited-memory quasi-Newton method in which one part of the quasi-Newton matrix approximately equilibrates the Hessian.

Often a differential equation is solved by discretizing it on a sequence of increasingly fine meshes. This technique can be used when solving differential-equation-constrained optimization problems. We describe a method to interpolate our limited-memory quasi-Newton matrix from a coarse to a fine mesh.

Acknowledgements

I would like to thank a number of mentors and colleagues.

Prof. Walter Murray has been a merry and patient advisor. My then-new interest in computational mathematics grew considerably when I took a course with Walter as an undergraduate. I value the tremendous freedom I have had to explore topics during the course of my Ph.D. work. Our discussions gave crucial direction to this research.

My friendship with Prof. Michael Saunders began when I took his course on large-scale optimization. His focus on developing high-quality, useful mathematical software is a model for my own work.

I would like to thank Prof. Yinyu Ye for being on my oral and reading committees and our helpful discussions about matrix scaling.

An exciting part of computational mathematics is collaborating with scientists and engineers to help to solve their problems. I have worked on two side projects that have been particularly rewarding. I would like to thank Profs. Lawrence Wein, Paul Segall, and Margot Gerritsen for their support of these endeavors; and the first two for serving on my oral examination committee.

Prof. Gene Golub kindly advised my undergraduate honors thesis and was a happy part of my first two years as a Ph.D. student.

I would like to thank the ICME staff for their hard work. I particularly would like to thank Indira Choudhury. We ICME students are very lucky to have such a warm and generous advocate.

ICME is a tight-knit community of students, and Durand 112 has been a fun and supportive place to work.

This work was supported by a National Science Foundation Graduate Research Fellowship; a Scott A. and Geraldine D. Macomber Stanford Graduate Fellowship; and grants from the Office of Naval Research and the Army High Performance Computing Research Center.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

Quasi-Newton methods

Consider the unconstrained quadratic program $\min_x f(x)$ for $f(x) \equiv \frac{1}{2}x^T Ax + b^T x$. The gradient is $g(x) \equiv \nabla f(x) = Ax + b$. In a quasi-Newton (QN) method, an approximation to the Hessian is constructed by assembling pairs $\{s, y\}$ into a matrix. s is the difference between iterates: $s \equiv x^+ - x$; y is the difference between the gradients at the iterates: $y \equiv g^+ - g$. x^+ occurs at the iteration subsequent to x, and similarly for other quantities using the superscript-+ notation. Observe that $As = y$ and so s and y are related by a matrix-vector product involving the Hessian. For a general optimization problem, the two are approximately related by a matrix-vector product.

On an unconstrained convex quadratic program, many quasi-Newton methods obtain the Hessian after a finite number of iterations. In real applications—for example, on nonconvex, constrained problems; and on very large problems for which the storage for a quasi-Newton matrix is limited—the Hessian is not recovered. A QN method may be thought of as changing the metric of the problem while the problem is being solved. Search directions are then steepest-descent directions for the modified problem corresponding to the current QN matrix.

A matrix B_0 initializes the QN matrix. Most often, B_0 is chosen to be a diagonal matrix. An algorithm can change B_0 as often as at each update of the QN matrix. Section 3.1.2 shows that in Broyden-class QN methods, a diagonal B_0 may be interpreted as *scaling* the problem. Therefore, an effective choice for B_0 is a scaling matrix that minimizes the condition number of the Hessian.

The time a solver requires to find a solution is a key performance criterion. Time may be divided into two parts: time spent in the solver, chiefly on linear algebra operations; and time spent in the *user function*, which is the software client's implementation of the problem. For many problems, the time required by the solver is considerably less than the time required by the user function. Hence an optimization algorithm is often developed with the objective of minimizing the number of user function calls. Improving the particular QN method an optimization software package uses can reduce the number of user function calls required to solve a problem.

Matrix equilibration

The performance of many algorithms, particularly iterative ones, depends on the condition number of an underlying matrix. For example, the convergence rate of the steepest-descent method and Krylov-subspace methods depends on the condition number relevant to the problem. Hence a good scaling method substantially reduces the condition number of the matrix.

If a matrix is symmetric positive definite, scaling it to unit diagonal is effective and always possible. If a matrix is indefinite, such scaling—substituting, say, 1 for a 0 diagonal element—is less often effective. Far more effective in the latter case is scaling the matrix so that its row and column norms are all equal. This scaling procedure is called *equilibration*. If a matrix is symmetric, one generally prefers symmetric scaling. The theory for symmetric scaling is often more intricate than for nonsymmetric scaling; additionally, algorithms must be specialized to this case.

For a particular scaling method, we ask three questions: 1. What matrices can be scaled by the method? 2. Is the resulting scaled matrix unique? 3. Are the scaling matrices unique? The first question is self-evidently important: a problem solver wants to know in advance whether a particular matrix of interest is scalable by a particular method. A positive answer to the second question assures that, relative to the particular scaling method, an algorithm has done as well as it can: there is not another scaled matrix that is better—say, has a lower condition number. The third question may matter if one wants the scaling matrices to have particular properties: for example, to have a bounded condition number. Nonuniqueness provides freedom to vary the scaling matrices according to additional criteria.

It is unnecessary to scale a matrix exactly; approximate scaling may achieve a sufficient reduction in condition number. The class of approximately scalable matrices may be larger than the class of exactly scalable matrices.

Matrix-free methods

An algorithm is said to be a *matrix-free* method if it obtains information about a matrix only through matrix-vector products. Matrix-free methods are useful when a matrix is represented implicitly: for example, by a factorization or by an algorithm. If a matrix is accessible only by matrix-vector products, we think of the matrix as an *operator*.

Quasi-Newton methods for differential-equation-constrained optimization problems

Simulating a physical phenomenon often involves solving a system of differential equations. The engineer or scientist may want to do more than simply simulate a system. An increasingly common computational problem is to optimize parameters governing the physical system. The resulting problem is a differential-equation-constrained optimization problem (DECP).

A common technique for solving a differential equation is to solve the discretized problem on a sequence of increasingly fine meshes. Often the fine mesh is adapted to the solution on the coarse mesh. The same technique is useful when solving a DECP. If a quasi-Newton method is used, the data by which the QN matrix is formed on the coarse mesh may be useful to initialize the QN matrix on the fine mesh.

Our work

In Chapter 2, we review the theory governing the existence of exactly scalable matrices and the uniqueness of the scaling matrices and the scaled matrix. Then we introduce ε-scaling, a formalization of approximate equilibration, and develop the analogous existence and uniqueness theory. We discuss both the nonsymmetric and symmetric cases.

Next we develop matrix-free methods to approximately scale symmetric, square nonsymmetric, and rectangular matrices. We test our algorithms on a large, well-known test set of matrices. Our performance criteria are change in condition number and a metric quantifying the degree of equilibration.

In Chapter 3, we use our ideas on matrix-free approximate equilibration to develop a limited-memory quasi-Newton method. The algorithm updates B_0 to scale the problem. It uses the pairs $\{s, y\}$ just as any QN method does, and so no additional user function calls are necessary. That s and y are related by an *approximate* matrix-vector product provides the crucial connection to our matrix-free equilibration methods. Since our algorithm updates B_0, it complements standard QN methods that build on the initial matrix B_0. We combine our update with a well-known limited-memory method.

After designing and analyzing the method, we describe its implementation in SNOPT [25], a software package that is designed to solve large, generally constrained optimization problems. Then we test the modified SNOPT on six test sets. While computation time is ultimately the only important criterion, it can be quite deceptive when testing an algorithm on test sets of problems. The test set may contain a mix of problems having different sizes and characteristics, for example. Thus, while we report time for test sets for which time makes sense, our primary assessment criterion is the number of user function calls.

In Chapter 4, we develop a method to interpolate a limited-memory quasi-Newton matrix from a coarse mesh to a fine one. We report results for two classes of problems.

Chapter 2

Equilibration of matrices

Preconditioning a matrix is an important first step for solving a linear equation; iterative methods particularly benefit. Broadly, preconditioning is intended to cluster eigenvalues or to reduce the condition number of a matrix. Problem-dependent preconditioners often achieve both goals. In the absence of any other preconditioner, a diagonal scaling matrix is useful to reduce the condition number of a matrix. The problem of finding a diagonal scaling matrix has the equivalent names *equilibration*, *balancing*, and *binormalization*. Equilibration algorithms exist to achieve a number of different objectives: for example, unit diagonal elements, equal row or column p-norms, bounded elements. The objectives may be achieved either exactly or approximately.

For a particular type of scaling, several questions arise: What is the class of matrices that can be scaled? Is the scaling unique? How much is the condition number of the matrix decreased? Must the scaling be exact, or can it be approximate? For a particular problem, additional questions arise. What algorithms can implement the scaling? In particular, does one have access to the elements of the matrix, or must one use matrix-vector products only?

In this chapter, we consider the problem of scaling a square matrix A by positive diagonal matrices so that the p-norm of each row and column of the resulting matrix is approximately the same. Our focus is on the symmetric problem, in which a symmetric matrix is symmetrically scaled. First, we review existence and uniqueness theorems for exact scaling of A. Second, we develop and prove a necessary and sufficient condition for approximately scaling positive diagonal matrices to exist. Third, we develop two new approximate scaling algorithms that balance the row and column 2-norms and that access A only through matrix-vector products. Finally, we describe the performance of the algorithms on a large test set.

2.1 Background

2.1.1 Notation

The vectors x and d are special: $x_i \equiv d_i^{-1}$. Let $V \equiv \mathrm{diag}(v)$, where v is a vector. For a matrix A, let B be such that $B_{ij} = |A_{ij}|^p$. If $p = 2$, $B \equiv A \circ A$, *i.e.*, B is the element-wise (Hadamard) product of A. If A is complex, the $\circ$ operator conjugates the first argument. e is the vector of all ones. The square of a vector is applied by element: $(v^2)_i \equiv v_i^2$. Similarly, a vector product written as xy is performed by element: $(xy)_i = x_i y_i$. An inequality such as $v > 0$, where v is a matrix or vector, applies element-wise. Index sets are denoted by calligraphic letters, and $\mathcal{N}$ is reserved for the set $\{1, 2, \ldots, n\}$. The cardinality of the set is denoted, for example, $|\mathcal{N}| = n$. Often a sum over an index set is abbreviated when the index set is clear; for example, $\sum_{i \in \mathcal{N}}$ may be written simply as $\sum_i$.

2.1.2 The scaling equation

Suppose we want the p-norm of each row of the symmetric matrix A to be $c > 0$. We must find a vector $x > 0$ such that

$$x_i \sum_j |A_{ij}|^p x_j = c,$$

or, more compactly,

$$XBx = ce. \tag{2.1}$$

If x^* satisfies (2.1) for $c = c_1 > 0$, then $\sqrt{c_2/c_1}x^*$ is a solution for $c = c_2 > 0$. Consequently, we need not consider the particular value of $c > 0$ in what follows and so we set $c = 1$.

We refer to (2.1) as the *scaling equation.* Although algorithms must use A, existence and uniqueness theory need consider only the nonnegative matrix B. If $p = 1$ and A is nonnegative, then $A = B$. We reserve the term *binormalization* for the case $p = 2$. We say A is *scalable* if there exists $x > 0$ satisfying (2.1). We call such a vector a *scaling vector* or, on occasion in the case of $p = 2$, a *binormalization vector.* If two matrices B_1 and B_2 are related to each other by $B_2 = XB_1X$, we say the two are *diagonally equivalent.*

Observe that $c^{-1}XBX$ is *doubly stochastic.*

Not all matrices are scalable. Furthermore, the benefits of scaling may be obtained by approximate rather than exact scaling. Livne and Golub [42] introduced a measure of approximate scaling: A is scaled to a residual whose norm is ε if $\|XBx - e\| = \varepsilon$. We introduce the following definition: A is *ε-scalable* if for any $\varepsilon > 0$, there exists $x > 0$ satisfying $\|XBx - e\| \leq \varepsilon$. Of course a scalable matrix is ε-scalable.

Nonsymmetric scaling is similar. In this case, there exist $x, y > 0$ such that

$$XBy = e \quad \text{and} \quad YB^T x = e.$$

If A is rectangular, the row and column norms may differ in magnitude. We address the rectangular case in numerical experiments only; our theoretical analysis assumes A is square. A is *nonsymmetrically ε-scalable* if for every $\varepsilon > 0$, each equation can be satisfied to a residual having norm no greater than ε.

Numerical experiments (see, *e.g.*, [42] and Section 2.4) for the case $p = 2$ show that the condition number of the scaled matrix A is often considerably less than that of A. For indefinite A, the far simpler procedure of scaling to unit diagonal elements is undefined if a diagonal element is zero and problematic if a diagonal element is small. Furthermore, the reduction in condition number is frequently substantially less than that achieved by equilibration.

2.1.3 Context and main results

Equilibration of matrices has a long history. Research on this problem divides into three parts: analyzing the improvement diagonal scaling offers, either in terms of reducing the condition number or a related quantity of the matrix or in terms of the improvement in sparsity or quality of a factorization; finding necessary or sufficient conditions for the existence or uniqueness of diagonal scaling matrices; and developing and analyzing algorithms. Several equilibration problems exist, although all are generally formulated in terms of nonnegative matrices; most notable are variations in which row or column norms are not equal but rather are specified by positive vectors.

Existence and uniqueness

Sinkhorn wrote a series of papers in the 1960s that detailed some of the existence and uniqueness theory governing matrix equilibration. Sinkhorn and Knopp's paper [62] is quite frequently cited and contains the well-known iteration for matrix balancing that bears their names. Sinkhorn and Knopp proved the major convergence result for the algorithm in [62] in the course of proving the condition for existence of scaling matrices for the nonsymmetric problem; independently, Brualdi, Parter, and Schneider proved overlapping results in [9]. Before we can discuss their theorems, we must introduce some definitions and preliminary results.

A square matrix has total support if every nonzero element occurs in the positive main diagonal of a matrix that is a column permutation of the original matrix. Formally, let A be a square matrix having total support, and suppose A_{ij} is nonzero. Then there exists a permutation σ such that $\sigma(i) = j$ and each $A_{k\sigma(k)}$ is nonzero. We say that σ, or alternatively the positive diagonal associated with σ, *supports* the nonzero element A_{ij}.

A matrix has *support that is not total*, or simply has *support*, if a positive main diagonal exists under a column permutation. By this definition it is evident that a square matrix has support if and only if it is *structurally nonsingular* [18].

Mirsky and Perfect [44] showed that a matrix has total support if and only if there exists a doubly stochastic matrix having the same zero pattern. Other authors use only the "if" part, which follows directly from Birkhoff's Theorem: a doubly stochastic matrix is a convex combination of permutation matrices (see, *e.g.*, Theorem 8.7.1 of [28]).

A matrix A is *partly decomposable* if there exist permutation matrices P and Q such that

$$PAQ = \begin{pmatrix} B & \\ C & D \end{pmatrix},$$

where B and D are square matrices. A square matrix is *fully indecomposable* if it is not partly decomposable. A fully indecomposable matrix has total support [8].

Sinkhorn and Knopp proved the following in [62].

Theorem 1 (Sinkhorn and Knopp)**.** *Let A be a nonnegative square matrix.*

1. *A is scalable if and only if A has total support. That is, there exist positive diagonal matrices D_1 and D_2 such that $B \equiv D_1AD_2$ is doubly stochastic.*
2. *If A is scalable, then B is unique.*
3. *D_1 and D_2 are unique up to a scalar multiple if and only if A is fully indecomposable.*
4. *The Sinkhorn-Knopp iteration converges to a doubly stochastic matrix if and only if A has support. If A has support that is not total, then D_1 and D_2 have elements that diverge.*

Hence scalability of a matrix is entirely determined by its zero pattern. Parts 1–3 were independently discovered in [9]: the authors of each paper acknowledge the other accordingly. The necessity of total support follows directly from Birkhoff's theorem.

In addition to working on the nonsymmetric problem, Brualdi, Parter, and Schneider [9] provided an early existence result for scaling symmetric matrices. Their Corollary 7.7 is equivalent to the existence part of our Theorem 4. Marshall and Olkin [43] obtained overlapping results, and our proof technique for Theorem 4 is essentially theirs. Brualdi and his coauthors also developed several technical results. Corollary 7.8 concerns the structure of a matrix: *Let A be nonnegative and symmetric. Then every other (not necessarily symmetric) matrix B having the same zero pattern as B can be scaled by a positive diagonal D so that DBD is row stochastic if and only if A has a positive diagonal.* Interestingly, their Lemma 7.3 reveals the same block structure (2.9) that we find in our Theorem 3, although the mathematical setting is different. Theorem 8.2 of [9] states a rather technical sufficient condition for uniqueness more general than our Theorem 4, and several other more general sufficient conditions have appeared, including in [43].

Brualdi, Parter, and Schneider proved a number of results for both the nonsymmetric and symmetric problems. However, according to Brualdi [7], Csima and Datta [16] obtained the definitive existence theorem for symmetric scaling three years after the publication of [62] and [9].

Theorem 2 (Csima and Datta)**.** *A symmetric matrix is scalable if and only if it has total support.*

Interestingly, the necessary and sufficient condition of total support in this theorem is identical to that in part 1 of Theorem 1. The necessary part follows directly from part 1 of Theorem 1, but the sufficiency part requires several steps. The key ideas are as follows. Total support of a symmetric matrix A implies A is composed (in a precisely defined sense) of fully indecomposable matrices having total support. Each such matrix can be scaled; by part 3 of Theorem 1, the scaling is unique; and so the scaling is symmetric (because $A = A^T$ and if $D_1 A D_2$ and $D_2 A^T D_1$ are doubly stochastic and A is fully indecomposable, D_1 and D_2 are unique up to a scalar multiple). The scaling matrices for the submatrices of A can be assembled to form a scaling matrix for A. Total support is essential to this proof, and so the matter remains of support that is not total.

It appears that subsequent work has not resolved the question. We discovered the following condition and prove it in Section 2.2.2.

Theorem 3. *A symmetric matrix is ε-scalable if and only if it has support.*

An analog of the "if" part of Theorem 3 for the nonsymmetric problem is directly implied by part 4 of Theorem 1: one can terminate the Sinkhorn-Knopp iteration when the current iterate produces a matrix that is approximately doubly stochastic to a given tolerance. Just as Theorem 2 reveals the same condition as part 1 of Theorem 1, Theorem 3 reveals the same condition as part 4 of Theorem 1. Yet in both cases, the proofs are quite different for the nonsymmetric and symmetric problems.

The question of uniqueness remains. Part 2 of Theorem 1 states that if A has total support, although D_1 and D_2 may not be unique, the doubly stochastic matrix $D_1 A D_2$ is. Part 4 states that if A has support that is not total, the Sinkhorn-Knopp iteration converges to a doubly stochastic matrix C. Parlett and Landis [57] obtained the same result for a class of iterations obeying certain conditions, of which the Sinkhorn-Knopp iteration is a member. Both sets of authors characterized the matrix C as follows. To each nonzero element in A_{ij} that lacks support there corresponds a zero element C_{ij}; to each nonzero element A_{ij} that has support there corresponds a nonzero element C_{ij}. This is not surprising: the equivalence of having total support and having the same zero pattern as a doubly stochastic matrix (due to Perfect and Mirsky), combined with part 1 of Theorem 1, show that a matrix lacking total support does not have the zero pattern of a doubly stochastic matrix. However, neither Sinkhorn and Knopp nor Parlett and Landis answered the following question: In what sense, if any, is C unique if A has support that is not total? We provide an answer to this question in terms of ε-scalability in Theorem 6. By developing a uniqueness theorem in terms of ε-scalability, we sidestep the issue of particular algorithms: any algorithm that ε-scales a matrix yields a matrix C that is unique in the sense of Theorem 6.

Quite a number of papers have reported new methods to prove classic results. For example, Borobia and Canto [5] proved Sinkhorn's early result on scaling positive matrices [61]—parts 1–3 of Theorem 1 applied to strictly positive matrices—using geometric methods. O'Leary [52] proved Marshall and Olkin's theorem from [43]—an spd matrix can be symmetrically scaled to achieve a given positive row sum vector—using a new constructive method, and slightly extended the result.

Johnson and Reams [30] recently described a number of new conditions characterizing the existence of a scaling matrix for the symmetric problem when the matrix is general rather than nonnegative. A symmetric matrix is *copositive* if $x^T Ax \geq 0$ for all $x \geq 0$ and *strictly copositive* if $x^T Ax > 0$ for $x \geq 0$ and $x \neq 0$. Marshall and Olkin [43] showed that A is scalable if it is strictly copositive. The rather technical Theorem 4 of [30] generalizes this condition by considering a certain cone.

Neumaier and Olschowka [49, 54] developed a scaling result related to structural rank for a different kind of scaling than we consider. They showed that a (symmetric) matrix that has support can be scaled so that every entry of the resulting (symmetric) matrix is bounded in magnitude by 1 and the diagonal elements are all 1.

Algorithms

Many equilibration algorithms have been proposed. The classical iteration is that of Sinkhorn and Knopp, who first analyzed its convergence properties. According to Parlett and Landis [57], the iteration itself was used as early as 1940, and according to Knight [35], as early as the 1930s in an application to traffic flow. Parlett and Landis [57] developed several iterations that outperformed the Sinkhorn-Knopp iteration on a test set. According to Knight and Ruiz [36], Khachiyan and Kalantari [33] were the first to propose using Newton's method to solve a particular system of equilibration equations. Livne and Golub [42] developed an algorithm based on the Gauss-Seidel-Newton method to solve the nonlinear binormalization equations. They proved that the algorithm converges locally. In practice, only a few iterations of the algorithm are performed to obtain a vector that suitably approximates a binormalization vector. Knight and Ruiz [36] compared their algorithm, an inexact Newton method using the conjugate gradients iteration, favorably with the classical Sinkhorn-Knopp iteration and a variation of Livne and Golub's method.

Equilibration in the infinity norm is not unique and so motivates multiple algorithms that consider both efficiency and quality of the scaling under criteria other than the infinity norms of the rows and columns. A matrix can be scaled in the infinity norm if it has no zero rows or columns. The simplest algorithm is as follows: First scale the rows (or columns), then scale the columns (or rows). After the first scaling, the largest number in the matrix is 1, and the second scaling cannot produce numbers that are larger than 1. Therefore, scaling is achieved after one iteration. Bunch [11] developed an algorithm that equilibrates any symmetric matrix in the infinity norm. More recently, Ruiz [60] developed another iteration that under his criteria compares favorably with Bunch's algorithm.

Convergence results for exact scalability in which total support is an assumption immediately yield corresponding convergence results for ε-scalability. The proof of Theorem 3 shows that if A is ε-scalable, an ε-scaling vector can be obtained by exactly scaling $B + \delta I$ for sufficiently small δ. By Lemma 6, a symmetric matrix whose main diagonal is positive has total support. Therefore, if an algorithm converges for matrices having total support, then it converges for $B + \delta I$ and so produces an ε-scaling matrix for B.

Existence and uniqueness theory focuses on nonnegative matrices because equilibration in a p-norm induces an equation on a nonnegative matrix even if the original matrix is general. But equilibration algorithms must take into account sign information. Not all authors are interested in scaling general symmetric matrices, and restricting the class of matrices on which an algorithm operates can be advantageous. Most often authors restrict their attention to only nonnegative or positive definite matrices. For example, Khachiyan and Kalantari [33] focus on spd matrices.

Matrix-free algorithms

To date it appears that all scaling algorithms for general symmetric matrices require access to the elements of the matrix. If A is nonnegative, the situation is much different; for example, the Sinkhorn-Knopp algorithm requires only the matrix-vector product Ax. For general matrices, algorithms need at least matrix-vector products of the form $|A|x$ ($p = 1$), $(A \circ A)x$ ($p = 2$), or similar expressions. We introduce an approximate scaling algorithm for the case $p = 2$ that requires only the matrix-vector product Ax.

Matrix-free and stochastic matrix-free methods have a vast literature. Of course the most common matrix-free methods are Krylov-subspace iterations. Stochastic matrix-free methods in linear algebra may have a more recent origin, but stochastic methods for general problems go back at least as far as the first work of Robbins and Monro [59] on what is now called stochastic approximation. Bekas, Kokiopoulou, and Saad [1] developed a matrix-free method to estimate the diagonal elements of a matrix. Chen and Demmel [14] developed a method to balance a matrix prior to eigenvalue computations. (Balancing for eigenvalue problems is a similarity transform and so is not the same as the balancing we discuss.)

2.1.4 Our contributions

Formalizing approximate scaling by ε-scaling is useful. Livne and Golub [42] seem to be the first to have recognized the value of defining a measure of approximate scaling. But, first, they did not make concrete their definition; and, second, they left open existence and uniqueness questions. We realized that developing a definition of approximate scaling in terms of a limit can be useful, and we answer existence and uniqueness questions in terms of this definition.

Without the framework of ε-scaling, it is hard to formulate concrete statements about scaling a matrix that lacks support. For example, Parlett and Landis [57] wrote: “Note that matrices without support are not covered by [Sinkhorn and Knopp’s theorem]. Such matrices are always singular, and

V. Kahan (private communication) has shown that the sequence of iteration matrices . . . produced by [the Sinkhorn-Knopp] iteration cycles for such a starting matrix." This observation has in common with others that statements about matrices lacking support are made with reference to the Sinkhorn-Knopp iteration or algorithms that share certain properties with it. As another example, part 4 of Theorem 1 concerns explicitly the Sinkhorn-Knopp iteration. In contrast, because Lemma 3 makes a statement about ε-scaling, it assures that *no* algorithm—not just certain ones—can approximately scale a matrix lacking support.

Some statements about ε-scaling follow directly from known results. Theorem 5 identifies support as a sufficient condition for nonsymmetric ε-scaling, and the proof follows immediately from part 4 of Theorem 1. The proof of the necessary condition Lemma 3 is fairly straightforward and so probably has been used before in a different context. Still, it may be new within the context of scaling. For although the proof of Lemma 3 requires only a few minor changes to show that support is a necessary condition for exact scaling, previous authors have used Birkhoff's theorem to prove this necessary condition in the exact case. Birkhoff's theorem seems not to be applicable in the case of ε- rather than exact scaling.

We believe Theorems 3 and 6 are new and are not immediate extensions of known results to the framework of ε-scaling. Theorem 3 establishes a necessary and sufficient condition for a symmetric matrix to be ε-scalable. Just as part 1 of Theorem 1 and Theorem 2 have the same condition but require different proofs, so do Theorems 5 and 3. Theorem 6 establishes uniqueness in a certain sense for both nonsymmetric and symmetric ε-scaling. This uniqueness result is analogous to part 2 of Theorem 1 for the case of exact scaling of a matrix having total support.

Finally, the stochastic iterations (2.25) and (2.27) are to our knowledge new and appear to be the first matrix-free algorithms to scale a matrix at least approximately.

2.2 Existence and uniqueness

One reason for investigating the conditions under which a matrix is scalable is to determine when scaling can be used as a preconditioner. From this practical viewpoint, approximate rather than exact scaling is all that is required. Uniqueness of the scaled matrix—in the case of approximate scaling, uniqueness in a certain meaningful sense—tells us that a particular scaling algorithm achieves all that scaling permits by finding one set of scaling vectors: although there may be other sets of scaling vectors, there is not some other, possibly better conditioned, scaled matrix.

In this section, we prove Theorems 3 and 6. The first gives the necessary and sufficient condition for ε-scalability. The second establishes uniqueness of the approximately scaled matrix. So that the theory supporting Theorem 3 is fully contained in this chapter, we prove a result about exact scaling—Theorem 4, a condition for existence and uniqueness for exact scaling—that we shall use in proving Theorem 3; furthermore, some of the steps of the proof of Theorem 4 are used in the proof of Theorem 10. The existence part of Theorem 4 is weaker than Theorem 2—a necessary and sufficient condition for existence of an exact scaling matrix—and the uniqueness part is weaker than other theorems in the literature, but it is all we need.

What follows is based on B, not A, and so is true for all p-norms; however, we often phrase the statements of the theorems in terms of A.

2.2.1 Exactly scalable matrices

In this section, we prove the following.

Theorem 4. *If every diagonal element of the symmetric matrix A is nonzero, then A is scalable. Moreover, the scaling vector is unique.*

We say $x > 0$ is the *unique* scaling vector for the matrix A if any other scaling vector $y > 0$ is related to x by a scalar multiple.

An immediate corollary is that every positive definite matrix A has a unique scaling vector. The existence part of the theorem is equivalent to Corollary 7.7 of [9]. The primary tool of finding a function for which the scaling equation is a gradient is similar to that used in [43]. The uniqueness part is weaker than Theorem 8.2 of [9] and the theorem of Marshall and Olkin [43].

Since $x > 0$, we can multiply both sides of (2.1) by X^{-1}: $Bx = X^{-1}e$. Rearranging, we recognize that

$$g(x) \equiv Bx - X^{-1}e$$

is the gradient of the function

$$f(x) \equiv \frac{1}{2}x^T Bx - \sum_i \ln x_i. \tag{2.2}$$

The Hessian of f is

$$H(x) \equiv B + X^{-2}.$$

Lemma 1. *If $A_{ii} \neq 0$ for each i, then $f(x)$ has a minimizer $x^* > 0$.*

Proof. Let $2\beta \equiv \min_i B_{ii}$, which is positive by assumption. The first term of f is bounded below by $\beta\|x\|_2^2 > 0$. The second term is bounded as

$$-\sum_{i=1}^{n} \ln x_i \geq -n \ln \|x\|_\infty \geq -n\|x\|_\infty.$$

Therefore, as $\|x\| \to \infty$, $f \to \infty$.

Let $\chi \equiv \min_i x_i$. Because the second term of f goes to ∞ as $\chi \to 0$ and the first term is bounded below by 0, $f \to \infty$ as $\chi \to 0$.

Choose $\bar{x} > 0$, $\varepsilon > 0$ and let $\bar{f} \equiv f(\bar{x})$. Since $\lim_{\chi\to 0} f(x) = \lim_{\|x\|\to\infty} f(x) = \infty$, there exists a compact set Ω containing $\bar{x}$ on whose boundary $\partial\Omega$ $f(y)|_{y\in\partial\Omega} = \hat{f} \geq \bar{f} + \varepsilon$. Since $\bar{x} \in \Omega$, $\bar{f} = \hat{f} - \varepsilon$, and $f \in C^\infty$, Ω contains a minimizer x^* of f in its interior. Therefore, $f(x)$ has a minimizer $x^* > 0$. □

Lemma 2. *If $A_{ii} \neq 0$ for each i and $g(x) = 0$, then $H(x)$ is positive definite.*

Proof. If XHX is positive definite, then so is H. Now,

$$XHX = XBX + I.$$

If $g(x) = 0$, then $C \equiv XBX$ is doubly stochastic. $A_{ii} \neq 0$ by assumption and so $C_{ii} > 0$. By the Gershgorin theorem [28], the eigenvalues of C lie in the union of discs centered on each of the C_{ii}. As $C_{ii} > 0$ and the sum of the elements in a row is one, each disc has radius less than one. Therefore, the minimum eigenvalue $\lambda_{\min}(C) > -1$, $\lambda_{\min}(C + I) > 0$, and so $\lambda_{\min}(H) > 0$. □

Proof of Theorem 4. Since $g(x)$ is the gradient of $f(x)$, and $g(x) = 0$ if and only if x is a scaling vector for A, a minimizer of f is a scaling vector for A. By Lemma 1, a scaling vector exists.

In fact, any stationary point of f is a scaling vector for A. But Lemma 2 shows that every stationary point of f is a strong minimizer. f has only one stationary point because $f \in C^\infty$ exists on the open set $x > 0$, $\lim_{\chi\to 0} f(x) = \lim_{\|x\|\to\infty} f(x) = \infty$, and every stationary point of f is a strong minimizer. Therefore, A has a unique scaling vector. □

A matrix with at least one zero diagonal element may be scalable, but the scaling vector may not be unique. For example, consider

$$A = \begin{pmatrix} & 1 \\ 1 & \end{pmatrix}$$

and $XBx = e$. Both rows of A imply $x_1 x_2 = 1$, and so there are infinitely many scaling vectors.

2.2.2 ε-scalable matrices

In this section, we find a necessary and sufficient condition for both nonsymmetric and symmetric ε-scaling, and we show that the approximately scaled matrix is unique in a certain sense.

Examples

Let us look at a few examples of the symmetric problem. Consider the structurally rank deficient matrix

$$A = \begin{pmatrix} 1 & 1 & 1 \\ 1 & & \\ 1 & & \end{pmatrix} \tag{2.3}$$

and the scaling equation $XBx = e$. The second and third rows imply $x_1 x_2 = 1$, $x_1 x_3 = 1$; substituting these into the first row yields

$$\begin{aligned} x_1^2 + x_1 x_2 + x_1 x_3 &= 1 \\ x_1^2 + 1 + 1 &= 1 \\ x_1^2 &= -1. \end{aligned}$$

Therefore, $XBx = e$ has no real solution, but it has two bounded imaginary solutions.

In contrast, consider the structurally (and numerically) nonsingular matrix $A = \begin{pmatrix} 1 & 1 \\ 1 & \end{pmatrix}$. The second row implies $x_1 x_2 = 1$; substituting this into the first row yields

$$\begin{aligned} x_1^2 + x_1 x_2 &= 1 \\ x_1^2 + 1 &= 1 \\ x_1^2 &= 0. \end{aligned}$$

Therefore, $XBx = e$ has no solution. However, suppose we set $x_1 = \rho$ and $x_2 = \rho^{-1}$. Then

$$XBx - e = \begin{pmatrix} \rho^2 \\ 0 \end{pmatrix};$$

the residual goes to zero as $\rho \to 0$, and so A is ε-scalable.

As a more complicated example (and one that is not *diluted*—a technical condition in [42] under which a matrix is not scalable—as the previous example matrix is), consider the structurally and numerically nonsingular matrix

$$A = \begin{pmatrix} 1 & 1 & 1 \\ 1 & 1 & \\ 1 & & \end{pmatrix}.$$

The first and third rows imply $x_1^2 + x_1 x_2 = 0$, and so the equation does not have a positive solution.

However, if we set $x_1 = \rho$, $x_2 = 1$, and $x_3 = \rho^{-1}$, then the residual goes to zero as $\rho \to 0$.

Finally, it is clear that the matrix A in (2.3) is not ε-scalable. For suppose the equations implied by the second and third rows are satisfied to a residual ε. Then the first equation is

$$1 = x_1^2 + x_1 x_2 + x_1 x_3 = x_1^2 + (1 + O(\varepsilon)) + (1 + O(\varepsilon)) = x_1^2 + 2 + O(\varepsilon) \Rightarrow x_1^2 = -1 + O(\varepsilon).$$

For ε sufficiently small, x_1^2 is negative.

Results

Lemma 3. *If B is ε-scalable, then it has support.*

Proof. Suppose B lacks support but is ε-scalable.

The structural rank of B is the size of the maximum matching of the bipartite graph induced by the rows and columns. Hence if B is structurally singular, then by Hall's Theorem [34] it has a set of column indices $\mathcal{C}$ such that the matrix $B(:,\mathcal{C})$ (using MATLAB notation) contains $r < |\mathcal{C}|$ nonzero rows. Let the row indices of the nonzero rows be $\mathcal{R}$; $|\mathcal{R}| = r$.

If B is ε-scalable, then there exist $x, y > 0$ such that

$$\begin{aligned} XBy &= e + O(\varepsilon) && \text{(row equations)} \\ YB^T x &= e + O(\varepsilon) && \text{(column equations).} \end{aligned}$$

Consider the sum over the column equations $j \in \mathcal{C}$:

$$\sum_{j \in \mathcal{C}} \sum_i B_{ij} x_i y_j = |\mathcal{C}| + O(\varepsilon).$$

Each term in the lhs of this equation also appears in the sum over the row equations $i \in \mathcal{R}$, and so

$$|\mathcal{C}| + O(\varepsilon) \le \sum_{i \in \mathcal{R}} \sum_j B_{ij} x_i y_j = |\mathcal{R}| + O(\varepsilon). \tag{2.4}$$

But $|\mathcal{R}| < |\mathcal{C}|$, and so for ε sufficiently small, we have a contradiction. Hence B is not ε-scalable. □

Lemma 4. *If B has support, then it is nonsymmetrically ε-scalable.*

Proof. By part 4 of Theorem 1, the Sinkhorn-Knopp iteration converges to a doubly stochastic matrix if B has support. Let C^k be the scaled matrix at iteration k; $C \equiv \lim_{k \to \infty} C^k$ is doubly stochastic. Hence for every $\varepsilon > 0$, there is an iteration K such that for all $k > K$,

$$\|X^k B y^k - e\| \le \varepsilon \quad \text{and} \quad \|Y^k B x^k - e\| \le \varepsilon. \qquad \square$$

Combining Lemmas 3 and 4 yields the following theorem.

Theorem 5. *B is nonsymmetrically ε-scalable if and only if it has support.*

Lemma 5. *If B is ε-scalable, then so is PBQ^T for permutation matrices P and Q.*

Proof. $XBy = e + O(\varepsilon) = Pe + O(\varepsilon) = (PXP^T)(PBQ^T)(Qy)$, and similarly for YB^Tx. $\square$

Proof of Theorem 3. The necessary part of the theorem follows from Lemma 3, for if B is not nonsymmetrically ε-scalable, then it is not symmetrically ε-scalable.

Now we prove the sufficiency part. Suppose $\delta > 0$ in the equation

$$Y(B + \delta I)y = e. \tag{2.5}$$

By Theorem 4, a unique $x > 0$ satisfies this equation. By the proof of Theorem 4, $x_i(\delta)$ is a continuous function of δ because it is the unique minimizer of

$$f(x, \delta) \equiv \frac{1}{2}x^T(B + \delta I)x - \sum_i \ln x_i$$

for δ fixed, and f is a continuous function of (x, δ).

Suppose we use $x(\delta)$ as an approximate scaling vector for B. Then the residual to the scaling equation is

$$XBx - e = -\delta Xx. \tag{2.6}$$

We shall show that if B has support, then $\lim_{\delta\to 0} \delta\|Xx\| = 0$; and so for every $\varepsilon > 0$, there exists a $\delta > 0$ such that $\delta\|Xx\| \le \varepsilon$.

Let $\mathcal{I} \equiv \{i : \lim_{\delta\to 0} x_i = \infty\}$. Observe that

$$i, j \in \mathcal{I} \text{ (possibly } i = j\text{) only if } B_{ij} = 0, \tag{2.7}$$

for otherwise the term $B_{ij}x_ix_j$ would grow without bound.

Let $\mathcal{Z} \equiv \{i : B_{ij} \neq 0 \text{ and } j \in \mathcal{I}\}$. If $i \in \mathcal{Z}$, then $\lim_{\delta\to 0} x_i = 0$, for again otherwise the term $B_{ij}x_ix_j$ would grow without bound.

Let $\mathcal{B} \equiv \{i : i \notin \mathcal{I} \cup \mathcal{Z}\}$. If $i \in \mathcal{B}$, then because $i \notin \mathcal{I}$,

$$\lim_{\delta\to 0} x_i = x_i^* < \infty. \tag{2.8}$$

By Lemma 5, we can assume B is ordered such that the first $|\mathcal{I}|$ rows are the equations $i \in \mathcal{I}$ and the next $|\mathcal{Z}|$ rows are the equations $j \in \mathcal{Z}$. The blocks $B_{\mathcal{I}\mathcal{I}}$ and $B_{\mathcal{I}\mathcal{B}}$ of B are zero. The first follows from (2.7). The second is true because if $B_{ij} \neq 0$ for $i \in \mathcal{I}$, then $j \in \mathcal{Z}$ by the definition of $\mathcal{Z}$. Therefore, the only nonzero block in the first $|\mathcal{I}|$ rows is the middle one, $B_{\mathcal{I}\mathcal{Z}}$. In summary, the block structure of B is

$$B = \begin{pmatrix} & B_{\mathcal{I}\mathcal{Z}} & \\ B_{\mathcal{Z}\mathcal{I}} & B_{\mathcal{Z}\mathcal{Z}} & B_{\mathcal{Z}\mathcal{B}} \\ & B_{\mathcal{B}\mathcal{Z}} & B_{\mathcal{B}\mathcal{B}} \end{pmatrix}. \tag{2.9}$$

Corresponding to the ith row of (2.5) is equation i:

$$(B_{ii} + \delta)x_i^2 + x_i \sum_{j \neq i} B_{ij} x_j = 1.$$

Consider an equation $k \in \mathcal{I}$. We arrange the terms so that

$$x_k \sum_{j \in \mathcal{Z}} B_{kj} x_j = 1 - \delta x_k^2. \tag{2.10}$$

Summing the equations $i \in \mathcal{Z}$,

$$\sum_{i \in \mathcal{Z}} \left((B_{ii} + \delta)x_i^2 + x_i \sum_{j \neq i} B_{ij} x_j \right) = |\mathcal{Z}|, \tag{2.11}$$

where $|\mathcal{Z}|$ is the number of elements in the set $\mathcal{Z}$. Similarly, summing the equations $k \in \mathcal{I}$ in the form (2.10),

$$\sum_{k \in \mathcal{I}} x_k \sum_{j \in \mathcal{Z}} B_{kj} x_j = |\mathcal{I}| - \delta \sum_{k \in \mathcal{I}} x_k^2. \tag{2.12}$$

Because B is symmetric and has the block structure (2.9), every term in the lhs of (2.12) appears in the lhs of (2.11). Subtracting (2.12) from (2.11),

$$\sum_{i \in \mathcal{Z}} \beta_i(\delta) x_i = (|\mathcal{Z}| - |\mathcal{I}|) + \delta \sum_{k \in \mathcal{I}} x_k^2, \tag{2.13}$$

where

$$\beta_i(\delta) \equiv (B_{ii} + \delta)x_i + x_i \sum_{j \notin \{i\} \cup \mathcal{I}} B_{ij} x_j.$$

By (2.8) and the definition of $\mathcal{Z}$, $\lim_{\delta \to 0} \beta_i(\delta) = 0$; and by the definition of $\mathcal{Z}$, $\lim_{\delta \to 0} x_i = 0$. Therefore, the lhs of (2.13) converges to 0.

Consider the rhs of (2.13). There are three cases to consider:

1. If $|\mathcal{Z}| > |\mathcal{I}|$, then the rhs is bounded away from 0, which contradicts that the lhs converges to 0.

2. If $|\mathcal{Z}| < |\mathcal{I}|$, then B, and so $A \in \mathbb{R}^{n\times n}$, is structurally singular, which contradicts our assumption. For if $|\mathcal{Z}| < |\mathcal{I}|$, then $B_{\mathcal{I}\mathcal{Z}}$ is taller than it is wide, and so A is structurally singular.
3. Consequently, $|\mathcal{Z}| = |\mathcal{I}|$, and so
$$\lim_{\delta\to 0} \sum_{k\in\mathcal{I}} \delta x_k^2 = 0, \tag{2.14}$$
as is true of each term separately.

The limit (2.14) and the fact that every x_i for $i \notin \mathcal{I}$ is bounded above by definition implies that the norm of the residual (2.6) converges to 0 as $\delta \to 0$. □

Lemma 6. *If B is structurally symmetric and has a positive main diagonal, then it has total support.*

Proof. If $B_{ij} \neq 0$, then a supporting permutation is σ such that $\sigma(i) = i$ if $i \neq j$, $\sigma(i) = j$, and $\sigma(j) = i$. □

Let $\text{ts}(B)$ be such that $\text{ts}(B)_{ij} = B_{ij}$ if $B_{ij} = 0$ or B_{ij} has a supporting diagonal; hence $\text{ts}(B)$ has total support if B has support. Let $\text{pat}(B)$ be such that $\text{pat}(B_{ij}) = 1$ if and only if $B_{ij} \neq 0$.

Let B be ε-scalable. An ε-scaling algorithm computes vectors $x(\varepsilon)$ and $y(\varepsilon)$ (possibly $x(\varepsilon) = y(\varepsilon)$) for $\varepsilon > 0$ such that the vectors satisfy the ε-scaling equation.

Lemma 7. *If B is ε-scalable, then $C \equiv \lim_{\varepsilon\to 0} X(\varepsilon)BY(\varepsilon)$ is doubly stochastic.*

Proof. By the definition of ε-scaling, $X(\varepsilon)By(\varepsilon) = e + O(\varepsilon)$, and so $\lim_{\varepsilon\to 0} X(\varepsilon)By(\varepsilon) = \lim_{\varepsilon\to 0} e + O(\varepsilon) = e$; and similarly for $Y(\varepsilon)B^T x(\varepsilon)$. □

Let $u \equiv \lim_{\varepsilon\to 0} u(\varepsilon)$ for various vectors u. In some cases, elements of u are 0 or ∞.

Theorem 6. *Let B be ε-scalable and $C \equiv \lim_{\varepsilon\to 0} X(\varepsilon)BY(\varepsilon)$, where $x(\varepsilon)$ and $y(\varepsilon)$ are produced by any ε-scaling algorithm. C is the unique doubly stochastic matrix to which $\text{ts}(B)$ is diagonally equivalent.*

Proof. 1. First, C has total support by Lemma 7 and the fact that a doubly stochastic matrix has total support.

2. Second, scaling cannot introduce a nonzero C_{ij} when $B_{ij} = 0$; hence if B_{ij} lacks support, then $C_{ij} = 0$. We shall show that if $B_{ij} \neq 0$ and has support, then $C_{ij} \neq 0$. These two statements imply
$$\text{pat}(C) = \text{pat}(\text{ts}(B)). \tag{2.15}$$

3. By Lemma 5, we can assume that B and C are permuted such that
$$x(\varepsilon)_j \in O(x(\varepsilon)_i) \quad \text{and} \quad y(\varepsilon)_j \in \Omega(y(\varepsilon)_i) \quad \text{for } j \geq i; \tag{2.16}$$

that is, asymptotically $x(\varepsilon)_j$ increases no faster than $x(\varepsilon)_i$ and $y(\varepsilon)_j$ decreases no faster than $y(\varepsilon)_i$.

Consider a particular product $\chi_{ij} \equiv \lim_{\varepsilon \to 0} x(\varepsilon)_i y(\varepsilon)_j$. If $\chi_{ij} = \infty$, then $B_{ij} = 0$, for otherwise $C_{ij} = \infty$. There must be at least one pair (i,j) in every row and every column such that $0 < \chi_{ij} < \infty$, for otherwise C would lack support.

Suppose there is a pair (i,j) such that $\chi_{ij} = 0$. By (2.16), $\chi_{km} = 0$ for all pairs (k,m) such that $k \geq i$ and $m \leq j$. Similarly, suppose there is a pair (i,j) such that $\chi_{ij} = \infty$; then $\chi_{km} = \infty$ for all pairs (k,m) such that $k \leq i$ and $m \geq j$.

Given a pair (k,m) such that $\chi_{km} = 0$, we can find a pair (i,j) such that $\chi_{ij} = 0$, $\chi_{(i-1)j} \neq 0$, and $\chi_{i(j+1)} \neq 0$; for otherwise C would have a zero column or row and so lack support. As $\chi_{ij} = 0$ but $\chi_{(i-1)j} \neq 0$, $x(\varepsilon)_i \notin \Omega(x(\varepsilon)_{i-1})$; and $x(\varepsilon)_i \in O(x(\varepsilon)_{i-1})$ by (2.16). Furthermore, $x(\varepsilon)_{i-1} \in \Theta(y(\varepsilon)_j^{-1})$ and so $0 < \chi_{(i-1)j} < \infty$, for otherwise C would have a zero column and so lack support. Similarly, $y(\varepsilon)_{j+1} \neq O(y(\varepsilon)_j)$, $y(\varepsilon)_{j+1} \in \Omega(y(\varepsilon)_j)$, and $y(\varepsilon)_{j+1}^{-1} \in \Theta(x(\varepsilon)_i)$. Hence $\chi_{(i-1)(j+1)} = \infty$, and so χ has the block structure

$$\chi = \begin{pmatrix} 0 < \chi_{\mathcal{AC}} < \infty & \infty \\ 0 & 0 < \chi_{\mathcal{BD}} < \infty \end{pmatrix}$$

and so C has the block structure

$$C = \begin{pmatrix} C_{\mathcal{AC}} & \\ & C_{\mathcal{BD}} \end{pmatrix}.$$

As C has total support, each nonzero block has total support and so is square. Hence $\mathcal{A} = \mathcal{C}$ and $\mathcal{C} = \mathcal{D}$.

Now suppose $C_{ij} = 0$ but $B_{ij} \neq 0$ and has support. The pair (i,j) must be in the $(2,1)$ block of χ. Consider a permutation σ that supports B_{ij}. As $\sigma(i) = j$, there is a row $a \in \mathcal{A}$ such that $\sigma(a) \in \mathcal{B}$, and so $B_{a\sigma(a)} \neq 0$. But $\chi_{a\sigma(a)} = \infty$. That $B_{a\sigma(a)} \neq 0$ and $\chi_{a\sigma(a)} = \infty$ contradicts that $C_{a\sigma(a)} = 0$. Equation (2.15) follows.

4. Next, we show that $\mathrm{ts}(B)$ is diagonally equivalent to C. Consider a pair (i,j) such that $C_{ij} \neq 0$. As $0 < C_{ij} < \infty$, $0 < \lim_{\varepsilon \to 0} x_i(\varepsilon) y_j(\varepsilon) < \infty$. Hence there exists a function $f(\varepsilon)$ such that $x_i(\varepsilon), y_j(\varepsilon)^{-1} \in \Theta(f(\varepsilon))$. If $C_{ik} \neq 0$ and $k \neq j$, $y_k(\varepsilon)^{-1} \in \Theta(f(\varepsilon))$, for otherwise $0 < C_{ik} < \infty$ does not hold; and similarly for C_{kj} and $k \neq i$. Hence there are index sets $\mathcal{M}$ and $\mathcal{N}$ such that $x_i(\varepsilon) \in \Theta(f(\varepsilon))$ for $i \in \mathcal{M}$ and $y_j(\varepsilon)^{-1} \in \Theta(f(\varepsilon))$ for $j \in \mathcal{N}$. Let $\hat{x}_i(\varepsilon) \equiv f(\varepsilon)^{-1} x_i(\varepsilon)$ for $i \in \mathcal{M}$ and $\hat{y}_j(\varepsilon) \equiv f(\varepsilon) y_j(\varepsilon)$ for $j \in \mathcal{N}$. Then $0 < \hat{x}_i < \infty$ and $0 < \hat{y}_j < \infty$. Furthermore, $\hat{x}_i(\varepsilon)\hat{y}_j(\varepsilon) = f(\varepsilon)^{-1} x_i(\varepsilon) f(\varepsilon) y_j(\varepsilon) = x_i(\varepsilon) y_j(\varepsilon)$, and so $B_{ij}\hat{x}_i\hat{y}_j = C_{ij}$.

We can repeat this process for all pairs (i,j) such that $C_{ij} \neq 0$ and $\hat{x}_i$, $\hat{y}_j$ have yet to be defined. Once $\hat{x}_i$, $\hat{y}_j$ are defined for every i and j, we have scaling vectors $\hat{x}$, $\hat{y}$ such that, recalling (2.15), $\hat{X}\,\mathrm{ts}(B)\hat{Y} = C$.

5. Finally, by part 2 of Theorem 1, C is the unique doubly stochastic matrix to which $\text{ts}(B)$ is diagonally equivalent. □

2.2.3 Symmetric positive definite matrices

Although we largely omit a discussion of conditioning results, the case of spd matrices will be important to us when we consider quasi-Newton methods.

We call symmetric scaling of A such that the result has all unit diagonal elements *Jacobi scaling*. If A is spd, then Jacobi scaling is always possible, as every diagonal element is positive.

Van der Sluis [63] showed the following. Let A be an $n \times n$ Hermitian positive definite matrix. Suppose all the main diagonal elements are 1. Let $\kappa(\cdot)$ denote the condition number of a matrix. Then $\kappa(A) \le n \min_d \kappa(\text{diag}(d) A \,\text{diag}(d))$ (Theorem 4.1 of [63]). Additionally, if A has at most q nonzero elements in any row, then $\kappa(A) \le q \min_d \kappa(\text{diag}(d) A \,\text{diag}(d))$ (Theorem 4.3 of [63]).

Quite a bit earlier, Forsthye and Straus [21] proved the following tighter result for a class of Hermitian p.d. matrices. A matrix B has *Young's property* A if there exists a permutation matrix P such that

$$PBP^T = \begin{pmatrix} D_1 & B_1 \\ B_2 & D_2 \end{pmatrix},$$

where D_1 and D_2 are square diagonal matrices. If a Hermitian p.d. matrix A has Young's property A, then $\kappa(A) \le \min_d \kappa(\text{diag}(d) A \,\text{diag}(d))$ (Theorem 4 of [21]).

In summary, in these three theorems Jacobi scaling is within a factor of n, q, or 1 of optimal among all diagonal scaling matrices. Since Jacobi scaling is evidently quite effective on spd matrices, we should investigate by how much equilibration differs from Jacobi scaling.

If A is spd, then so is $B \equiv A \circ A$ by the Schur Product Theorem (see, for example, Theorem 7.5.3 of [28]). Suppose A has unit diagonal elements. Then so does B. Moreover, if $i \ne j$, then $B_{ij} < 1$. For suppose $B_{ij} \ge 1$. Let v be the vector such that $v_i = 1$, $v_j = -1$, and $v_k = 0$ for all other elements. Then $v^T B v = 2 - 2B_{ij} \le 0$, which contradicts that B is spd.

Suppose Jacobi scaling has been applied to an $n \times n$ symmetric matrix $\bar{A}$ to yield a matrix A, and again let $B \equiv A \circ A$. If $\bar{A}$ is indefinite and has a zero diagonal element, we set the corresponding element in the scaling vector to 1. Consider the vector of row sums $b \equiv Be$. If $\bar{A}$ is indefinite, $0 \le b_i < \infty$. If $\bar{A}$ is p.d., as every diagonal element of B is 1, $b_i \ge 1$; and as every off-diagonal element $B_{ij} < 1$, $b_i < n$.

Let $\text{var}(v)$ be the variance of an n-vector v: $\text{var}(v) \equiv n^{-1} \sum_i (v_i - \mu(v))^2$, where μ is the mean of the elements. If a matrix is binormalized, then the variance of the vector of its row norms is 0. If $\bar{A}$ is indefinite, $\text{var}(Be)$ can be arbitrarily large. But if A is p.d., we have the following result.

Theorem 7. *If A is spd and has all unit diagonal elements, then $var(Be) < (n-1)^2$.*

Proof. As each $1 \le b_i = (Be)_i < n$, $(b_i - \mu(b))^2 < (n-1)^2$. Therefore, $n^{-1}\sum_i (b_i - \mu(b))^2 < n^{-1}\sum_i (n-1)^2 = (n-1)^2$. $\square$

From the other direction, an immediate corollary of some results in [41] is that if an spd matrix $\bar{A}$ is equilibrated in the 1-norm to form A, then $n^{-1} < A_{ii} \le 1$ (the upper bound follows immediately from equilibration to unit row and column 1-norms); if A is indefinite, of course, then $-1 \le A_{ii} \le 1$.

Theorem 7 shows that when A is spd, Jacobi scaling produces a matrix that is not arbitrarily far from being binormalized. This observation suggests that the two methods of scaling condition spd matrices comparably. Numerical experiments in Section 2.4 compare the conditioning each method yields on a set of spd matrices; the results are indeed quite similar.

2.3 Matrix-free approximate symmetric equilibration

Exact scaling algorithms for general matrices require access to the matrix elements. We develop approximate algorithms that access the matrix only through matrix-vector products.

Our analysis in Section 2.2 shows that every structurally nonsingular matrix can be ε-scaled. The degree of approximation is measured by ε, which quantifies how much the scaled matrix deviates from being doubly stochastic. In a number of algorithms [42, 57, 62, 36], one can measure this error at each iteration and terminate when it falls below ε.

As we focus on the problem in which one does not have direct access to the elements of the matrix, our algorithms cannot monitor the error. In the absence of any error information, our algorithms run for the number of iterations specified by the user and then terminate.

2.3.1 The symmetric Sinkhorn-Knopp iteration

Consider a nonnegative $n \times n$ matrix B. Let r and c be positive n-vectors. Recall our convention that r^{-1} is the element-wise reciprocal of r. The Sinkhorn-Knopp iteration is as follows:

$$\begin{aligned} r^{k+1} &= (Bc^k)^{-1} \\ c^{k+1} &= (B^T r^{k+1})^{-1}. \end{aligned}$$

By Theorem 1, if B has total support, then the limits $r \equiv \lim_{k\to\infty} r^k$ and $c \equiv \lim_{k\to\infty} c^k$ exist and RBC is doubly stochastic; and if B is fully indecomposable, r and c are unique up to a scalar multiple.

The symmetric Sinkhorn-Knopp iteration is

$$x^{k+1} = (Bx^k)^{-1}.$$

In this iteration, x alternately takes the role of r and c in the nonsymmetric iteration: in particular, $x^{2k} = r^k$ and $x^{2k+1} = c^k$. Knight [35] discusses similar ideas. We shall see that because of the simple relationship between x, r, and c, in general x^k oscillates between two sequences.

A matrix A is *reducible* if there exists a permutation matrix P such that

$$PAP^T = \begin{pmatrix} B & \\ C & D \end{pmatrix},$$

where B and D are square. Otherwise A is *irreducible*. Observe that if A is symmetric, then PAP^T is block diagonal and symmetric.

Theorem 8. *If the symmetric nonnegative matrix B is irreducible and fully indecomposable, then the symmetric Sinkhorn-Knopp iteration converges to x such that αXBX is doubly stochastic for all positive starting vectors, where $\alpha > 0$ is a scalar.*

Proof. If B is fully indecomposable, then B has total support [8].

By part 4 of Theorem 1, as B has support, the nonsymmetric Sinkhorn-Knopp iteration converges for all positive starting vectors. By part 3 of the same theorem, if B is fully indecomposable, the scaling matrices D_1 and D_2 are unique up to a scalar multiple. As both D_1BD_2 and $(D_1BD_2)^T$ are doubly stochastic and D_1 and D_2 are unique up to a scalar multiple, $d_1 \propto d_2$. Therefore, D_1BD_1 and D_2BD_2 are scalar multiples of each other and of a doubly stochastic matrix.

In the nonsymmetric Sinkhorn-Knopp iteration, let us associate the following vectors: $x^{2k} = r^k$ and $x^{2k+1} = c^k$. As the nonsymmetric iteration converges, let $r \equiv \lim_{k\to\infty} r^k$ and $c \equiv \lim_{k\to\infty} c^k$. By our previous comments, $r \propto c$ and RBC is doubly stochastic. Let $r = \beta c$. Then $RBR = \beta RBC$ and $CBC = \beta^{-1}CBC$. If the iteration terminates for k odd, the constant $\alpha = \beta$; if even, $\alpha = \beta^{-1}$. $\square$

The theorem does not hold if B is reducible. Consider the matrix $B = \text{diag}(1\ 2)^T$. If $x^0 = e$, the even iterates remain e while the odd iterates immediately converge to $v \equiv (1\ 1/2)^T$. Of course IBV equilibrates B. The symmetric equilibration vector is $(1\ 1/\sqrt{2})^T = \sqrt{ev}$.

That the symmetric Sinkhorn-Knopp may not converge on reducible matrices is not a problem. The blocks of the reducible matrix B decouple the problem, and one can construct a symmetric equilibrating vector x from the nonsymmetric equilibrating vectors r and c. Suppose r and c equilibrate B by RBC. Let $q \equiv c/r$, and let $\mathcal{I}$ be the indices of a subset of identical elements of q. If RBC is doubly stochastic, so is $(\alpha R)B(\alpha^{-1}C)$ for a scalar α. Set $\alpha = \sqrt{c/r}$ so that $\alpha r(\mathcal{I}) = \alpha^{-1}c(\mathcal{I}) = \sqrt{r(\mathcal{I})c(\mathcal{I})} \equiv x(\mathcal{I})$. Repeating this procedure for each such subset of indices yields the symmetric equilibration vector x.

Our algorithms are motivated by the symmetric Sinkhorn-Knopp iteration. They average iterates and so do not exhibit the oscillation we have just discussed.

2.3.2 Stochastic binormalization

The symmetric Sinkhorn-Knopp algorithm uses the matrix-vector product Bx. If A is a general symmetric matrix, then $B_{ij} = |A_{ij}|^p$ for $p \geq 1$, and so B is not available if one does not have access to the elements of A. How can we compute Bx, at least approximately, by using a matrix-vector product with A rather than B?

Let a be a vector, possibly random. Suppose a and the random vector u are independent.

Lemma 8. *If the elements of $u \in \mathbb{R}^n$ have zero mean, positive and finite variance, and are iid, then*

$$E(a^T u)^2 = \eta E a^T a. \tag{2.17}$$

for finite $\eta > 0$.

Proof. Because $\mathrm{E}\, u_i u_j = 0$ if $i \neq j$, $\mathrm{E}\left(\sum_j a_j u_j\right)^2 = \mathrm{E}\sum_j a_j^2 u_j^2 = \eta \sum_j a_j^2$, where $\eta = \mathrm{E}\, u_j^2 > 0$ is finite. □

For use in Chapter 3, let f be the pdf of each element of u. Suppose f is symmetric around zero. Then normalization is permitted, as demonstrated in the following lemma.

Lemma 9. *If the elements of $u \in \mathbb{R}^n$ have positive and finite variance and are iid with pdf f, and f is symmetric around zero, then*

$$E\frac{(a^T u)^2}{u^T u} = n^{-1} E a^T a.$$

Proof. A cross term now has the form

$$\mathrm{E}\,\frac{u_1 u_2}{\sum_j u_j^2}. \tag{2.18}$$

We evaluate the integral as follows:

$$\begin{aligned}
\mathrm{E}\,\frac{u_1 u_2}{\sum_j u_j^2} &= \int_{\mathbb{R}^n} \frac{u_1 u_2}{\sum_j u_j^2} \prod_j f(u_j)\; \mathrm{d}u_j \\
&= \int_{\mathbb{R}^{n-1}} u_2 \prod_{j>1} f(u_j) \left(\int_{-\infty}^{\infty} \frac{u_1}{\sum_j u_j^2} f(u_1)\; \mathrm{d}u_1 \right) \prod_{j>1} \mathrm{d}u_j.
\end{aligned}$$

The integrand in the inner integral is antisymmetric around zero and so the inner integral is zero. Therefore, (2.18) is also zero.

As the cross terms are again zero,

$$\mathrm{E}\,\frac{\left(\sum_j a_j u_j\right)^2}{\sum_j u_j^2} = \mathrm{E}\,\frac{\sum_j a_j^2 u_j^2}{\sum_j u_j^2} = \eta \mathrm{E} \sum_j a_j^2, \quad \text{where} \quad \eta = \mathrm{E}\,\frac{u_i^2}{\sum_j u_j^2};$$

and as

$$n\eta = \sum_i \mathrm{E}\, \frac{u_i^2}{\sum_j u_j^2} = \mathrm{E}\, \frac{\sum_i u_i^2}{\sum_j u_j^2} = 1,$$

we have $\eta = n^{-1}$. □

A vector u whose elements are iid normal random variables obeys the conditions of Lemma 9. Such a vector samples uniformly from the unit sphere [47].

If a is complex valued, both Lemmas 8 and 9 generalize simply by considering $(\bar{a}^T u)(a^T u)$ rather than $(a^T u)^2$. We present our algorithms using notation suitable only for real-valued matrices, but they immediately extend to complex-valued matrices.

Returning to our problem, we can approximate Bx when $p = 2$ by computing the matrix-vector product $AX^{1/2}u$, where u obeys the assumptions of either Lemmas 8 or 9. By Lemma 8, for example,

$$\mathrm{E}\,(AX^{1/2}u)^2 = \eta(AX^{1/2})^2 e = \eta(A \circ A)Xe = \eta Bx. \tag{2.19}$$

To increase the accuracy of the estimate, one could compute the mean of multiple matrix-vector products $AX^{1/2}u$. Then one could construct an approximate scaling algorithm by replacing the exact computation Bx with this estimate in the symmetric Sinkhorn-Knopp algorithm.

However, the methods of *stochastic approximation* suggest a better approach. In stochastic approximation, the exact iteration

$$x^{k+1} = x^k + \omega^k f(x^k) \tag{2.20}$$

is replaced by the stochastic iteration

$$x^{k+1} = x^k + \omega^k \hat{f}(x^k),$$

where $\mathrm{E}\,\hat{f}(x^k) = f(x^k)$ [38]. If several conditions are satisfied—some of which are not straightforward to verify—then the second iteration converges with probability 1 to a limit point of the first.

Recall that the symmetric Sinkhorn-Knopp iteration is $x^+ = (Bx)^{-1}$. A scalar does not matter, and so we can use the iteration $x^+ = \|Bx\|_1^{-1}(Bx)^{-1}$. Let us rewrite the latter as follows. First, $d = x^{-1}$. Second, let $\omega = 1$. Then

$$d^+ = \frac{Bx}{\|Bx\|_1} = (1-\omega)\frac{d}{\|d\|_1} + \omega\frac{Bx}{\|Bx\|_1}.$$

If $\omega \neq 1$, we have the new iteration

$$d^{k+1} = (1-\omega)\frac{d^k}{\|d^k\|_1} + \omega\frac{Bx^k}{\|Bx^k\|_1}. \tag{2.21}$$

The iteration takes a convex combination of the current d and the Sinkhorn-Knopp update of d when each of the two terms is normalized by its 1-norm and $0 < \omega < 1$. Furthermore, d^k in (2.21) is related to $\hat{d}^k$ in the iteration

$$\hat{d}^{k+1} = \hat{d}^k + \omega \left(-\hat{d}^k + \frac{\|\hat{d}^k\|_1}{\|B\hat{x}^k\|_1} B\hat{x}^k \right)$$

by a scalar, and the latter iteration has the form of (2.20) if ω is replaced by the sequence ω^k. We shall consider the stochastic iteration corresponding to (2.21) after we discuss some convergence properties of the latter.

Sinkhorn and Knopp [62] proved that their iteration converges globally if the matrix has support; Parlett and Landis [57] generalized the global convergence proof to all iterations that satisfy certain conditions; and Theorem 8 proves the symmetric Sinkhorn-Knopp iteration converges globally if the symmetric matrix B is irreducible and fully indecomposable. It appears that none of the methods of proof can be extended to show the global convergence of the iteration (2.21) for two principal reasons: our iteration is for the symmetric problem, and nonsymmetric scaling is essential to some of the steps of the classic proofs; and the additional term $(1-\omega)d$ means row or column stochasticity is not alternately achieved. However, we have the following partial results.

Lemma 10. *The iteration* (2.21)*, with* $0 < \omega \leq 1$*, has a fixed point* $d^* > 0$ *if and only if* B *has total support.*

Proof. Suppose the iteration (2.21) has a fixed point d^*:

$$d^* = G(d^*) \equiv (1-\omega)\frac{d^*}{\|d^*\|_1} + \omega \frac{Bx^*}{\|Bx^*\|_1}.$$

Let $s^* \equiv \|Bx\|_1^{-1}$. d^* is a convex combination of two unit-1-norm vectors and so itself has unit 1-norm. Hence

$$\begin{aligned} d^* &= (1-\omega)d^* + \omega s^* Bx^* \\ s^* X^* B x^* &= e, \end{aligned} \tag{2.22}$$

and so d^* is a scaling vector. As B is scalable, it has total support.

From the other direction, by Theorem 2, $x > 0$ exists such that $XBx = ce$, $c > 0$, and $\|d\|_1 = 1$, if B has total support. Then $Bx = cd$. Substituting this expression into (2.21) shows that d is a fixed point. □

Theorem 9. *If* A *is symmetric and has total support, then a fixed point* $d^* > 0$ *of* (2.21) *is a point of attraction for* $0 < \omega \leq 1$.

We shall require some supporting results.

Lemma 11. *Let $A \in \mathbb{R}^{n\times n}$ be symmetric, and let $\{\lambda_i, v_i\}$ be its eigenvalues and eigenvectors. For $a \in \mathbb{R}^n$, the eigenvalues of $A - v_1 a^T$ are $\{\lambda_1 - a^T v_1, \lambda_2, \dots, \lambda_n\}$.*

Proof. A is diagonalized as $A = V\Lambda V^T$. $A - v_1 a^T$ is similar to

$$V^T(A - v_1 a^T)V = \Lambda - e_1 b^T,$$

where $b \equiv V^T a$. The matrix on the rhs is upper triangular and so its eigenvalues lie on its diagonal. Only the first eigenvalue λ_1 is altered: it is λ_1 minus the first element of b, which is just $a^T v_1$. □

Theorem 10.1.3 of [55]. Suppose that $G : D \subset \mathbb{R}^n \to \mathbb{R}^n$ has a fixed point x^* in the interior of D and is Frechet-differentiable at x^*. If the spectral radius of $G'(x^*)$ satisfies $\rho(G'(x^*)) = \sigma < 1$, then x^* is a point of attraction of the iteration $x^{k+1} = G(x^k)$.

Lemma 12. *The iteration* (2.21) *is invariant to symmetric permutations.*

Proof. Let P be a permutation matrix. Then

$$\begin{aligned} \|d\|_1 &= \|Pd\|_1 \\ \|Bx\|_1 &= \|(PBP^T)(Px)\|_1. \end{aligned}$$

Hence if d^k and d^{k+1} satisfy (2.21), then

$$Pd^{k+1} = (1-\omega)\frac{Pd^k}{\|Pd^k\|_1} + \omega\frac{(PBP^T)(Px^k)}{\|(PBP^T)(Px^k)\|_1},$$

and so Pd^k and Pd^{k+1} satisfy (2.21) for the matrix PBP^T. □

Proof of Theorem 9. First we assume A is irreducible.

By Theorem 10.1.3 of [55], the theorem follows for irreducible A if $\rho(\nabla_d G(d)|_{d=d^*}) < 1$. The Jacobian is

$$\begin{aligned} \nabla_d G(d) &= \nabla_x G(d)\nabla_d x = -\nabla_x G(d) X^2 \\ &= \frac{1-\omega}{\|d\|_1} I - \frac{1-\omega}{\|d\|_1^2} de^T - \frac{\omega}{\|Bx\|_1} BX^2 + \frac{\omega}{\|Bx\|_1^2} Bxe^T BX^2. \end{aligned}$$

By Lemma 10, a fixed point $x = x^* > 0$ is a scaling vector. At such a point, $Bx = s^{-1}d$ by (2.22) and $\|d\|_1 = 1$. Hence

$$\nabla_d G(d)|_{d=d^*} = (1-\omega)I - (1-\omega)de^T - \omega s BX^2 + \omega s^2 Bxe^T BX^2.$$

Noting that $Xd = e$, we use X to obtain the similar matrix

$$X \nabla_d G(d)|_{d=d^*} X^{-1} = (1-\omega)I - (1-\omega)ed^T + \omega s XBX - \omega s^2 XBxe^T BX. \tag{2.23}$$

As $XBx = s^{-1}e$ and $e^T = d^T X$, $XBxe^T BX = s^{-1}ee^T BX = s^{-1}ed^T XBX$, and this is substituted into the fourth term of (2.23). Combining the second and fourth terms, we obtain the rank-one matrix

$$Q \equiv e[(\omega - 1)d^T - \omega s d^T XBX],$$

which has the nonzero eigenvalue $[(\omega-1)d^T - \omega s d^T XBX]e$. As $sd^T XBXe = se^T Bx = se^T s^{-1} d = e^T d$, the eigenvalue is $(\omega - 1)d^T e - \omega e^T d = -e^T d = -\|d\|_1 = -1$.

Because $sXBx = e$ and B is symmetric, $C \equiv sXBX$ is doubly stochastic. First, $\rho(C) \leq 1$. Second, we already showed in the proof of Lemma 2 that $\lambda_{\min}(C) > -1$. Since C is symmetric, the only eigenvalue having unit magnitude is 1. As for the moment we assume A is irreducible, by the Perron-Frobenius theorem (see, for example, Theorem 8.4.4 in [28]), the eigenvalue 1 is simple.

Next, the rank-one matrix Q has the right eigenvector e. e is also a right eigenvector of C, associated with the eigenvalue 1. Therefore, by Lemma 11, $\bar{C} \equiv C + Q$ has the same eigenvalues as C, except that the single eigenvalue 1 is now 0. We conclude that $\rho(\bar{C}) < 1$; therefore, $\rho((1-\omega)I + \omega\bar{C}) < 1$ as well, and so $\rho(\nabla_d G(d)|_{d=d^*}) < 1$, as desired.

Now suppose A is reducible. By Lemma 12, we can assume A is block diagonal and each block is irreducible. Our analysis to this point applies to each block separately. As the Jacobian associated with each block has spectral radius less than 1, so does the Jacobian as a whole. □

Additionally, we can show global convergence for the simpler iteration

$$d^{k+1} = d^k + \alpha^k(-d^k + Bx^k) = d^k + \alpha^k h(d^k), \tag{2.24}$$

where α^k is an appropriate step size. One way this iteration arises is as follows. Suppose we solve the nonlinear equation $f(d) \equiv d - Bx = 0$ using Newton's method. The Jacobian is $J \equiv f_d(d) = I + BX^2$, and so Newton's method yields the iteration

$$(I + BX^2)(d^{k+1} - d^k) = -d^k + Bx^k.$$

If we approximate the Jacobian by $J \approx I$ and introduce a step size, we obtain (2.24). However, we treat the latter as a minimization method so that we can use certain techniques in our proof.

We use the following theorem.

Propositions 1.2.1 and 1.2.2 of [2]. Let $\{d^k\}$ be a sequence generated by a gradient method

$d^{k+1} = d^k + \alpha^k h^k$, and assume that $\{h^k\}$ is gradient related and α^k is chosen by the minimization rule, the limited minimization rule, the Goldstein rule, or the Armijo rule. Then every limit point of $\{d^k\}$ is a stationary point.

Various conditions assure $\{g^k\}$ is gradient related; we use the following: $h^k = -H^k \nabla f(d^k)$, and the eigenvalues of the positive definite symmetric matrix H^k are bounded above and below: $\lambda_{\min}(H^k) \geq c_1$ and $\lambda_{\max}(H^k) \leq c_2$.

Theorem 10. *If A is symmetric and $A_{ii} \neq 0$, then the iteration (2.24) converges globally if α^k is chosen according to one of the step size rules in Propositions 1.2.1 or 1.2.2 of [2].*

Proof. Lemmas 1 and 2 show that the global minimizer of the function $f(x)$ defined in (2.2) is a scaling vector. To review, we define

$$\begin{aligned} f(x) &\equiv \frac{1}{2} x^T B x - \sum_i \ln x_i \\ g(x) &\equiv \nabla_x f(x) = Bx - d \\ H(x) &\equiv \nabla_x^2 f(x) = B + D^2. \end{aligned}$$

f and its derivatives are defined with respect to x rather than d; in contrast, the iteration (2.24) is defined with respect to d. f and its derivatives are transformed as follows:

$$\begin{aligned} f_d(d) &\equiv \frac{1}{2} x^T B x - \sum_i \ln x_i = f(x) \\ g_d(d) &\equiv \nabla_d f(d) = -X^2 g(x) = -X^2 B x + x \\ H_d(d) &\equiv \nabla_d^2 f(d) = 2X^3 \operatorname{diag}(g(x)) + X^2 H(x) X^2. \end{aligned}$$

If $H(x)$ is positive definite (as Lemma 2 shows), then so is $H_d(d)$: $X^2 H(x) X^2 \succ 0$ because a p.d. matrix remains p.d. if pre- and post-multiplied by symmetric (in this case, diagonal) matrices; and $2X^3 \operatorname{diag}(g(x))$ is a p.d. diagonal matrix.

In the final paragraph of the proof to Lemma 1, we observe that given $\bar{x} > 0$, there exists a compact set Ω containing $\bar{x}$. As the minimizer of f lies in Ω, the iteration (2.24) has a limit point. In the same proof we observe that $f \to \infty$ as any $x_i \to 0$ or $x_i \to \infty$. Hence $\Omega \subset \{x : x > 0\}$. Let $\bar{x}$ be the initial point of the iteration. Let us define the following quantities relative to the boundary of Ω:

$$d_L \equiv \inf_{x \in \partial\Omega} \min_i d_i \quad \text{and} \quad d_U \equiv \sup_{x \in \partial\Omega} \max_i d_i.$$

As any step size rule in Propositions 1.2.1 and 1.2.2 of [2] is such that $f^{k+1} < f^k$, the iterates remain in Ω. Hence for each iterate d and element d_i, $d_L < d_i < d_U$ and so the eigenvalues of the matrix D^2 are bounded from below and above by d_L^2 and d_U^2 respectively. Therefore, as $-D^2 g_d(d) = g(x) = Bx - d = h(d)$, $h(d)$ is gradient related. □

Substituting (2.19) into (2.21) and replacing ω with ω^k, we obtain the stochastic iteration

$$\begin{aligned} y^k &= (A(X^k)^{1/2}u^k)^2 \\ d^{k+1} &= (1-\omega^k)\frac{d^k}{\|d^k\|_1} + \omega^k \frac{y^k}{\|y^k\|_1}. \end{aligned} \tag{2.25}$$

Based on numerical experiments, we choose ω^k according to the rule

$$-\log_2 \omega^k = \max(\min(\lfloor \log_2(k) \rfloor - 1, 4), 1). \tag{2.26}$$

Consequently, $1/16 \le \omega^k \le 1/2$ for all k, and ω^k decreases by a factor of 2 at particular values of k. This sequence for ω^k allows for large changes in $d/\|d\|_1$ when k is small and smaller changes when k is large. Our numerical experiments show that the maximum number of iterations is a small fraction of the size of the matrix; otherwise, the algorithm would not be useful.

The iteration (2.25) is straightforwardly generalized to the nonsymmetric problem. Let ρ and γ be such that $\rho_i = r_i^{-1}$ and $\gamma_i = c_i^{-1}$. Let v be a second random vector. Each iteration now performs a forward and a transpose matrix-vector product:

$$\begin{aligned} y^k &= (A(C^k)^{1/2}u^k)^2 \\ \rho^{k+1} &= (1-\omega^k)\frac{\rho^k}{\|\rho^k\|_1} + \omega^k \frac{y^k}{\|y^k\|_1} \\ z^K &= (A^T(R^k)^{1/2}v^k)^2 \\ \gamma^{k+1} &= (1-\omega^k)\frac{\gamma^k}{\|\gamma^k\|_1} + \omega^k \frac{z^k}{\|z^k\|_1}. \end{aligned} \tag{2.27}$$

We omit analysis of both the corresponding deterministic and this iteration because our results would again be at best partial. But we show the results of extensive numerical experiments in Section 2.4. For the problem of square nonsymmetric matrices, Theorems 5 and 6 completely characterize the class of ε-scalable matrices; but we have not investigated the existence and uniqueness theory for rectangular matrices.

Using the convergence theory of stochastic approximation (see, *e.g.*, Chapters 4–6 of [38]), it may be possible to show that if ω^k is a sequence obeying certain conditions (in particular, $\lim_{k\to\infty} \omega^k = 0$, and so the sequence (2.26) does not conform), then the sequences (2.25) and (2.27) converge with probability 1 to scaling vectors. We have not succeeded in this analysis for these or similar iterations; the analysis is quite technical and takes us far afield. In any case, our algorithm is practical only if a small number of iterations succeeds in conditioning the matrix sufficiently, and so theoretical

```
function x = ssbin(A,nmv,n)
% Stochastic matrix-free binormalization for symmetric real A.
% x = ssbin(A,nmv,n)
%   A is a symmetric real matrix or function handle. If it is a
%     function handle, then v = A(x) returns A*x.
%   nmv is the number of matrix-vector products to perform.
%   [n] is the size of the matrix. It is necessary to specify n
%     only if A is a function handle.
%   diag(x) A diag(x) is approximately binormalized.
  op = isa(A,'function_handle');
  if(~op) n = size(A,1); end
  d = ones(n,1);
  for(k = 1:nmv)
    u = randn(n,1);
    s = u./sqrt(d);
    if(op) y = A(s); else y = A*s; end
    omega = 2^(-max(min(floor(log2(k))-1,4),1));
    d = (1-omega)*d/sum(d) + omega*y.^2/sum(y.^2);
  end
  x = 1./sqrt(d);
```

Figure 2.1: MATLAB implementation of the symmetric stochastic binormalization algorithm SSBIN.

results that apply in the limit $k \to \infty$ are not necessarily relevant. In the end, we have chosen to motivate our algorithms by examining a deterministic sequence and then demonstrate their efficacy on a large test set, to the results of which we now turn.

2.4 Numerical experiments

Framework

Figure 2.1 shows a MATLAB implementation of SSBIN, the stochastic binormalization algorithm for symmetric matrices corresponding to the sequence (2.25); and Figure 2.2 shows an implementation of SNBIN, the stochastic binormalization algorithm for nonsymmetric matrices corresponding to the sequence (2.27).

We test our algorithms in MATLAB on problems in the University of Florida Sparse Matrix Collection [17]. We use the following MATLAB code to obtain matrices obeying our criteria:

```
  index = UFget('refresh');
  % Symmetric
  sids = find(index.nnz <= 1e7 & ~index.isBinary &...
              index.numerical_symmetry == 1 &...
              index.sprank == index.nrows & index.isReal);
  % Square nonsymmetric
  nids = find(index.nnz <= 1e7 & ~index.isBinary &...
              index.nrows == index.ncols &...
              index.sprank == index.nrows & index.isReal);
  % Rectangular
```

```
function [x y] = snbin(A,nmv,m,n)
% Stochastic matrix-free binormalization for nonsymmetric real A.
% [x y] = snbin(A,nmv,m,n)
%   A is a matrix or function handle. If it is a function handle, then
%     v = A(x) returns A*x and v = A(x,'trans') returns A'*x.
%   nmv is the number of matrix-vector products to perform.
%   m,n is the size of the matrix. It is necessary to specify these only if
%     A is a function handle.
%   diag(x) A diag(y) is approximately binormalized.
  op = isa(A,'function_handle');
  if(~op) [m n] = size(A); end
  r = ones(m,1); c = ones(n,1);
  for(k = 1:nmv)
    omega = 2^(-max(min(floor(log2(k))-1,4),1));
    s = randn(n,1)./sqrt(c);
    if(op) y = A(s); else y = A*s; end
    r = (1-omega)*r/sum(r) + omega*y.^2/sum(y.^2);
    s = randn(m,1)./sqrt(r);
    if(op) y = A(s,'trans'); else y = (s'*A)'; end
    c = (1-omega)*c/sum(c) + omega*y.^2/sum(y.^2);
  end
  x = 1./sqrt(r);
  y = 1./sqrt(c);
```

Figure 2.2: Matlab implementation of the nonsymmetric stochastic binormalization algorithm snbin.

```
rids = find(index.nnz <= 1e7 & ~index.isBinary &...
            index.nrows ~= index.ncols &...
            index.sprank == min(index.nrows,index.ncols) & index.isReal);
```

The predicate `index.sprank == index.nrows` assures that the matrices have full structural rank, the condition Theorems 3 and 5 give for a square matrix to be ε-scalable. (But recall that we do not have a theorem to support rectangular matrices.) At the time of our experiments, only three complex-valued matrices in the collection have full structural rank and are symmetric, and so we focus on only real-valued matrices.

Our primary metric for assessing the performance of the algorithms is the reduction in condition number of the matrix. A secondary metric, used to verify that our algorithms indeed approximately equilibrate matrices, is based on *normalized variance*. The normalized variance of the vector v is

$$\mathrm{nvar}(v) \equiv \frac{\|v-\mu\|_2^2}{\|v\|_2^2} = n\frac{\mathrm{var}(v)}{\|v\|_2^2},$$

where μ is the mean of the elements of v and $\mathrm{var}(v)$ is their variance. A useful property is that $0 \le \mathrm{nvar}(v) \le 1$. For symmetric matrices, we use the normalized variance of the row 2-norms, denoted NVR or NVR(A). For nonsymmetric matrices, we use the maximum of the normalized variance of the row and column 2-norms, denoted MVR or MVR(A).

To evaluate the condition number of a matrix A, we use the Matlab function `scond` shown in Figure 2.3. This function uses the sparse methods `eigs` and `svds` to obtain the extremal singular

```
function c = scond(A,issym)
% cond for sparse matrices.
% c = scond(A,issym)
%   A is a sparse matrix.
%   issym is a boolean: true if A is symmetric, false otherwise.
  if(nargin < 2) issym = false; end
  s1 = []; sn = [];
  opts = struct('maxit',1e3,'disp',0,'issym',issym,'isreal',true);
  [m n] = size(A);
  try
    if(issym) % Symmetric
      s1 = abs(eigs(A,1,'LM',opts));
      sn = abs(eigs(A,1,'SM',opts));
    elseif(m == n) % Nonsymmetric square
      s1 = svds(A,1,'L',opts);
      sn = svds(A,1,0,opts);
    else % Rectangular
      s1 = svds(A,1,'L',opts);
      if(m < n) B = A*A'; else B = A'*A; end
      opts.issym = true;
      sn = sqrt(eigs(B,1,'SM',opts));
    end
  catch
  end
  if(isempty(s1) || isempty(sn)) c = 0; else c = s1/sn; end
```

Figure 2.3: MATLAB function to compute the condition number of the sparse matrix A.

values of a matrix. To find the smallest singular value in the thin SVD of the rectangular matrix A, `scond` forms the product $B = AA^T$ or A^TA, depending on the shape of the matrix, and then uses `eigs`. We found that this method is more robust, from the perspective of our software testing framework, than applying `svds` directly to A.

SSBIN and SNBIN are useful only if the number of matrix-vector products is significantly smaller than the size of the matrix. We assess the results of the algorithms when they are allowed $K = \lfloor pn \rfloor$ iteration, where $p < 1$ and n is the size of the matrix. If the matrix is rectangular, n is the larger of the number of rows and columns. For SSBIN, one iteration corresponds to one matrix-vector product; for SNBIN, two, one with the matrix and one with its transpose. We test four values of p: 0.5%, 1%, 2%, 5%. If a matrix is too small, then K is too small; therefore, we consider matrices having size $n \geq 200$, so that the four values of p correspond to minimum values of K of 1, 2, 4, 10. Of course, in practice, one likely has a much larger matrix.

Since our algorithms are stochastic, their performance on a matrix varies with each application. We run the stochastic algorithms five times for each matrix and accumulate the results.

Lemmas 8 and 9 allow for a variety of distributions for the random vector involved in the matrix-vector product. Chen and Demmel [14] use the random vector z such that $z_i \in \{-1, 1\}$ is iid, each value 1 and -1 having equal probability. Their Lemma 6 shows that this distribution has minimal variance among all distributions such that $\mathrm{E}\, z_i = 0$ and $\mathrm{E}\, z_i^2 = 1$. They connect minimal variance to fast convergence in estimates based on z. We find that the vector u such that $u_i \sim N(0, 1)$ produces

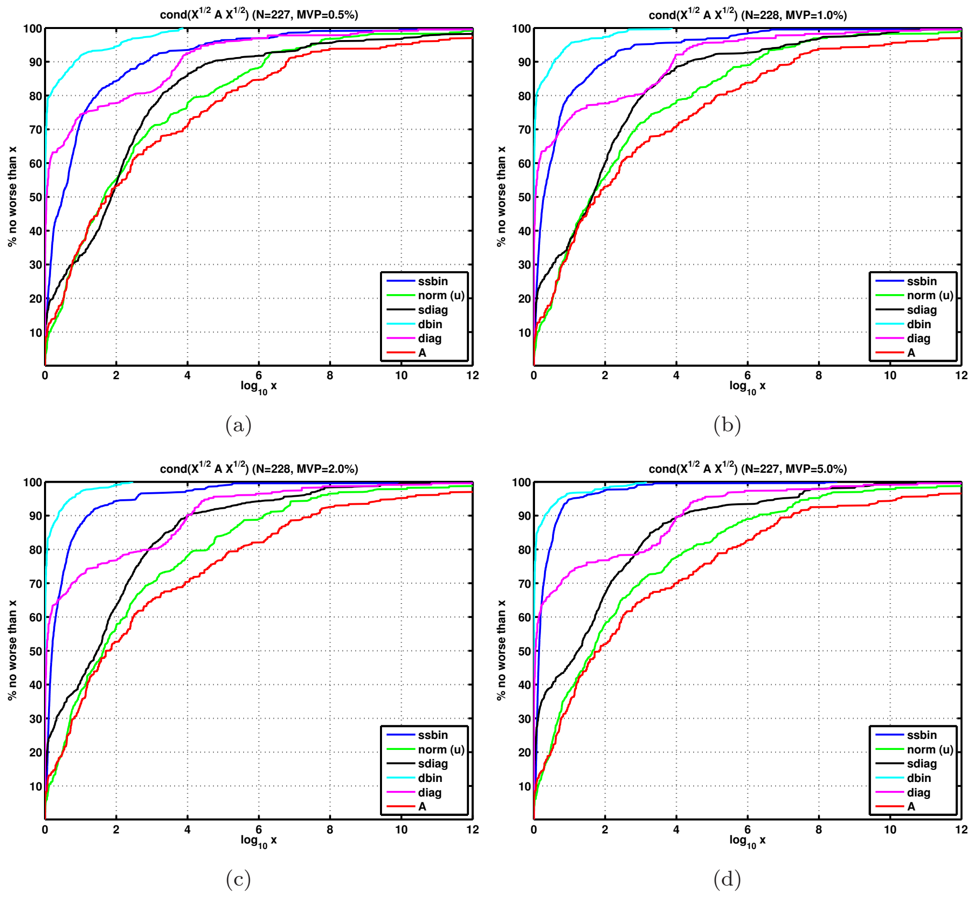


Figure 2.4: The performance of all methods for symmetric matrices. The title of each figure indicates the number of matrix-vector products and the performance metric. MVP $= p\%$ means that $\lfloor \frac{pn}{100} \rfloor$ matrix-vector products are used for a matrix of size n. In these plots, the performance metric is the condition number of the scaled matrix.

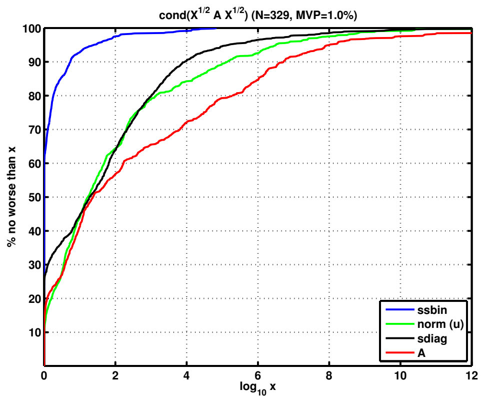
slightly better results for our algorithms, and we always use this distribution. Other algorithms we discuss in this section use z. Two sets of test results compare u and z.

For convenience, we refer to the algorithm that uses the identity matrix as a scaling matrix as algorithm A. This algorithm corresponds to doing nothing.

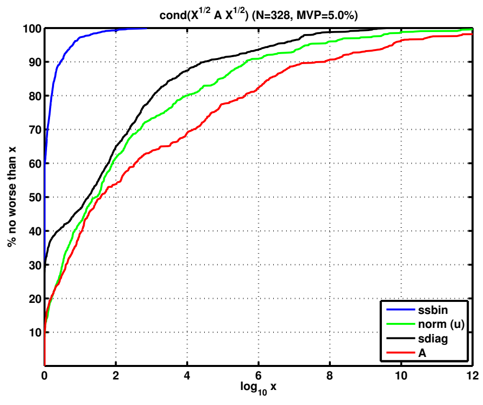
We plot results in three primary formats. Figure 2.7 uses all three, and so we refer to this figure in the following descriptions.

- We display the distribution in sizes of the matrices in our test sets by a plot of cumulative density. Let (n, y) be a point on the blue curve. At least $y\%$ of the test matrices have size at least n. See Figure 2.7(a) for an example.

- Let the performance of an algorithm on a particular problem be characterized by a single

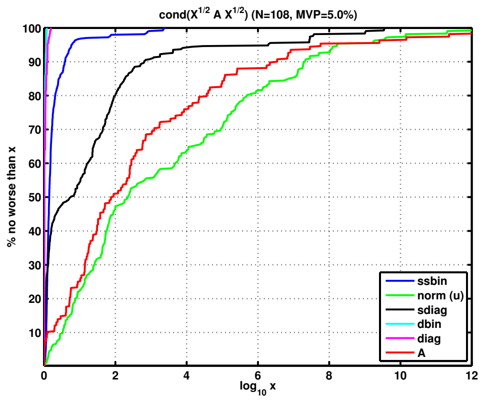


(a)

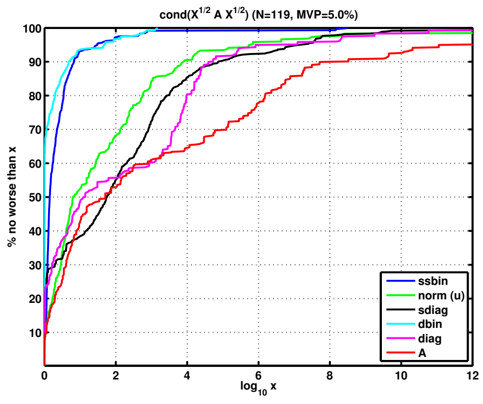


(b)

Figure 2.5: The performance of all stochastic methods for symmetric matrices.



(a)



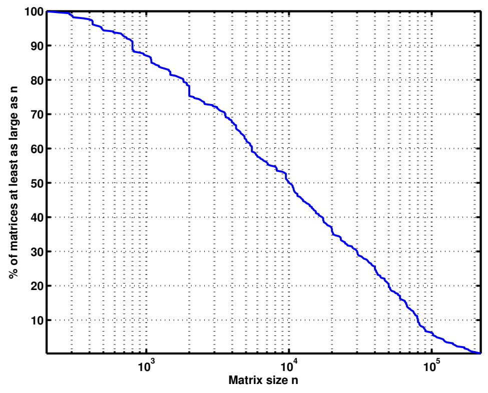
(b)

Figure 2.6: The performance of all methods for symmetric matrices; $p = 5\%$. The matrices are (a) positive definite and (b) indefinite.

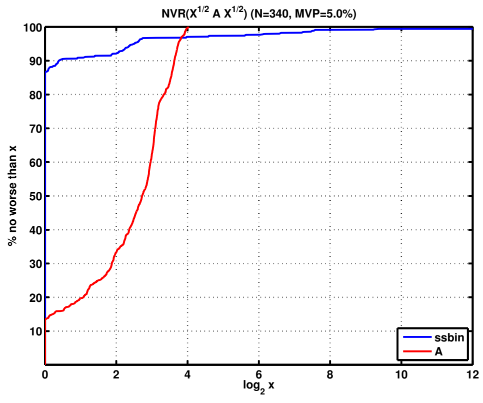
scalar metric χ such that a smaller χ indicates better performance. Suppose K algorithms are run on each problem in a test set of size N. For problem n, let the smallest metric value be $\bar{\chi}_n$. Dolan and Moré [19] introduced performance profiles as shown in Figure 2.7(b) and, as a more complex example, each of the plots in Figure 2.5. Each curve corresponds to an algorithm. Consider the point (x, y) on curve k. The interpretation is that algorithm k yields metric values $\{\chi_n\}_{n=1}^N$ that are within a factor x of $\{\bar{\chi}_n\}_{n=1}^N$ on y percent of problems. The title of each plot gives the test set size N, and the legend indicates the algorithms considered.

A simple rule of thumb when looking at performance profiles is that curves corresponding to better algorithms go to $y = 100\%$ faster.

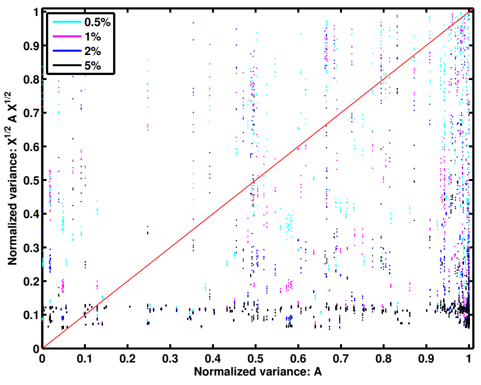
- The scatter plot, an example of which is Figure 2.7(c), compares one algorithm, for each of the



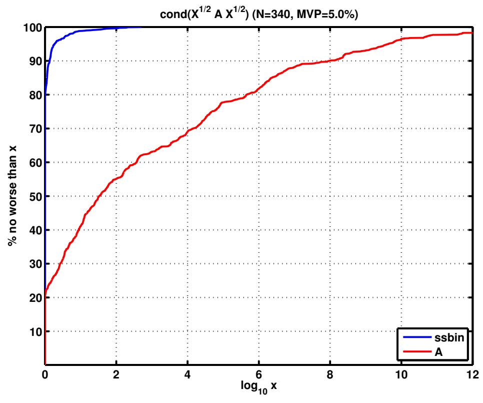
(a)



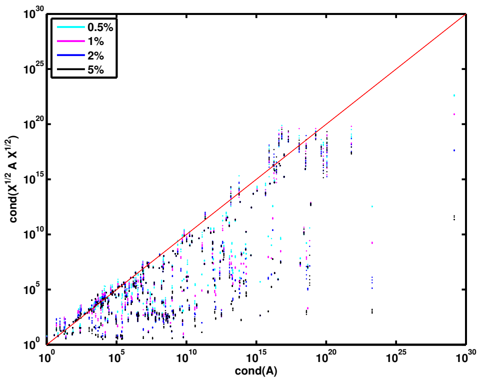
(b)



(c)

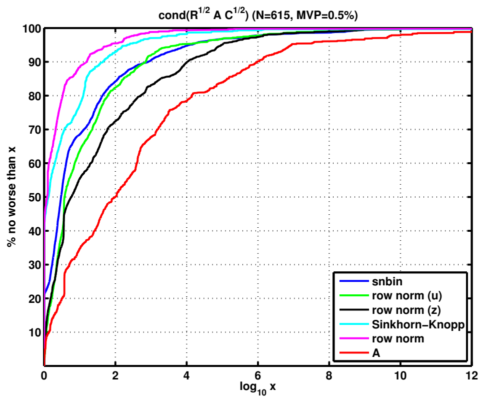


(d)

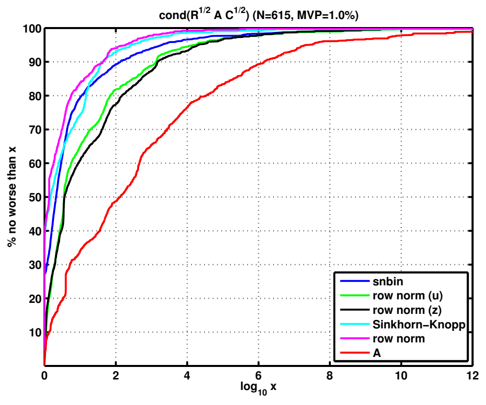


(e)

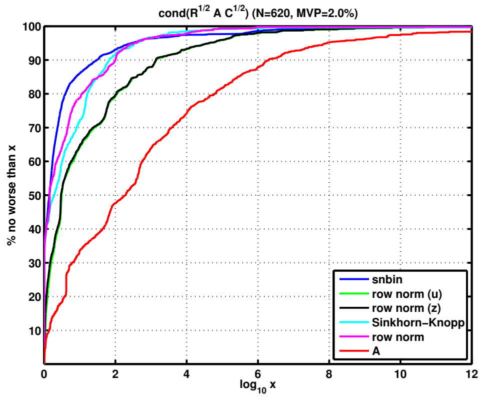
Figure 2.7: The performance of SBIN on symmetric matrices. **(a)** Matrix sizes. **(b, c)** The performance metric is the normalized variance of the row 2-norms (NVR). **(d, e)** The performance metric is the condition number.



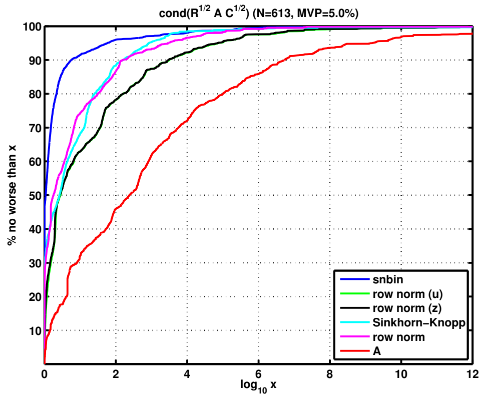
(a)



(b)



(c)



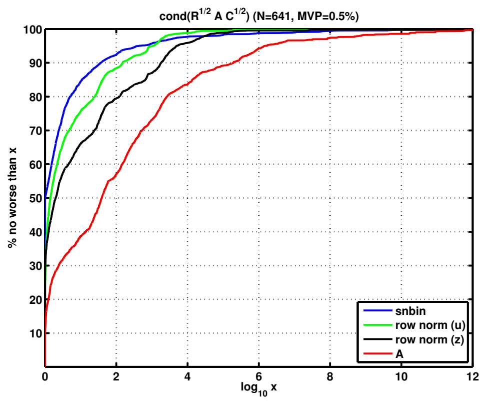
(d)

Figure 2.8: The performance of all methods for nonsymmetric square matrices.

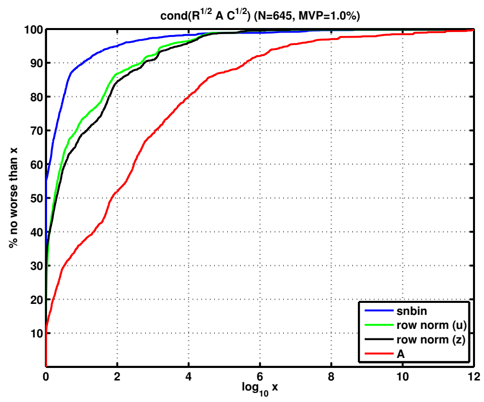
four values of p, against algorithm A. A point (x, y) having the kth color, as indicated in the legend, is interpreted as follows. Algorithm A yields the metric value x, while the algorithm yields the value y. The red line is for convenience: any point below the red line improves upon algorithm A. The points corresponding to $p = 5\%$ are bolder than the others.

When constructing a plot of results, we discard any matrix for which we do not have results from every algorithm included in the plot. MATLAB's `eigs` and `svds` functions can consume a large amount of memory on some large matrices, causing the MATLAB process to die. In other cases, the function `scond` obtains an erroneous result from `eigs` or `svds`, and again we lose a measurement. Finally, we do not run the deterministic algorithms on some of the largest problems because our MATLAB implementation of DBIN is rather slow. It would be be better to implement it as a `mex` function. Consequently, we often plot results for subsets of the algorithms on a larger set of matrices in addition to comparing all the algorithms on a smaller set.

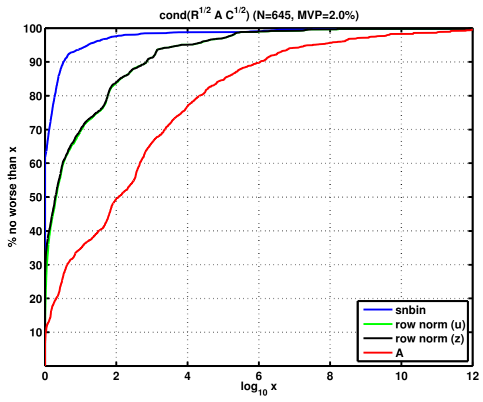
We also discard any matrix that is already binormalized. For both the sets of symmetric and



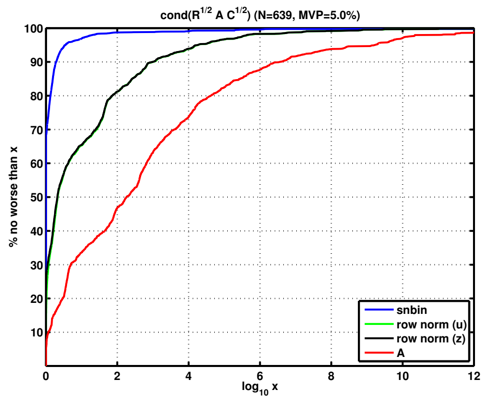
(a)



(b)



(c)



(d)

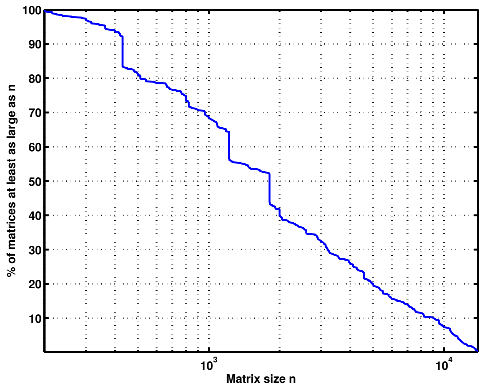
Figure 2.9: The performance of all stochastic methods for nonsymmetric square matrices.

nonsymmetric square matrices, we end up discarding only slightly more than 10 matrices each for this reason. But quite a number of the matrices in the rectangular set are already binormalized.

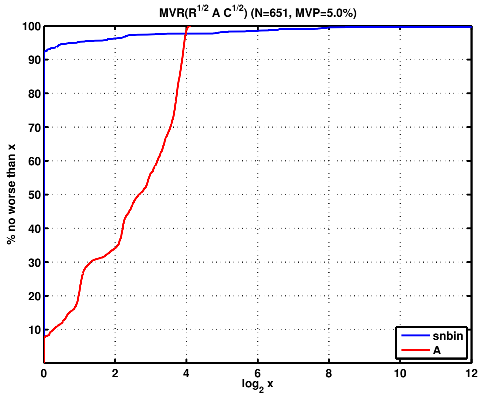
Symmetric matrices

We test other algorithms, both deterministic and stochastic, in addition to ours. In the case of symmetric matrices, we consider the following additional algorithms:

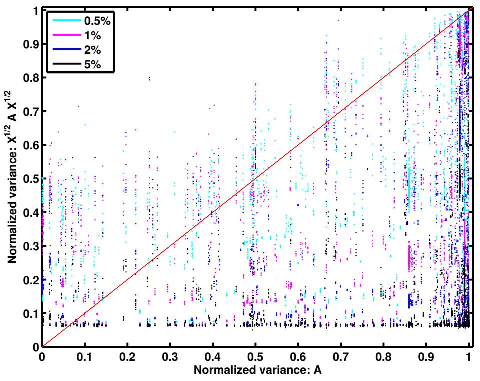
- NORM(U). Compute the sample mean of the row 2-norms. Use u rather than z. NORM(Z) will be considered in the experiments involving nonsymmetric matrices. Let the estimate be r. Scale A as $R^{-1/2}AR^{-1/2}$.



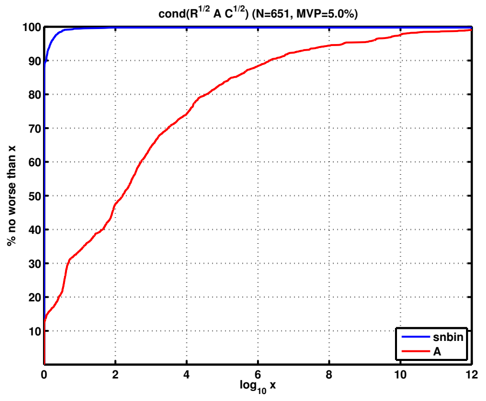
(a)



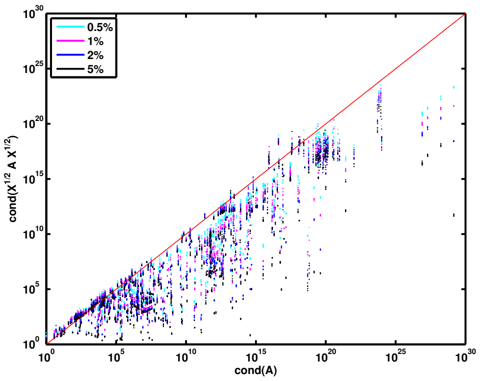
(b)



(c)



(d)



(e)

Figure 2.10: The performance of SNBIN on nonsymmetric square matrices.

- SDIAG. The simplest of Bekas, Kokiopoulou, and Saad's [1] stochastic methods to estimate the diagonal of a matrix. The algorithm uses z and is as follows:

$$d = \frac{1}{N}\sum_{i}^{N} z^i (Az^i).$$

The algorithm computes N matrix-vector products with random vectors z^i and then computes a sample mean. $z^i(Az^i)$ is an element-wise product between z^i and Az^i. The quantities $z^i_j z^i_j = 1$, and $z^i_j z^i_k \in \{-1, 1\}$ with equal probability, and so the diagonal elements of A sum while the off-diagonal elements tend to cancel out. Use the result just as exact Jacobi scaling uses the diagonal. The method is not intended to produce a preconditioner, but we include it in our numerical experiments because the diagonal estimators developed in [1] are the only matrix-free methods of which we know, other than the straightforward NORM methods, that may be considered to produce a preconditioner.

- DBIN. We rename Livne and Golub's [42] method SBIN as DBIN for clarity; D stands for *deterministic*.
- DIAG. Jacobi scaling. If a diagonal element is 0, substitute 1.

Each iteration of each of these algorithms except DIAG performs work equivalent to one matrix-vector product, and so the values of p apply equally to all of them. DIAG of course requires very little work at all.

The algorithms are related to each other. SSBIN is the stochastic matrix-free algorithm corresponding to the deterministic DBIN; similarly, SDIAG corresponds to DIAG. NORM(U) is like SSBIN in the sense that each iteration involves estimating the 2-norm of the rows of a matrix.

Figure 2.4 shows performance profiles for all four values of p using condition number as the metric. As expected, DBIN does best on average. But SSBIN gets close as p increases. Similarly, SDIAG's performance approaches that of DIAG as p increases, as one expects. All the algorithms do better than A on average.

Whether a matrix is positive definite is important. Figure 2.6 shows results for $p = 5\%$, grouping the results into those for positive definite and indefinite matrices. Based on the analysis in Section 2.2.3, we expect Jacobi scaling and binormalization to have similar performance on positive definite matrices, and indeed this expectation is quite clearly borne out. In contrast, on indefinite matrices, the binormalization algorithms perform significantly better than the Jacobi scaling algorithms.

We recall again that Bekas, Kokiopoulou, and Saad [1] had no intention of devising a method to compute a diagonal preconditioner using only matrix-vector products. Moreover, they developed a more complicated algorithm that we did not implement. It uses a sequence of vectors taken from Hadamard matrices and is particularly effective on banded matrices. Hence any criticism of SDIAG in this context is quite unwarranted. We note simply as a sanity check that even on positive definite matrices, SSBIN significantly outperforms SDIAG. On indefinite matrices, a diagonal estimator, however effective, is ultimately limited by the poor performance of Jacobi scaling.

Next, we show comprehensive results for SSBIN. Figure 2.7(a) shows the sizes of the test matrices. To confirm that SSBIN equilibrates matrices, Figures 2.7(b,c) show performance profiles and a scatter plot using normalized variance of the row 2-norms as the metric. As expected, the NVR is reduced significantly on average. Finally, Figures 2.7(d,e) show the same types of plots but once more using condition number as the metric.

The dots in Figure 2.7(c) have a horizontal trend at slightly above 0.1. If the parameter ω in SSBIN were allowed to go to 0 in a controlled way that obeyed certain conditions, then the theory of stochastic approximation suggests the normalized variance should go to 0. As we are interested in finite-iteration—indeed, small-iteration—results, we keep ω fixed after decreasing it for a certain number of iterations. We believe the horizontal scatter of dots corresponds to the lower limit of the normalized variance that can be achieved given the final fixed value of ω. One might decrease ω beyond our final value. Our experience is that if ω becomes small too quickly, more iterations may be required. As robustness is more important than exact equilibration, we have chosen the final value of ω accordingly. Still, a careful parameter study of ω for certain classes of matrices appearing in a particular application may reveal a better sequence $\{\omega_k\}$.

Nonsymmetric square matrices

Our original motivation for this work was equilibrating symmetric matrices, but the existence and uniqueness theory in Section 2.2 supports both symmetric and nonsymmetric matrices, and SSBIN extends immediately to SNBIN. Hence we present results for the latter.

In addition to SNBIN, we test the following other algorithms:

- ROW NORM(U). Estimate the row norm of the matrix using u. Let r be the row norms. The scaled matrix is $R^{1/2}A$.
- ROW NORM(Z). Estimate the row norm of the matrix using z.
- SK. The Sinkhorn-Knopp iteration.
- ROW NORM. Exact row norm.

The two stochastic ROW NORM methods require only a forward matrix-vector product per iteration. Since they are being compared with SNBIN, which requires both a forward and a transpose matrix-vector product, they are permitted two forward products for every iteration of SNBIN.

Figure 2.8 shows results for all the methods, and Figure 2.9 for just the stochastic methods. The random vector u appears to perform slightly better than z in the ROW NORM methods on the matrices in this set when the number of matrix-vector products is small. For $p \geq 1\%$, all the methods do better on average than A.

The most important observation comes from Figure 2.9: as the stochastic ROW NORM methods require the same work as SNBIN, and as SNBIN outperform those methods on average, one might prefer the more complicated SNBIN to the simple ROW NORM methods.

Interestingly, the Sinkhorn-Knopp iteration does not do particularly well on these test problems under the restrictions on the number of matrix-vector products we impose. Indeed, Figures 2.8(c,d) show that the iteration's performance is somewhat worse than SNBIN's. We believe the disparity in performance relates to an observation about the Sinkhorn-Knopp iteration we made early in our investigations: it takes a while for the iterates to settle down. One can regularize the Sinkhorn-Knopp iteration in several ways, and we have found—although we do not present the experiments, as they are just some of many we performed that are outside the scope of this chapter—that certain methods of regularization can improve the behavior of the Sinkhorn-Knopp iteration in the early iterations, although at the cost of decreased convergence speed in later iterations. SNBIN effectively implements one form of regularization simply by averaging the current iterate with a stochastic estimate of the next Sinkhorn-Knopp iterate. In any case, regularizing the Sinkhorn-Knopp iteration when one has access to the elements of the matrix is almost certainly not the right thing to do; rather, one should implement one of the algorithms described in, for example, [36].

Rectangular matrices

Again, our primary concern was symmetric matrices, but SNBIN works on both square and rectangular matrices. Recall, however, that we have not developed a theory of ε-scaling for rectangular matrices.

In our tests, a matrix is transposed, if necessary, so that it has more rows than columns. In addition to the other stochastic algorithms, we also test COLUMN NORM (U); it is implemented just as ROW NORM (U) but for the columns.

Figure 2.11 shows our results. Figure 2.11(b) compares using u and z vectors for the ROW NORM algorithm. In contrast to the results on the test set of square nonsymmetric matrices, the algorithm using z slightly outperforms that using u.

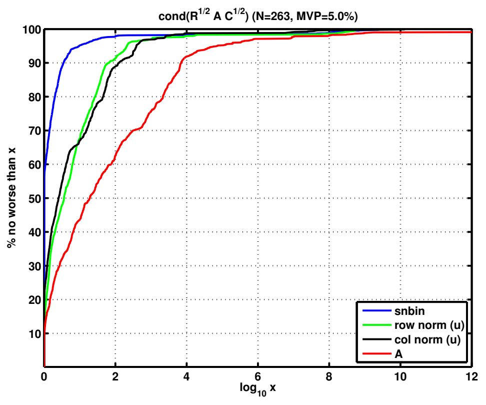
Fixed number of matrix-vector products

In the previous experiments, we have set the number of matrix-vector products proportional to the size of the problem. Figure 2.12 shows results of experiments in which the number of matrix-vector products is fixed for all problems.

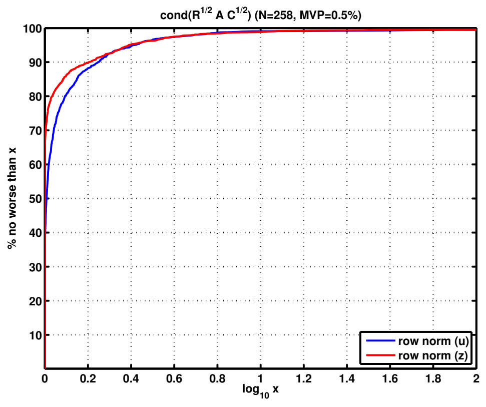
2.5 Summary and conclusions

Scaling a matrix usually reduces its condition number. Perhaps the simplest scaling method for symmetric matrices is Jacobi scaling. But this method is not effective for many indefinite matrices. Other simple scaling methods are based on using the row or column norms.

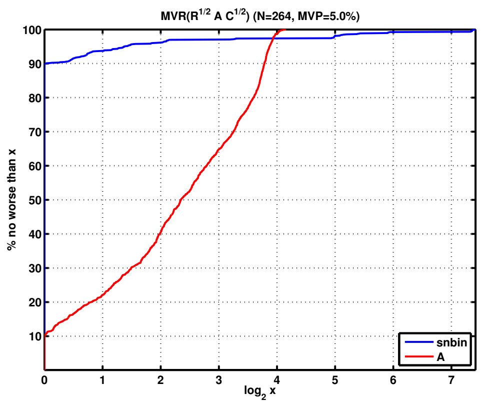
Equilibration in a p-norm can be an effective alternative. Using a combination of theory and numerical experiments, we showed that symmetric binormalization and Jacobi scaling are about equally effective on symmetric positive definite matrices. But symmetric binormalization is quite a



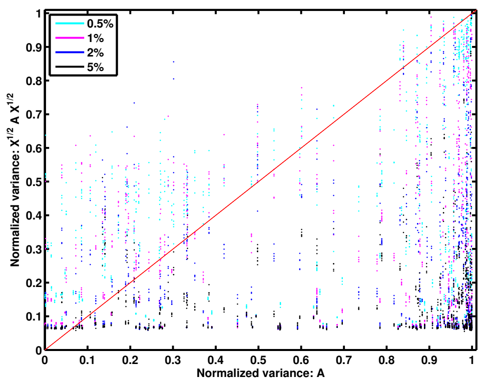
(a)



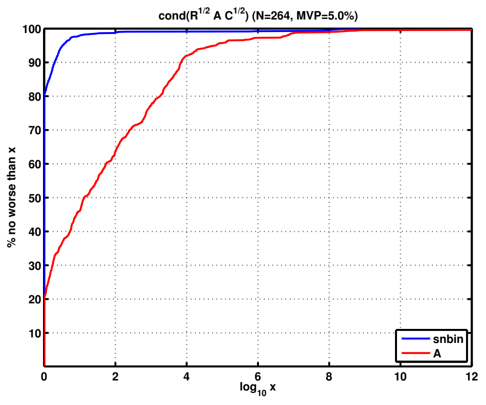
(b)



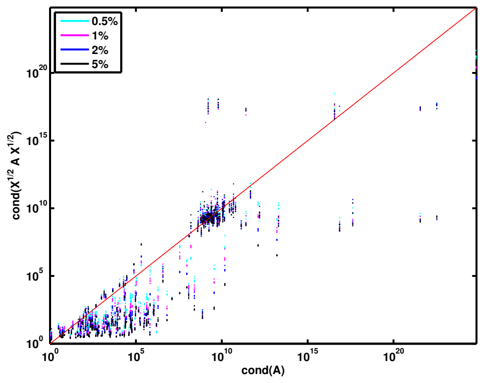
(c)



(d)

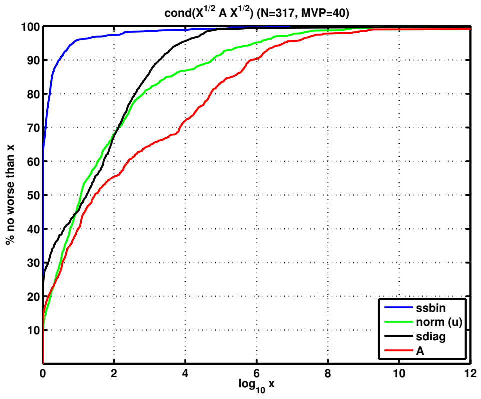


(e)

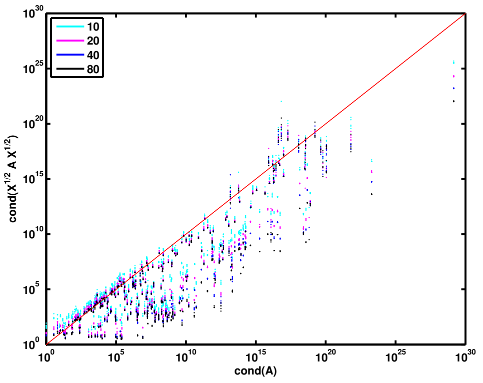


(f)

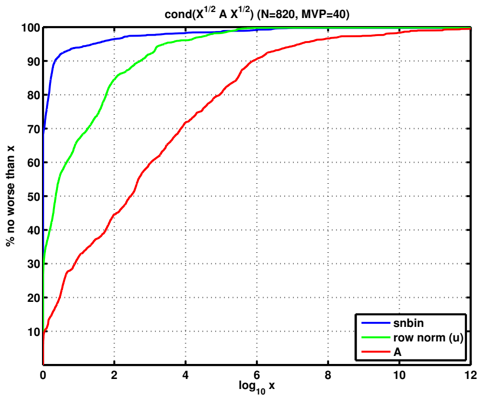
Figure 2.11: The performance of SNBIN on rectangular matrices.



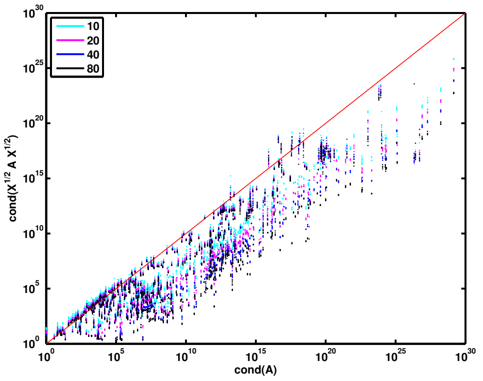
(a)



(b)



(c)



(d)

Figure 2.12: The performance of the stochastic methods when the number of matrix-vector products is fixed for all matrices. **(a, c)** Performance profiles for all stochastic methods on symmetric and square nonsymmetric matrices, respectively, when 40 matrix-vector products are used. **(b, d)** The performance of SSBIN and SNBIN, respectively, for 10, 20, 40, and 80 matrix-vector products.

bit better than Jacobi scaling on indefinite matrices. For nonsymmetric matrices, it is also better, on average, than simply scaling to unit row or column norms.

Approximate scaling is a practical alternative to exact scaling, and we wanted to develop a theory characterizing it. We reviewed the theory governing exact scaling and developed the theory of ε-scaling. It was already known that a (symmetric) square matrix is (symmetrically) scalable if and only if it has total support (part 1 of Theorem 1 and Theorem 2), and it is diagonally equivalent to a unique doubly stochastic matrix (part 2 of Theorem 1). We discovered that a (symmetric) square matrix is (symmetrically) ε-scalable if and only if it has support (Theorems 5 and 3), and the doubly stochastic matrix C obtained in the limit $\varepsilon \to 0$ is unique (Theorem 6).

While the class of scalable matrices excludes some numerically nonsingular matrices, the class of ε-scalable matrices is precisely the class of structurally nonsingular matrices and so includes all numerically nonsingular matrices. This observation suggests that ε-scaling is a useful formalization of approximate scaling.

Previously developed scaling methods require access to the elements of a matrix. We developed the matrix-free stochastic binormalization methods SSBIN and SNBIN to perform approximate symmetric and nonsymmetric scaling. The performance of these algorithms on a large test set suggests that stochastic binormalization scales a matrix using a number of matrix-vector products that is small relative to the size of the matrix and so is a practically useful method when direct access to the elements of a matrix is not available.

Chapter 3

A limited-memory quasi-Newton method

In this chapter, we develop a limited-memory (LM) quasi-Newton (QN) method. It has two main features: it uses a circular buffer to store QN pairs, and it updates an initial matrix B_0 each time it updates the buffer. We develop an update that sets B_0 to a positive diagonal matrix D such that $D^{-1/2}$ approximately binormalizes the Hessian.

First, we provide the foundation for the update. Second, we describe the diagonal update and the rest of the algorithm. Third, we discuss the implementation of the algorithm in the current version of SNOPT [25] and a future version that will have a new quadratic program solver. Finally, we describe the performance of the modified version of SNOPT on several test problem sets.

3.1 Background

3.1.1 Notation

Let n be the size of the problem. The difference between two iterates x_{k+1} and x_k is $s_k \equiv x_{k+1} - x_k$, and the difference in the gradients, denoted g, of the Lagrangian at the two iterates is $y_k \equiv g_{k+1} - g_k$. If the particular iteration update index k is irrelevant, the notation $s = x^+ - x$ may be used. A *pair* is the tuple $\{s_k, y_k\}$. A buffer of pairs is a finite sequence of pairs and is denoted $\{s_j, y_j\}_{j=k_1}^{k_2}$ for $k_1 \le k_2$. Let n_p be the number of pairs the pair buffer can hold. $B = \text{bfgs}(\{s_j, y_j\}_{j=k_1}^{k_2}, B_0)$ denotes the limited-memory BFGS Hessian approximation, where B_0 is the initial matrix, $\{s_j, y_j\}_{j=k_1}^{k_2}$ is the pair buffer, and $k_2 - k_1 + 1 \le n_p$. In some cases, the second argument is a vector, say d_0, and $B_0 = \text{diag}(d_0)$.

The key property of quasi-Newton methods is the quasi-Newton, or *secant*, condition, $B_{k+1} s_k = y_k$. We omit a review of quasi-Newton methods, as it is far better to read those in, for example, [26] and [51].

3.1.2 Linear invariance of Broyden-class updates and diagonal scaling

As a first step, we examine the diagonal matrix B_0 in the context of Broyden-class updates.

Linear invariance

Let $f(x) \equiv \bar{f}(\bar{x})$, where $\bar{x} \equiv Cx$ and C is a nonsingular matrix. Then

$$\begin{aligned} g(x) &\equiv \nabla_x f(x) = C^T \bar{g}(\bar{x}) \\ H(x) &\equiv \nabla_x^2 f(x) = C^T \bar{H}(\bar{x}) C. \end{aligned} \tag{3.1}$$

Corresponding to s and y are the transformed vectors $\bar{s} = Cs$ and $\bar{y} = C^{-T}y$. Let B be a quasi-Newton approximation to H that is updated according to a formula in the Broyden class:

$$\begin{aligned} v_k &= \frac{y_k}{y_k^T s_k} - \frac{B_k s_k}{s_k^T B_k s_k} \\ B_{k+1} &= B_k - \frac{B_k s_k s_k^T B_k}{s_k^T B_k s_k} + \frac{y_k y_k^T}{s_k^T y_k} + \phi_k (s_k^T B_k s_k) v_k v_k^T, \end{aligned} \tag{3.2}$$

where ϕ_k is a scalar.

Theorem 11. *If $B_0 = C^T \bar{B}_0 C$, and $\bar{s}_i = Cs_i$ and $\bar{y}_i = C^{-T} y_i$ hold for all s_i, y_i, $\bar{s}_i$, $\bar{y}_i$, then $B_{k+1} = C^T \bar{B}_{k+1} C$.*

This property of the update is widely known, although we do not have a specific reference for exactly the form we use here. The proof is by induction.

Proof. Suppose $B_k = C^T \bar{B}_k C$. Then we have the following equalities:

$$\begin{aligned} B_k s_k s_k^T B_k &= (C^T \bar{B}_k C)(C^{-1}\bar{s}_k)(C^{-1}\bar{s}_k)^T (C^T \bar{B}_k C) = C^T (\bar{B}_k \bar{s}_k \bar{s}_k^T \bar{B}_k) C \\ s_k^T B_k s_k &= (C^{-1}\bar{s}_k)^T (C^T \bar{B}_k C)(C^{-1} s_k) = \bar{s}_k \bar{B}_k \bar{s}_k \\ y_k y_k^T &= (C^T \bar{y}_k)(C^T \bar{y}_k)^T = C^T (\bar{y}_k \bar{y}_k) C \\ s_k^T y_k &= (C^{-1}\bar{s}_k)^T (C^T \bar{y}_k) = \bar{s}_k^T \bar{y}_k \\ B_k s_k y_k^T &= (C^T \bar{B}_k C)(C^{-1}\bar{s}_k)(C^T \bar{y}_k)^T = C^T (\bar{B}_k \bar{s}_k \bar{y}_k^T) C. \end{aligned}$$

Hence

$$
\begin{aligned}
B_{k+1} &= B_k - \frac{B_k s_k s_k^T B_k}{s_k^T B_k s_k} + \frac{y_k y_k^T}{s_k^T y_k} + \phi(s_k^T B_k s_k) v_k v_k^T \\
&= C^T \left(\bar{B}_k - \frac{\bar{B}_k \bar{s}_k \bar{s}_k^T \bar{B}_k}{\bar{s}_k^T \bar{B}_k \bar{s}_k} + \frac{\bar{y}_k \bar{y}_k^T}{\bar{s}_k^T \bar{y}_k} + \phi(\bar{s}_k^T \bar{B}_k \bar{s}_k) \bar{v}_k \bar{v}_k^T \right) C \\
&= C^T \bar{B}_{k+1} C. \qquad \square
\end{aligned}
$$

B_k is a preconditioner for a steepest-descent method

A quasi-Newton matrix can be thought of as a preconditioner for the problem. Let the QN matrix $B_k = C^T C$. Consider again the function $f(x)$. The quasi-Newton method gives the search direction

$$
p = -B_k^{-1} g(x) = -C^{-1} C^{-T} g(x) = -C^{-1} \bar{g}(\bar{x}).
$$

With step size α, the new iterate is

$$
x - \alpha B_k^{-1} g(x) = C^{-1} \bar{x} - \alpha C^{-1} \bar{g}(\bar{x}) = C^{-1}(\bar{x} - \alpha \bar{g}(\bar{x}));
$$

hence the QN step is the steepest-descent step for the transformed problem in the transformed space. We think of the transformed problem $\bar{f}(\bar{x})$ as the *preconditioned* problem.

That B_k is a preconditioner has a recursive aspect to it. Again, let $B_k = C^T C$. By Theorem 11,

$$
\begin{aligned}
B_{k+1} &= B_k - \frac{B_k s_k s_k^T B_k}{s_k^T B_k s_k} + \frac{y_k y_k^T}{s_k^T y_k} + \phi(s_k^T B_k s_k) v_k v_k^T \\
&= C^T \left(I - \frac{\bar{s}_k \bar{s}_k^T}{\bar{s}_k^T \bar{s}_k} + \frac{\bar{y}_k^T \bar{y}_k^T}{\bar{s}_k^T \bar{y}_k} + \phi(\bar{s}_k^T \bar{s}_k) \bar{v}_k \bar{v}_k^T \right) C,
\end{aligned}
$$

where $\bar{s}_k = C s_k$ and $\bar{y}_k = C^{-T} y_k$; hence the Broyden-class update may be thought of as an update of the identity matrix for a problem preconditioned by $B_k = C^T C$.

Scaling

In the special case that $B_0 = D$ is a positive diagonal matrix, $D^{-1/2}$ is a scaling matrix that scales the original problem. If we view B_0 as a preconditioner, then an effective procedure is to set $B_0 = D$, where scaling the Hessian by $D^{-1/2}$ minimizes its condition number over all positive scaling matrices.

In general the Hessian is a function of the iterate and, moreover, we do not have access to it either by element or by matrix-vector products. This chapter describes a method that approximately binormalizes the Hessian based on the sequence of pairs $\{s_j, y_j\}$. The diagonal B_0 is updated every time the QN matrix as a whole is updated, and so the diagonal scaling matrix changes with the iterate.

3.1.3 SNOPT

SNOPT's current limited-memory quasi-Newton method is as follows [25]. Let r be a major iteration when the pair buffer is empty. B_0 is a positive diagonal matrix that initializes the approximation to the Hessian. For $r \le k \le r + n_p$, the approximation B_k is updated as follows:

$$\begin{aligned} q_j &= B_j s_j \\ \theta_j &= (y_j^T s_j)^{-1} \\ \phi_j &= (q_j^T s_j)^{-1} \\ B_k &= B_0 + \sum_{j=r}^{k-1} (\theta_j y_j y_j^T - \phi_j q_j q_j^T). \end{aligned} \tag{3.3}$$

This form of the update is called the *summation form*. To assure positive definiteness in the presence of numerical errors, the update is rewritten in *product form* as $B_k = G_k^T G_k$, where

$$\begin{aligned} v_j &= \pm(\theta_j \phi_j)^{1/2} y_j - \phi_j q_j \qquad (3.4) \\ G_k &= B_0^{1/2} \prod_{j=r}^{k-1} (I + s_j v_j^T). \end{aligned}$$

The sign in v_j is chosen to minimize cancellation error when forming the vector.

At update index $k = r + n_p + 1$, both the full approximation and B_0 are reset to the diagonal of B_{r+n_p+1} (*i.e.*, the diagonal of the approximation that includes the new pair $\{s_j, y_j\}_{j=r+n_p+1}$), and all pairs $\{s_j, v_j\}$ (including the one for $j = r + n_p + 1$) are flushed from the pair buffer. Additionally, the diagonal matrix is multiplied by a scalar to calibrate the unit step. The implementation of the diagonal matrix B_0 in the context of this scheme is quite efficient: two vectors b_0 and $\bar{b}_0$ store diagonal information. b_0 is static during the accumulation of pairs in the buffer; meanwhile, $\bar{b}_0$ is efficiently updated at each major iteration. When the buffer is flushed, b_0 is set to $\bar{b}_0$.

A feature of SNOPT's current method—indeed, a primary motivation for its form—is that the factorization of the reduced Hessian can be updated directly when SNOPT solves a linearly constrained problem.

3.1.4 Our work

For nonlinearly constrained problems and linearly constrained problems for which evaluating the objective is expensive, SNOPT's method may be improved in two ways. First, rather than discarding all pairs at certain iterations, we could maintain a circular buffer of pairs. Second, rather than updating the initial diagonal B_0 only at certain iterations, it could be updated at every iteration if information is available to support such frequent updates. This chapter develops a method having these two properties.

3.1.5 Related work

Possibly the earliest work on a limited-memory method that does not enforce a sparsity pattern is that of Nocedal [50]. He developed the method L-BFGS, which maintains a circular buffer of a limited number of pairs. The method forms the foundation for many modern variations.

The simplest updating method for B_0 is to set $B_0 = \sigma_k I$, where σ_k is updated at each update of the QN matrix. Liu and Nocedal [40], among others, set $\sigma_k = y_k^T y_k / s_k^T y_k$.

Recent work on quasi-Newton methods includes that of Kolda, O'Leary, and Nazareth [37] and Gill and Leonard [23, 24]. Kolda and her coworkers studied limited-memory methods using Broyden-class updates. They showed that only BFGS among the Broyden-class updates maintains the property of finite termination on unconstrained convex quadratic problems when modified to be a limited-memory method. They also tested several variants of full- and limited-memory methods, studying update skipping and various rules for discarding pairs from the pair buffer.

Gill and Leonard [23] developed a quasi-Newton method that pays particular attention to the subspace spanned by the gradients or, equivalently, the search directions. The method optionally can limit the subspace from which the next search direction is obtained. In the case of a full-memory method, their factorization of the BFGS matrix permits the update of $B_0 = \sigma_k I$; the authors note that this can be quite advantageous in the full-memory case, as a poor static matrix $B_0 = \sigma I$ can slow convergence. They developed a limited-memory variant [24] of the method that uses half the storage of standard limited-memory methods, including the one we use in this chapter.

Several authors have studied methods using diagonal initial matrices B_0. SNOPT's current method is of course one such method. Nazareth [48] and Zhu, Nazareth, and Wolkowicz [66] have developed quasi-Newton-like methods—they call them quasi-Cauchy methods—based on the quasi-Cauchy, rather than quasi-Newton, condition $s_k^T D_{k+1} s_k = s_k^T y_k$, where D_{k+1} is a diagonal matrix. The condition multiplies each side of the quasi-Newton condition by s_k^T. These methods are not themselves relevant to our work; however, in the conclusion to [66], the authors remark that the diagonal matrix D_{k+1} was incorporated into L-BFGS in the thesis work of Zhu. We shall remark further on this in a moment.

The work closest to ours is that of Gilbert and Lemarechal [22] and the follow-up work of Veersé and Auroux [64] and Veersé, Auroux, and Fisher [65]. These authors tested several diagonal updates with the L-BGFS method on a set of unconstrained problems. Their best diagonal update is, with the exception of implementation details and a calibration of the unit step length, the same as the one we develop. However, they do not attempt to understand why the update is effective, nor do they extend the implementation to constrained problems.

Interestingly, the authors of [65] assessed incorporating the quasi-Cauchy method of [66] into L-BFGS, as the latter authors had suggested in their concluding section, and found the results were relatively poor. Prior to discovering the work of Veersé and coworkers, we had implemented this combined method and could not obtain good results. Our view is that the quasi-Cauchy method, restricted as it originally was simply to a diagonal matrix, may be quite effective. But the interpretation of a diagonal B_0 as a matrix that attempts to capture curvature information is incompatible with subsequently updating this matrix to a full one: as the discussion in Section 3.1.2 shows, in the context of a Broyden-class method, a diagonal B_0 really ought to be viewed as a scaling matrix.

Morales [45] conducted a numerical study comparing L-BFGS [50] with SNOPT on a set of unconstrained problems. Of course SNOPT's reduced-Hessian method is computationally unsuited to large unconstrained problems, but Morales correctly used the number of calls to the user function to assess the two methods. He found that L-BFGS generally outperformed SNOPT.

SNOPT's current method cannot be directly modified to accommodate a circular buffer and continual diagonal update. SNOPT stores the pairs $\{s_j, v_j\}$. If B_0 changes, then so does each v_j, for v_j depends on $q_j = B_{j-1}s_j$, and B_{j-1} changes with B_0. Furthermore, the efficient implementation using the vectors b_0 and $\bar{b}_0$ is no longer possible, for in general one pair is removed from the circular buffer at each iteration. If we ignore matters of implementation, we can generalize SNOPT's update to the diagonal matrix by setting it to the diagonal of the Hessian approximation at the previous iteration. Let $B_k = \text{bfgs}(\{s_j, y_j\}_{j=k-n_p}^{k-1}, d_k)$, where $B_0 = \text{diag}(d_k)$ is the current initial diagonal matrix. At the next iteration, d_{k+1} is set to $\text{diag}(B_k)$, the pair $\{s_j, y_j\}_{j=k-n_p}$ is removed from the buffer, the pair $\{s_k, y_k\}$ is added, and $B_{k+1} = \text{bfgs}(\{s_j, y_j\}_{j=k-n_p+1}^{k}, d_{k+1})$.

One might imagine other diagonal updates as well. Our objective in this chapter is to develop a method whose diagonal update has a motivating interpretation, and to implement the method efficiently in the context of a limited-memory method that uses a circular buffer.

3.2 The diagonal update

This section discusses the diagonal update for our limited-memory quasi-Newton Hessian approximation.

Let A be a symmetric positive definite matrix. Let $\tilde{A} = \tilde{D}^{-1/2} A \tilde{D}^{-1/2}$ be binormalized; in particular, let $B\tilde{x} = \tilde{d}$ (recall $B \equiv A \circ A$ and $X \equiv D^{-1}$; we use B to denote both the element-wise square of A and the QN matrix, but the meaning should be clear from the context). Consider the problem

$$\min_x f(x) \tag{3.5}$$

for

$$f(x) \equiv \frac{1}{2} x^T A x = \frac{1}{2} x^T \tilde{D}^{1/2} \tilde{A} \tilde{D}^{1/2} x = \frac{1}{2} \tilde{x}^T \tilde{A} \tilde{x},$$

where $\tilde{x} \equiv \tilde{D}^{1/2} x$. We refer to quantities having a tilde as scaled quantities. The gradient is

$g(x) \equiv \nabla_x f(x) = Ax$ and the scaled gradient is $\tilde{g}(\tilde{x}) \equiv \nabla_{\tilde{x}} f(\tilde{x}) = \tilde{A}\tilde{x}$, and so

$$g(x) = \tilde{D}^{1/2}\tilde{g}(\tilde{x}).$$

Let the approximate Hessian used in the iteration be B. The search direction is $p = -B^{-1}g$, the step is

$$s = \alpha p = -\alpha B^{-1} g = -\alpha B^{-1}\tilde{D}^{1/2}\tilde{g},$$

and the change in gradient is

$$y = As = -\alpha AB^{-1} g = -\alpha AB^{-1}\tilde{D}^{1/2}\tilde{g},$$

where α is the step size.

3.2.1 The algorithm design principles

The motivation for our diagonal update is the stochastic symmetric binormalization algorithm of Chapter 2. In an unconstrained quadratic program, y is related to s by a matrix-vector product. We explore the hypothesis that expressions involving s and y provide information about the scaling of the problem, much as the stochastic binormalization algorithm uses matrix-vector products with the random vector u.

Let us remark, however, that the discussion in this section is not a proof of any sort. Iterations in optimization algorithms are quite complicated, and it would be difficult, if not impossible, to formulate and prove a meaningful theorem about the diagonal update in the context of such complexity. The update we describe turns out to be useful, and our discussion attempts to explain why.

Our first task is to find an analog in the current setting to the random vector u in the stochastic binormalization algorithm. It is likely that a scaled vector is better than an unscaled one. $\tilde{x}$ is not the right one, as the behavior of it depends on the origin. $\tilde{s} \equiv \tilde{D}^{1/2}s$, $\tilde{y} \equiv \tilde{A}\tilde{s}$, and $\tilde{g}$ are all independent of the origin. We shall ultimately use $\tilde{g}$, but we also consider $\tilde{y}$.

Consequently, our first design principle is that $\tilde{g}/\|\tilde{g}\|$ behaves like a random vector sampled independently from the uniform distribution on the sphere.

Next, the Hessian approximation B, if too complicated, can make analysis difficult. In designing our diagonal update, we assume $B = D$, where D is our diagonal; that is, we assume there is no other quasi-Newton update to B.

Finally, we use an unconstrained convex quadratic problem as our model. Constraints impose restrictions on the space from which $\tilde{g}$ is drawn. However, just as in our stochastic binormalization algorithm, our interest is in what happens during only relatively few iterations. If the constraints are not too restrictive, their presence may not be obvious in the sequence of $\tilde{g}$ vectors until after many iterations have occurred.

The motivation for these principles is that many probability density functions may have the property stated in (2.17)—not necessarily just those pdfs considered in Lemmas 8 and 9—or a similarly useful one, and there may be many processes that update B along with the diagonal update that do not significantly affect the behavior of the diagonal update. But these may be analytically difficult or intractable to study. What we need—and what we believe we achieve with our design principles—is an analytical setting simple enough to admit theoretical exposition but rich enough that we can explain, at least partially, why our algorithm works well in practice.

3.2.2 Some useful results

Estimators of the trace

Estimators similar to those in Lemmas 8 and 9 exist for the trace of a matrix. Suppose the matrix A and the random vector u are independent.

Lemma 13. *If the elements of $u \in \mathbb{R}^n$ have zero mean, positive and finite variance, and are iid, then $E\, u^T A u = \eta \; E \; \operatorname{trace} A$ for finite $\eta > 0$.*

Proof. The cross terms $\mathrm{E}\, u_i u_j = 0$ if $i \neq j$, and so

$$\mathrm{E} \sum_i u_i \left(\sum_j A_{ij} u_j \right) = \mathrm{E} \left(\sum_i A_{ii} u_i^2 + \text{cross terms} \right) = \eta \mathrm{E} \sum_i A_{ii}, \quad \text{where} \quad \eta = \mathrm{E}\, u_i^2 > 0.$$

□

Lemma 14. *If the elements of $u \in \mathbb{R}^n$ have positive and finite variance and are iid with pdf f, and f is symmetric around zero, then*

$$E \frac{u^T A u}{u^T u} = \frac{1}{n} \; E \; \operatorname{trace} A.$$

Proof. A cross term has the form (2.18), and so we are left with

$$\mathrm{E} \frac{\sum_i u_i \left(\sum_j A_{ij} u_j \right)}{\sum_i u_i^2} = \eta \mathrm{E} \sum_i A_{ii}, \quad \text{where} \quad \eta = \mathrm{E} \frac{u_i^2}{\sum_j u_j^2} = n^{-1}.$$

□

Construction of random vectors and matrices

In the absence of analytical results, we can explore properties of an algorithm by running it on randomly generated data. To obtain generic results, it is important to sample data from a meaningful distribution.

We use random spd matrices to test some of the ideas in this chapter. How can we assure that our experiments capture generic properties of the algorithm? For example, does sparsity matter? Scaling?

Symmetric positive definite matrices are entirely characterized by an orthonormal matrix and a positive diagonal matrix: $A = Q\Lambda Q^T$. A sparse matrix A becomes dense upon rotation by a generically chosen orthonormal matrix. Hence if an algorithm is invariant to a transformation by an orthonormal matrix, then sparsity does not affect the mathematical behavior of the algorithm, although it may affect storage and computation efficiency. Indeed, such an invariance property means it is enough to sample only from a distribution over positive diagonal matrices. If an algorithm is not invariant to scaling, then it is not invariant to transformation by an orthonormal matrix.

Let us first discuss vectors. Consider the uniform distribution of points on a unit sphere; for the sphere embedded in n dimensions, call this distribution $S[n]$. We generate a unit vector $v \sim S[n]$ as follows [47]:

1. Generate v such that each element is an iid normal variable.
2. Set $v \leftarrow v/\|v\|_2$.

The distribution $S[n]$ obeys the conditions of Lemmas 9 and 14. Observe that if Q is orthonormal and $v \sim S[n]$, then $Qv \sim S[n]$ as well.

We sample three types of matrices:

1. *Positive diagonal matrices.* We specify the distribution in the context of the experiment.
2. *Symmetric positive definite matrices.* We generate A as follows. Let Λ be a positive diagonal matrix obtained by method 1.
   (a) Generate a square matrix A having normally distributed iid elements.
   (b) Compute the QR factorization $A = QR$.
   (c) Set $A \leftarrow Q\Lambda Q^T$.
3. *Scaled spd matrices.*
   (a) Given the positive diagonal matrix Λ, generate the spd matrix A by method 2.
   (b) Given the positive diagonal matrix D, set $\tilde{A} \leftarrow DAD$.

Expectation of a product

Because $\mathrm{Cov}(x, y) \equiv \mathrm{E}\,(x - \mathrm{E}\,x)(y - \mathrm{E}\,y) = \mathrm{E}\,xy - \mathrm{E}\,x\mathrm{E}\,y$,

$$\mathrm{E}\,xy = \mathrm{E}\,x\mathrm{E}\,y + \mathrm{Cov}(x, y). \tag{3.6}$$

Jensen's inequality

Consider the random variable x and a convex function ϕ. Jensen's inequality [15] is

$$\phi(\mathrm{E}\,x) \le \mathrm{E}\,\phi(x).$$

Consequently, $(\mathrm{E}\,x)^{-1} \le \mathrm{E}\,x^{-1}$.

$\mathrm{E}\,\phi(x)$ need not have an upper bound: consider $\phi(x) \equiv x^{-1}$ for x allowed to approach 0. Still, $(\mathrm{E}\,x)^{-1}$ can be a good approximation to $\mathrm{E}\,x^{-1}$ if x is bounded away from 0. For example, suppose x has the uniform distribution between a and b, and $a, b > 0$: $x \sim U[a, b]$. Then

$$\begin{aligned} (\mathrm{E}\,x)^{-1} &= \left(\frac{1}{b-a}\int_a^b x \;\mathrm{d}x\right)^{-1} = \frac{2}{a+b} \\ \mathrm{E}\,x^{-1} &= \frac{1}{b-a}\int_a^b x^{-1}\;\mathrm{d}x = \frac{\ln(b/a)}{b-a}. \end{aligned}$$

If $b/a \approx 1$, then $\ln(b/a) \approx (b-a)/a$ and so $\mathrm{E}\,x^{-1} \approx a^{-1}$; similarly, $2/(a+b) \approx a^{-1}$, and so the two expectations are approximately equal.

The estimate $(nu^TAu)^{-1} \approx (\textbf{trace}\,A)^{-1}$

Lemmas 13 and 14 describe a matrix-free estimate of the trace of a matrix. We examine the hypothesis that if $u \sim S[n]$ and A is an $n \times n$ spd matrix,

The estimate $(nu^TAu)^{-1} \approx (\mathrm{trace}\,A)^{-1}$ is accurate. (3.7)

Our only analytical characterization of the relationship between the two expressions is the following:

$$\begin{aligned} (\mathrm{trace}\,A)^{-1} &= (n\mathrm{E}\,u^TAu)^{-1} && \text{(by Lemma 14)} \\ &\le \mathrm{E}\,(nu^TAu)^{-1} && \text{(by Jensen's inequality).} \end{aligned} \tag{3.8}$$

Rather than further attempt to find a useful analytical characterization, we describe two numerical experiments that give empirical evidence supporting (3.7).

The first experiment demonstrates that the inequality (3.8) is almost certainly strict:

1. Choose the condition number c of the matrix A, the size of the matrix n, and the number of trials n_t.

2. Generate a vector λ such that $\min_j \lambda(j) = 1$ and $\max_j \lambda(j) = c$ as follows.

   (a) Generate a vector u such that each of $n-2$ elements is iid and $u(j) \sim U[0,1]$, one element is 0, and another is 1.
   (b) Set $u \leftarrow c(i)^u$.

3. Because we shall compute the estimate $nv^T Av$ using a vector $v \sim S[n]$, and so $Q^T v \sim S[n]$ for an orthonormal matrix Q, we may assume the matrix A is simply the diagonal matrix Λ. Record $\operatorname{trace} A = \|\lambda\|_1$.

4. For i from 1 to n_t:
   (a) Generate a unit vector $v \sim S[n]$.
   (b) Record the following:
      i. The estimate $nv^T Av = n \sum_j \lambda(j)v(j)^2$.
      ii. The sample mean of this estimate, $t(i)$, using samples 1 to i.
      iii. The estimate $(nv^T Av)^{-1}$.
      iv. The sample mean of this estimate, $r(i)$, using samples 1 to i.
   (c) Plot t and r against the true values $\operatorname{trace} A$ and $(\operatorname{trace} A)^{-1}$, respectively.

Figure 3.1(a) shows results for a condition number of 10^{12} and matrix size n. Observe that the sample mean for the estimate of $\operatorname{trace} A$ converges to $\operatorname{trace} A$, while the sample mean for the estimate of $(\operatorname{trace} A)^{-1}$ converges to a value greater than $(\operatorname{trace} A)^{-1}$.

The second experiment explores the robustness of the estimate $(nu^T Au)^{-1} \approx (\operatorname{trace} A)^{-1}$:

1. Choose a vector c, each of whose elements gives a condition number for which it is desired to obtain samples. Choose a matrix size n. Choose the number of trials n_t.

2. For each $c(i)$:
   (a) Generate a vector λ as in the first experiment.
   (b) Generate a unit vector $v \sim S[n]$.
   (c) As in the first experiment, we may assume the matrix A is simply the diagonal matrix Λ. For each of n_t trials, compute and record the following:
      i. $\operatorname{trace} A = \|\lambda\|_1$.
      ii. $nv^T Av = n \sum_j \lambda(j)v(j)^2$.
      iii. The relative error $r = \frac{(\operatorname{trace} A)^{-1} - (nv^T Av)^{-1}}{(\operatorname{trace} A)^{-1}}$.
   (d) Plot the middle p percentile of relative errors.

Figure 3.1(b) shows results for condition numbers between 10^2 and 10^{16}, number of trials $n_t = 10^4$, and matrix size $n = 10^3$. Consistent with the inequality (3.8), the stochastic estimate tends to be larger than $(\operatorname{trace} A)^{-1}$. But the estimate is certainly meaningful: half the sampled estimates lie well within $\pm 20\%$ of $(\operatorname{trace} A)^{-1}$ (green line), and 90% lie within $\pm 40\%$.

We conclude that the hypothesis (3.7) holds and so we may use the relationship between the two expressions in our subsequent analysis.

3.2.3 The update

Stochastic framework

We have already seen that we have the two basic quantities

$$\begin{aligned} s &= \alpha p = -\alpha D^{-1} g = -\alpha D^{-1} \tilde{D}^{1/2} \tilde{g} \\ y &= As = -\alpha A D^{-1} g = -\alpha A D^{-1} \tilde{D}^{1/2} \tilde{g}. \end{aligned}$$

Let $u \equiv \tilde{g}/\|\tilde{g}\|$, and let us replace each instance of $\alpha\tilde{g}$ with u:

$$\begin{aligned} s &= -D^{-1} \tilde{D}^{1/2} u \\ y &= -A D^{-1} \tilde{D}^{1/2} u. \end{aligned}$$

We can remove the scalar $\alpha\|\tilde{g}\|$ as later it will cancel out of a quotient.

By Lemmas 9 and 14, these latter s and y vectors yield (recall $X = D^{-1}$)

$$\mathrm{E}_u y^2 = n^{-1} B D^{-2} \tilde{D} e = n^{-1} B \tilde{D} x^2 \tag{3.9}$$

$$\mathrm{E}_u (Ds)^2 = \mathrm{E}_u d^2 s^2 = n^{-1} \tilde{d} \tag{3.10}$$

$$\mathrm{E}_u s^T D s = n^{-1} \operatorname{trace} \tilde{D}^{1/2} X \tilde{D}^{1/2} = n^{-1} x^T \tilde{d} \tag{3.11}$$

$$\mathrm{E}_u y^T s = n^{-1} \operatorname{trace} \tilde{D}^{1/2} X A X \tilde{D}^{1/2} = n^{-1} \operatorname{trace} A \tilde{D} X^2 \tag{3.12}$$

$$\mathrm{E}_u y^T y = n^{-1} \operatorname{trace} \tilde{D}^{1/2} X A^2 X \tilde{D}^{1/2} = n^{-1} \operatorname{trace} A^2 \tilde{D} X^2 \tag{3.13}$$

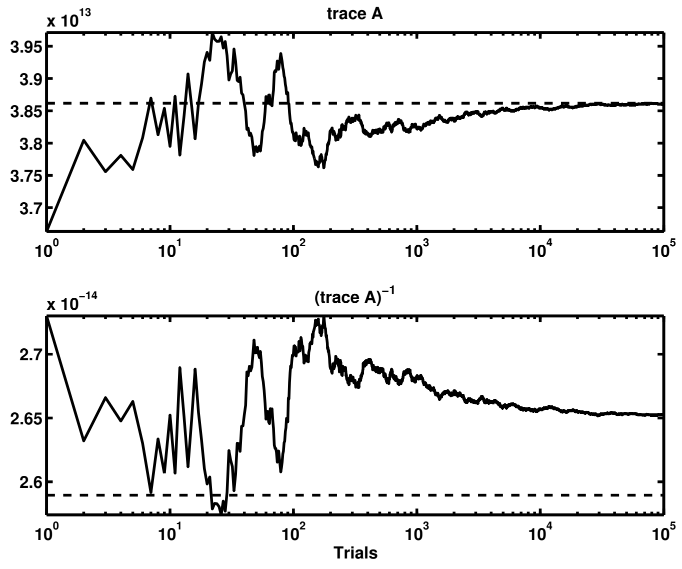
$$\mathrm{E}_u s^T s = n^{-1} \operatorname{trace} \tilde{D}^{1/2} X^2 \tilde{D}^{1/2} = n^{-1} \tilde{d}^T x^2, \tag{3.14}$$

where the subscripted u on the expectation operator indicates the replacement by u.

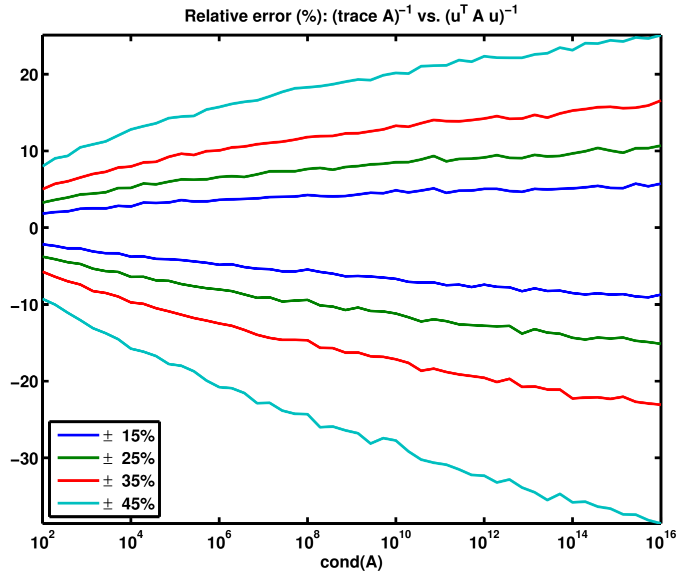
In this section, we derive a deterministic iteration based on the rhs of each of (3.9)–(3.14).

First, observe that $\mathrm{E}_u d^2 s^2 = n^{-1} \tilde{d}$ immediately provides the scaling matrix. Even better, in the case of s, we need not make the simplifying assumption $B = D$. Then $\mathrm{E}_u (Bs)^2 = n^{-1} \tilde{d}$. But y has quite valuable data and so should not be ignored.

Second, suppose $v \equiv \tilde{y}/\|\tilde{y}\|$, rather than $u \equiv \tilde{g}/\|\tilde{g}\|$, is taken to be sampled from a uniform



(a)



(b)

Figure 3.1: Results for two numerical experiments studying the relationship between $(\operatorname{trace} A)^{-1}$ and the estimate $(nu^T Au)^{-1}$. **(a)** Experiment 1. The x axis is the number of trials used to compute the sample mean. The y axes correspond to the values $\operatorname{trace} A$, $(\operatorname{trace} A)^{-1}$, and their estimates. The dashed lines are the true values. **(b)** Experiment 2. The x axis is the condition number of A; the y axis is the relative error of the estimate. Each line color corresponds to the indicated middle percentile of samples.

distribution on the sphere; and let us replace each instance of $\alpha\tilde{y}$ by v. Then

$$y = -\tilde{D}^{1/2} v$$
$$s = -A^{-1}\tilde{D}^{1/2} v = -\tilde{D}^{-1/2}\tilde{A}^{-1} v,$$

where we have assumed A has full rank; and, analogous to the quantities obtained from the E_u operator,

$$\mathrm{E}_v y^2 = n^{-1}\tilde{d}$$
$$\mathrm{E}_v s_i^2 = n^{-1}\tilde{x}_i \sum_j (\tilde{A}^{-1})_{ij}^2.$$

The expression for s^2 is not useful. A simple stochastic iteration based on the expression for y^2 is

$$d^+ = (1-\omega)d + \omega y^2,$$

although we shall use a more complicated one in a numerical test. Again, however, we seek an iteration based on both s and y.

Consider the iteration

$$d^+ = d - \frac{d^2 s^2}{s^T D s} + \frac{y^2}{y^T s}. \tag{3.15}$$

An equivalent expression for the diagonal update is

$$d^+ = \operatorname{diag}\left(D - \frac{D s s^T D}{s^T D s} + \frac{y y^T}{y^T s}\right). \tag{3.16}$$

The diagonal update is obtained simply by extracting the diagonal of the BFGS update to a diagonal matrix. As a BFGS update preserves positive definiteness and the main diagonal of an spd matrix is positive, $d^+ > 0$ if $d > 0$.

Consider any two random quantities y and z inside the E_u operators among (3.9)–(3.14). From the discussion in Section 3.2.2, $\mathrm{E}\, yz \neq \mathrm{E}\, y \mathrm{E}\, z$—the two are separated by $\mathrm{Cov}(y,z)$ according to (3.6)—and $\mathrm{E}\, y^{-1} \neq (\mathrm{E}\, y)^{-1}$, Jensen's inequality providing a one-sided bound. Furthermore, we suggested in that same section that $\mathrm{E}\, z^{-1} \approx (\mathrm{E}\, z)^{-1}$ if z is bounded away from 0. In particular, we discussed the accuracy of the approximation when $z \equiv u^T A u$ for $u \sim S[n]$; see the discussion of the hypothesis (3.7). Here we neglect $\mathrm{Cov}(y,z)$; and as our denominators will turn out to be of the form $z \equiv u^T A u$, we make the approximation

$$\mathrm{E}\, \frac{y}{z} \approx \frac{\mathrm{E}\, y}{\mathrm{E}\, z}. \tag{3.17}$$

This approximation justifies our removing $\alpha\|\tilde{g}\|$ when we defined the E_u operator.

According to the approximation (3.17) and (3.9)–(3.14),

$$\begin{aligned}\frac{d^2 s^2}{s^T D s} &\approx \frac{\tilde{d}}{x^T \tilde{d}}\\ \frac{y^2}{y^T s} &\approx \frac{B\tilde{D}x^2}{\operatorname{trace} A\tilde{D}X^2}.\end{aligned}$$

Therefore, the deterministic iteration corresponding to (3.15) is

$$d^+ = d - \omega\left(\frac{\tilde{d}}{x^T\tilde{d}} - \frac{B\tilde{D}x^2}{\operatorname{trace} A\tilde{D}X^2}\right) \equiv G(d), \tag{3.18}$$

where the scalar ω is introduced for generality. Notice that scaled quantities do not appear in (3.15) but do appear in (3.18). The decomposition $A = \tilde{D}^{1/2}\tilde{A}\tilde{D}^{1/2}$ of course is not known to us and is for analysis only: it reveals the stochastic binormalizing process underlying (3.15).

Analysis of the deterministic iteration

The analysis in this section is very similar to that surrounding Theorem 9.

Theorem 12. *$d = \xi\tilde{d}$, where $\xi \equiv n/\operatorname{trace}\tilde{A}$, is the unique fixed point of the iteration* (3.18).

Proof. Substituting $d = \xi\tilde{d}$ into (3.18) yields

$$\begin{aligned}d^+ &= \xi\tilde{d} - \frac{\omega\xi\tilde{d}}{n} + \frac{\omega B\tilde{x}}{\operatorname{trace} A\tilde{X}}\\ &= \xi\tilde{d} - \frac{\omega n\tilde{d}}{n \operatorname{trace}\tilde{A}} + \frac{\omega B\tilde{x}}{\operatorname{trace}\tilde{A}}\\ &= \xi\tilde{d},\end{aligned}$$

where we have used the equations $B\tilde{x} = \tilde{d}$ and $\operatorname{trace} A\bar{X} = \operatorname{trace}\bar{A}$. Hence $\xi\tilde{d}$ is a fixed point.

d is a fixed point if

$$\frac{\tilde{d}}{x^T\tilde{d}} = \frac{B\tilde{D}x^2}{\operatorname{trace} A\tilde{D}X^2}.$$

As A is, by assumption, positive definite, so is B by the Schur Product Theorem; and so $B\tilde{D}$ is nonsingular. Therefore,

$$\frac{x^T\tilde{d}}{\operatorname{trace} A\tilde{D}X^2}x^2 = (B\tilde{D})^{-1}\tilde{d}.$$

The solution x^2 has the form $\alpha(B\tilde{D})^{-1}\tilde{d}$ for a scalar α. The lhs increases monotonically with α, and so the solution is unique. □

Lemma 15. *The iteration* (3.18) *is invariant to symmetric permutations.*

Proof. Let P be a permutation matrix. Then

$$\begin{aligned} x^T\tilde{d} &= (Px)^T(P\tilde{d}) \\ \operatorname{trace} A\tilde{D}X^2 &= \operatorname{trace}(PAP^T)(P\tilde{D}P^T)(PXP^T)^2. \end{aligned}$$

Hence if d and d^+ satisfy (3.18), then

$$Pd^+ = Pd - \omega\left(\frac{P\tilde{d}}{(Px)^T(P\tilde{d})} - \frac{(PBP^T)(P\tilde{D}P^T)(Px)^2}{\operatorname{trace}(PAP^T)(P\tilde{D}P^T)(PXP^T)^2}\right) = G(Pd),$$

and so Pd and Pd^+ satisfy (3.18) for the matrix PAP^T. □

Theorem 13. $d^* \equiv \xi\tilde{d}$ *is a point of attraction of the iteration* (3.18) *if* $0 < \omega < n$*.*

Proof. First we assume A is irreducible.

Let

$$\begin{aligned} \alpha(d) &\equiv (\tilde{d}^T x)^{-1} \\ \beta(d) &\equiv (\operatorname{trace} A\tilde{D}X^2)^{-1}. \end{aligned}$$

Substituting d^* yields the relations

$$\begin{aligned} \alpha(d^*) &= n^{-1}\xi \\ \beta(d^*) &= \xi^2(\operatorname{trace}\tilde{A})^{-1} = n^{-1}\xi^3, \end{aligned}$$

and using $\partial_d x = -X^2$ and $\partial_x \operatorname{trace} CX = \operatorname{diag}(C)$ for a matrix C yields the derivatives

$$\begin{aligned} \alpha_d(d) &= \frac{\tilde{d}^T X^2}{(\tilde{d}^T x)^2} \\ \beta_d(d) &= \frac{2\operatorname{diag}(A\tilde{D}X^3)}{(\operatorname{trace} A\tilde{D}X^2)^2} \\ \alpha_d(d^*) &= n^{-2}\tilde{x} \\ \beta_d(d^*) &= \frac{2\xi\operatorname{diag}(A\tilde{X}^2)}{(\operatorname{trace}\tilde{A})^2} = 2n^{-2}\xi^3\operatorname{diag}(A\tilde{X}^2). \end{aligned}$$

Then

$$\begin{aligned}
G(d) &= d - \omega\left(\alpha(d)\tilde{d} - \beta(d)B\tilde{D}x^2\right)\\
G_d(d) &= I - \omega\left(\tilde{d}\alpha_d(d) - B\tilde{D}x^2\beta_d(d) + 2\beta(d)B\tilde{D}X^3\right) \equiv J(d)\\
J(d^*) &= I - \omega\left(n^{-2}\tilde{d}\tilde{x}^T - 2n^{-2}\xi B\tilde{x}\ \mathrm{diag}(A\tilde{X}^2)^T + 2n^{-1}B\tilde{X}^2\right).
\end{aligned}$$

Under a similarity transform,

$$\tilde{X}J(d^*)\tilde{D} = I - \omega\left(n^{-2}ee^T - 2n^{-2}\xi\tilde{X}B\tilde{x}\ \mathrm{diag}(A\tilde{X})^T + 2n^{-1}\tilde{X}B\tilde{X}\right).$$

As $B\tilde{x} = \tilde{d}$, $\tilde{X}B\tilde{X} = \tilde{X}\tilde{d} = e$; and $\mathrm{diag}(A\tilde{X}) = \mathrm{diag}(\tilde{D}^{-1/2}A\tilde{D}^{-1/2}) = \mathrm{diag}(\tilde{A})$. We obtain

$$\begin{aligned}
\tilde{X}J(d^*)\tilde{D} &= I - \omega\left(n^{-2}ee^T - 2n^{-2}\xi e\ \mathrm{diag}(\tilde{A})^T + 2n^{-1}\tilde{X}B\tilde{X}\right)\\
&= I - \omega n^{-1}\left(n^{-1}e\left[e - 2\xi\ \mathrm{diag}(\tilde{A})\right]^T + 2\tilde{X}B\tilde{X}\right).
\end{aligned}$$

Let us determine the interval containing the eigenvalues of $J(d^*)$.

1. The first term in parentheses has rank one and the single nonzero eigenvalue

$$n^{-1}e^T\left[e - 2\xi\ \mathrm{diag}(\tilde{A})\right] = 1 - 2n^{-1}\xi\ \mathrm{trace}\,\tilde{A} = 1 - 2 = -1.$$

2. As A is positive definite and $x_i > 0$, so is B and $\tilde{X}B\tilde{X}$; and as $B\tilde{x} = \tilde{d}$, $\tilde{X}B\tilde{X}$ is doubly stochastic. Therefore, $\tilde{X}B\tilde{X}$ has eigenvalues in the interval $(0, 1]$, and as we are temporarily assuming that A is irreducible, the eigenvalue 1 has multiplicity one.

3. As e is an eigenvector of each of the two terms in parentheses, we conclude from Lemma 11 that

$$n^{-1}e\left[e - 2\xi\ \mathrm{diag}(\tilde{A})\right]^T + 2\tilde{X}B\tilde{X}$$

has eigenvalues in the interval $(0, 2)$.

4. Finally, if $0 < \omega < n$, $\tilde{X}J(d^*)\tilde{D}$, and so $J(d^*)$ itself, has eigenvalues in the interval

$$1 - \omega n^{-1}(0, 2) \subseteq (-1, 1).$$

Hence $\rho(G_d(d^*)) < 1$.

Now suppose A is reducible. By Lemma 15, we can assume A is block diagonal and each block is irreducible. Our analysis to this point applies to each block separately. As the Jacobian associated with each block has spectral radius less than 1, so does the Jacobian as a whole.

Hence by Theorem 10.1.3 of [55], the iteration $d^+ = G(d)$ converges locally. $\square$

The third term

The denominator of the third term of (3.15) need not be $y^T s$—for example, Theorems 12 and 13 hold with only minor changes to their statements and proofs if the denominator is either $s^T s$ or $y^T y$—but numerical experiments show that $y^T s$ is best. This may be explained as follows. Let the eigenvectors of A in the model QP (3.5) be v_i; and the eigenvalues, λ_i. The diagonal of A may be written

$$\operatorname{diag}(A) = \sum_i \lambda_i \frac{v_i^2}{v_i^T v_i}.$$

Suppose s is an eigenvector of A and has associated eigenvalue λ. Then $y = As = \lambda s$ and so

$$\frac{y^2}{y^T s} = \frac{\lambda^2 s^2}{\lambda s^T s} = \lambda \frac{s^2}{s^T s},$$

which is exactly the contribution to the diagonal of A that the eigenvector s makes.

A numerical experiment

The analysis in this section makes a number of simplifying assumptions, and it does not do more than attempt to explain the effectiveness of the diagonal update (3.15). Now we describe a numerical experiment that supports the hypothesis that the diagonal update equilibrates the Hessian.

We implemented the L-BFGS method [50] in a solver for unconstrained problems. The line search routine is a Matlab translation of the MINPACK routine `cvsrch` [46] by Dianne O'Leary [53]. We multiply the QN matrix by the scalar α described in Section 3.3.1. We tested the following updates. The colors refer to the corresponding line colors used in the plots in Figure 3.2.

- (black) $B_0 = \sigma I$, where $\sigma = y^T y / s^T y$ [40]. The results of other methods should be compared against those of this one.

- (blue) Just s. The diagonal update is

$$d^+ = \frac{n}{n-1}\left(1 - \frac{ds^2}{s^T D s}\right) d. \tag{3.19}$$

  When $d \propto \tilde{d}$, each element of the parenthesized vector is $(n-1)/n$; the factor $n/(n-1)$ in the update compensates for this.

- (red) Just Bs. An update analogous to (3.19) can produce an indefinite diagonal. Instead, we use an update analogous to (2.21):

$$\begin{aligned} q &= Bs \\ d^+ &= (1-\omega)\frac{d}{\|d\|_1} + \omega\frac{q^2}{\|q^2\|_1}. \end{aligned}$$

We use $\omega = 1/10$.

- (green) Just y, except for a scalar formed by an inner product with s. We use the first and third terms of (3.15). To compensate for the loss of the term involving s, we multiply d by $(n-1)/n$:

$$d^+ = \frac{n-1}{n}d + \frac{y^2}{s^T y}.$$

- (cyan) The algorithm LMCBD, which uses the update (3.15). We discuss this and the next algorithm in Section 3.3.
- (magenta) The algorithm LMCBD-γ, which uses the update (3.15).

The test problem is the QP $f(x) = \frac{1}{2}x^T Ax$. A is constructed as follows. First, $\tilde{A}$ is created by generating a random orthonormal matrix Q and a random diagonal matrix Λ having iid diagonal elements, each the square of a random normal variable, and then setting $\tilde{A} = Q\Lambda Q^T$. Second, a random diagonal scaling matrix $\tilde{D}^{1/2}$ having condition number 20 is generated according to step 2 of the first experiment in Section 3.2.2. Then $A = \tilde{D}^{1/2}\tilde{A}\tilde{D}^{1/2}$. The results we show in Figure 3.2 are generic over other distributions and condition numbers for $\tilde{D}^{1/2}$ and Λ, although we chose an instance from several trials that produced the clearest plots.

The size of the problem is $n = 100$, and we use a circular buffer that can store $n_p = 10$ pairs.

The four plots in Figure 3.2 show the following:

- $\|g\|_2$. The 2-norm of the gradient.
- f. The function value.
- $\text{nvar}(X(A \circ A)x)$. The normalized variance of the 2-norms of the rows of the Hessian A when scaled according to B_0. The constant black line is associated with the constant diagonal σI. The blue curve shows that the update based on Ds does not reduce the variance of the row 2-norms as effectively as the other updates. The other updates approximately equilibrate A.
- $\text{cond}(R^{-T}AR^{-1})$. The QN matrix can be thought of as a preconditioner, and so this plot shows the reduction in condition number of A when preconditioned by two separate matrices: B_0 (curving dashed line) and the full QN matrix B (solid). The three horizontal dashed lines are, from top to bottom, (a) the condition number of A, (b) the condition number of A with

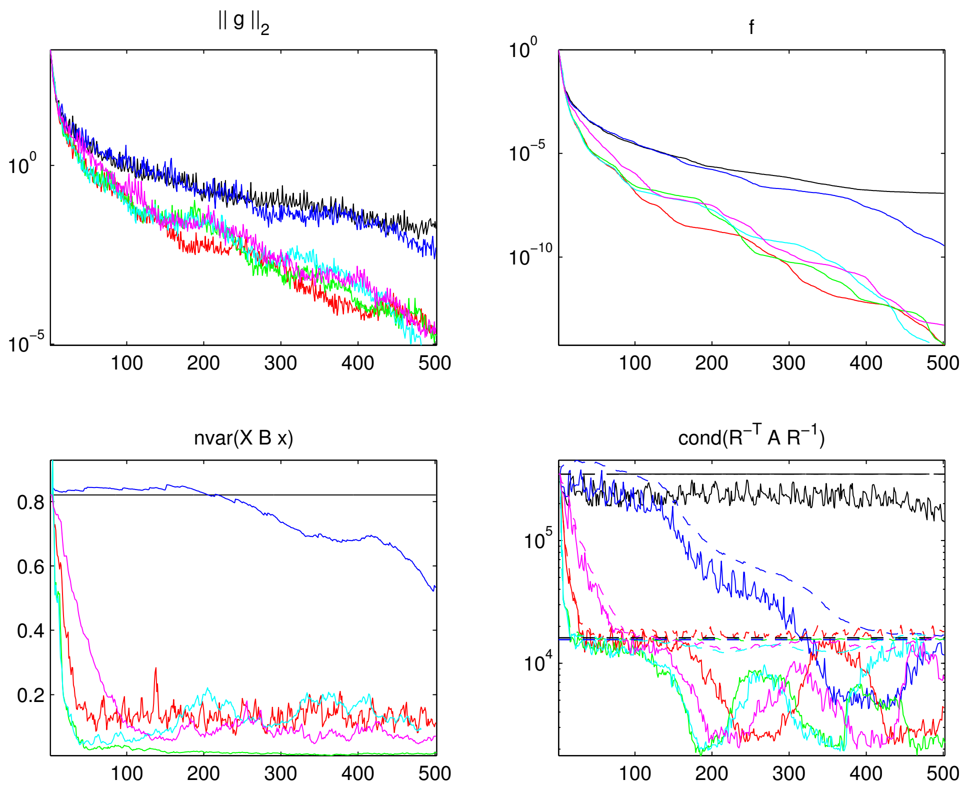


Figure 3.2: Numerical experiment for various diagonal updates.

Jacobi scaling, and (c) the condition number of A with exact binormalization. The discussion regarding the relationship between binormalization and Jacobi scaling for spd matrices in Section 2.2.3 explains why (b) and (c) are quite close to each other. The solid curves—corresponding to preconditioning by the full QN matrices—fall below (c), but the dashed curves—corresponding to preconditioning by just B_0—level off at (b) and (c), as we expect.

This experimental procedure does not predict the performance of diagonal updates on constrained problems: when implemented in SNOPT, only the update (3.15), implemented in the algorithm LMCBD and LMCBD-γ, proved to be robust over the range of test problems we consider in Section 3.5.

3.3 The limited-memory quasi-Newton methods: LMCBD and LMCBD-γ

In this section we describe two closely related algorithms that use the diagonal update (3.15). We call these algorithms LMCBD and LMCBD-γ: LM is for limited memory; CB, circular buffer; and D, diagonal update.

First we describe a scalar that calibrates the unit step length.

3.3.1 A scalar

After certain steps, the QN matrix H should be multiplied by a scalar to calibrate the unit step length. SNOPT performs this calibration after the first QN update following the reset of the QN matrix to a diagonal. We use the same calibration formula as SNOPT; but as we update the diagonal each time the full QN matrix is updated, we find it best to calibrate at each update.

When updating B_k to B_{k+1}, the basic calibration formula is

$$\bar{\gamma}_k = \frac{y_k^T s_k}{s_k^T B_k s_k}; \tag{3.20}$$

but we also safeguard the value:

$$\gamma_k = \min(100, \max(10^{-4}, \bar{\gamma}_k)). \tag{3.21}$$

We maintain a scalar β_k that is updated by $\beta_{k+1} = \gamma_k \beta_k$. After updating the vector d_k to d_{k+1} according to the diagonal update (3.15), we set $B_0 = \beta_{k+1} d_{k+1}$.

The calibration formula (3.20) has at least one natural interpretation. At iterate x_k, we have a quadratic approximation to the Lagrangian:

$$F(p) = \frac{1}{2} p^T B_k p + g_k^T p + F_0.$$

The solution is the step $p_k = -B_k^{-1} g_k$. Next, the line search determines the step size α_k. The new iterate is $x_{k+1} = x_k + \alpha_k p_k$, and the new gradient is g_{k+1}. As $s_k = x_{k+1} - x_k = \alpha_k p_k$,

$$B_k s_k = -\alpha_k g_k. \tag{3.22}$$

Given this information, we would like to determine a scalar γ_k such that if $\bar{B}_k \equiv \gamma_k B_k$, then the unit step is likely acceptable. One obvious choice is $\gamma_k = \alpha_k^{-1}$, for then $\bar{B}_k s_k = \alpha_k^{-1} B_k s_k = -g_k$ and the corresponding unit step has already been accepted. But this choice ignores the additional information g_{k+1}.

Let

$$f(\alpha) \equiv p_k^T F(\alpha p_k) = \frac{a}{2}\alpha^2 + b\alpha + c$$

be a scalar quadratic function in the step size α. Then

$$f'(\alpha) = a\alpha + b.$$

As

$$\begin{aligned} f'(0) &= b = p_k^T g_k \\ f'(\alpha_k) &= a\alpha_k + b = p_k^T g_{k+1}, \end{aligned}$$

we obtain

$$\begin{aligned} a &= \frac{p_k^T(g_{k+1} - g_k)}{\alpha_k} = \frac{p_k^T y_k}{\alpha_k} \\ b &= p_k^T g_k. \end{aligned}$$

Minimizing $f(\alpha)$ gives a step size that can be expected to be better than α_k. $f'(\bar{\alpha}) = 0$ if

$$\bar{\alpha} = -\frac{b}{a} = -\frac{\alpha_k p_k^T g_k}{p_k^T y_k} = \frac{\alpha_k s_k^T g_k}{s_k^T y_k} = \frac{s_k^T B_k s^T k}{s_k^T y_k},$$

where the last equality follows from (3.22). Finally, we set $\gamma_k = \bar{\alpha}^{-1}$.

An obvious question is whether one could do better based on information internal to the line search: fit a higher-order polynomial than a quadratic, for example. In fact, the unit step should be taken frequently as the algorithm converges, and so the quadratic fit uses all the information that is typically available without expending any extra user function calls.

A less quantitative interpretation of the calibration formula is revealing. $s_k^T y_k$ is the change in the gradient of the Lagrangian along s_k. $s_k^T B_k s_k$ is the expected change in the gradient based on the quadratic approximation $F(p)$. Hence $\gamma_k > 1$ when the actual change is larger than expected, and $\gamma_k < 1$ when it is smaller.

The classical L-BFGS method of Nocedal [50] uses the scalar $\sigma_k = y_k^T H_0 y_k / y_k^T s_k$, where typically $H_0 = I$. The interpretation of this scalar is that it broadly accounts for the magnitude of the curvature of the problem. In Section 3.5, we compare γ_k with this classical one on unconstrained problems and find γ_k gives slightly superior performance.

The update (3.15) is the same as one of those considered in [22, 64, 65], but our scalar is quite different. Their scalar is

$$\nu_k = \frac{y_k^T D_k^{-1} y_k}{s_k^T y_k},$$

which is the same as σ_k for $H_0 = D_k^{-1}$. Using ν_k on constrained problems is disastrous; and as we demonstrate in Section 3.5, ν_k is not as good as γ_k on unconstrained problems.

Our view is that γ_k is superior to both ν_k and σ_k because it is motivated by the most natural interpretation: not as a means to account for the magnitude of curvature, but rather as a means to calibrate the unit step length given sufficient information to form a quadratic model of the merit function projected along the current step direction.

3.3.2 LMCBD

Now we can assemble our primary algorithm, LMCBD. If $s_k^T y_k$ is sufficiently positive, the update is as follows:

1. Update the diagonal d_k to d_{k+1} according to (3.15).
2. If $\mathrm{cond}(\mathrm{diag}(d_{k+1}))$ exceeds a tolerance, reinitialize the quasi-Newton approximation, set $d_{k+1} = e$, reset β_{k+1}, and go to step 5.
3. Compute γ_k according to (3.20) and (3.21).
4. Set $\beta_{k+1} = \gamma_k \beta_k$.
5. Set $B_0 = \beta_{k+1} \mathrm{diag}(d_{k+1})$.
6. Update the pair buffer with the new pair $\{s_k, y_k\}$.

In step 2, other choices are possible; for example, we could set $B_0 = \sigma I$, where $\sigma = y_k^T y_k / s_k^T y_k$, and retain the pair buffer. However, we have found that on the rare occasions that the condition number of $\mathrm{diag}(d_{k+1})$ exceeds our tolerance, it is likely that the current pairs are not helpful for finding a good descent direction. β_0 could be set to 1; but as it is available to us in SNOPT, we use as a better initial estimate the scalar `U0pre**2`, a description of which is beyond the scope of this chapter.

3.3.3 LMCBD-γ

When the iterates are converging, d may remain approximately the same. But how does d behave during the initial iterations? In fact, making a general statement is not possible: the behavior of d in the early iterations depends very much on the problem. We have observed, particularly when the degrees of freedom in a problem is large and so a limited-memory method will typically require a large number of iterations, that d may change quickly during early iterations and so be unhelpful. In contrast, on problems having few degrees of freedom, the total number of major iterations can be quite small, and so a quickly changing d can be useful.

In any case, the problem is not severe; and we certainly do not recommend using the modified form of LMCBD we describe in this section for anything other than unconstrained problems. Our primary motivation for this section and the subsequent numerical experiments involving both LMCBD and

LMCBD-γ is that Veersé and Auroux [65] considered essentially both algorithms—using ν_k rather than γ_k—among the many other diagonal updates they tried, and could not make a firm recommendation in favor of one over the other. Our objective, then, is to understand how the two algorithms differ.

Suppose γ_k is much larger than 1. Then evidently the step size α_k was quite small, and the current model of the second-order information greatly overpredicts the extent to which the iterates should differ. In that case, it may be useful to prevent d from changing by much. Next, suppose γ_k is much smaller than 1. Then the step size α_k was quite large, and so the iterates greatly differ. In this case, it may be useful to allow d to change greatly.

In LMCBD, $\beta_k = \prod_{j=0}^{k} \gamma_k$. In LMCBD-$\gamma$, we set $\beta_k \equiv 1$ and accumulate γ_k directly in the diagonal d_k:

$$d^+ = \left(e - \frac{ds^2}{s^T D s}\right)\gamma d + \frac{y^2}{y^T s}.$$

The inclusion of γ in the diagonal update moderates the influence of the term $y^2/y^T s$. If γ is large, the term has a relatively smaller effect; if γ is small, larger.

3.4 Implementation

The software implementations of the diagonal update (3.15) and the scalar β are straightforward, as each involves only element-wise or inner products among the vectors d_k, s_k, and y_k.

The critical part of the implementation is the limited-memory quasi-Newton method itself. It must handle a continually updated diagonal initial matrix B_0 and a circular buffer.

3.4.1 Implementation in SNOPT 7 and 8

Three computational requirements guide our implementation. First, SNOPT solves the quadratic program (QP) subproblem using a null-space method. This QP solver accesses the quasi-Newton matrix through matrix-vector products. Second, the implementation should allow for a fast update. Third, storage should be minimal: the optimal amount of storage is $(2n_p + 1)n$: storage for n_p pairs and one vector for the diagonal.

The product-form update maintains numerical positive definiteness, and it may be that this concern should have priority. However, we have implemented a summation-form update and have not experienced numerical problems. For a circular buffer, the summation form requires less storage.

The product-form stores the pairs $\{s_j, v_j\}$ (see Section 3.1.3). v_j is a linear combination of y_j and q_j. When a pair is removed from the circular buffer, a new B_{j-1} corresponds to the old s_j, and so q_j and v_j are incorrect. Therefore, y_j must be stored whether or not v_j is. But storage for either v_j or q_j must remain, as the product-form matrix-vector product depends on one of these. Hence the product form requires storage for $3n_p + 1$ n-vectors.

Byrd, Nocedal, and Schnabel's [13] compact representation of a limited-memory BFGS method is useful. Let

$$\begin{aligned} D_{ii} &= s_i^T y_i \\ L_{ij} &= s_i^T y_j \quad \text{for } i > j. \end{aligned}$$

Their representation is as follows:

$$\begin{aligned} JJ^T &= S^T B_0 S + L D^{-1} L^T \equiv C \\ F &= \begin{pmatrix} D^{1/2} & 0 \\ -LD^{-1/2} & J \end{pmatrix} \\ M &= \begin{pmatrix} -D & L^T \\ L & S^T B_0 S \end{pmatrix} = F \begin{pmatrix} -I & 0 \\ 0 & I \end{pmatrix} F^T \\ B &= B_0 - \begin{pmatrix} Y & B_0 S \end{pmatrix} M^{-1} \begin{pmatrix} Y^T \\ S^T B_0 \end{pmatrix}. \end{aligned} \tag{3.23}$$

Theorem 2.4 of [13] shows that if B_0 is p.d. and $s_i^T y_i > 0$, then C is p.d., and so the Cholesky factorization $C = JJ^T$ exists.

Let k be the number of pairs in the buffer; here we reserve n_p for the number of pairs the buffer can hold. At a particular major iteration, the diagonal is updated, and so $S^T B_0 S$ must be computed anew in $O(k^2 n)$ operations. Then the Cholesky factorization $C = JJ^T$ is computed in $O(k^3)$ operations.

A matrix-vector product $u \leftarrow Bu$ requires $O(kn)$ operations:

1. Set $v \leftarrow \begin{pmatrix} Y^T \\ S^T B_0 \end{pmatrix} u$.
2. Solve $Mw = v$.
3. Set $u \leftarrow B_0 u - \begin{pmatrix} Y & B_0 S \end{pmatrix} v$.

3.4.2 SNOPT in the future: Block-LU QP solver

A future version of SNOPT will use a new QP solver that uses the block-LU method. The block-LU method is particularly suited to large problems in which the number of active constraints is substantially fewer than the number of variables; in this situation, the reduced Hessian in the null-space approach becomes excessively large. Our limited-memory method is well suited to problems on which the block-LU method will excel, for in such problems, second-order information assumes a larger role relative to constraints in determining the search direction.

Consider the quadratic program

$$\begin{aligned}&\underset{x}{\text{minimize}}\ g^T x + \frac{1}{2} x^T H x\\ &\text{subject to } Ax \geq 0.\end{aligned}$$

At a particular iteration, let A_0 be the matrix corresponding to the current active set. The KKT matrix is

$$K_0 = \begin{pmatrix} H & A_0^T \\ A_0 & 0 \end{pmatrix} = L_0 U_0,$$

where $K_0 = L_0 U_0$ is a factorization of K. The block-LU method handles updates to the active set by appending blocks V and D to K_0 and computing a block factorization of the resulting KKT system:

$$\begin{pmatrix} K_0 & V \\ V^T & D \end{pmatrix} = \begin{pmatrix} L_0 & \\ Z^T & I \end{pmatrix} \begin{pmatrix} U_0 & Y \\ & C \end{pmatrix}. \tag{3.24}$$

See Hanh Huyhn's thesis [29] for more about the method.

Suppose $B_k = \text{bfgs}(\{s_j, y_j\}_{j=1}^k, B_0)$. First we consider the product-form BFGS matrix. Let v_j and q_j be as they are defined in Section 3.1.3, and recall $s_j^T q_j = \phi_j^{-1}$. The solutions y_1 and y_2 are the same in the following two systems [29]:

$$\begin{pmatrix} B_k & A^T \\ A & \end{pmatrix} \begin{pmatrix} y_1 \\ y_2 \end{pmatrix} = \begin{pmatrix} d_1 \\ d_2 \end{pmatrix}$$

$$\begin{pmatrix} B_0 & A^T & W \\ A & & \\ W^T & & D \end{pmatrix} \begin{pmatrix} y_1 \\ y_2 \\ r \end{pmatrix} = \begin{pmatrix} d_1 \\ d_2 \\ 0 \end{pmatrix},$$

where $W = \begin{pmatrix} q_1 & v_1 & \cdots & q_k & v_k \end{pmatrix}$, and D is block diagonal and has jth block $\begin{pmatrix} \phi_j^{-1} & 1 \\ 1 & 0 \end{pmatrix}$. If $K_0 = \begin{pmatrix} B_0 & W^T \\ W & \end{pmatrix}$, then the second system has the form (3.24), and so the block-LU method can efficiently incorporate a limited-memory BFGS matrix in product form. If instead the BFGS matrix is expressed in summation form—$B_k = B_0 + UU^T - ZZ^T$—then we can set [25] $W = \begin{pmatrix} U & Z \end{pmatrix}$ and $D = \begin{pmatrix} I & \\ & -I \end{pmatrix}$. U and Z can be expressed in at least two ways. The summation form (3.3) shows that

$$u_j = (s_j^T y_j)^{-1/2} y_j \quad \text{and} \quad z_j = (s_j^T q_j)^{-1/2} q_j.$$

Equivalently, in terms of the compact representation's matrix F in (3.23),

$$\begin{pmatrix} U & Z \end{pmatrix} F^T = \begin{pmatrix} Y & B_0 S \end{pmatrix}. \tag{3.25}$$

Let us consider storage and computation requirements in each case:

1. *Store* $\{s_j, v_j\}$. The updated diagonal and circular buffer require storing the y_j vectors as well. At each major iteration, each v_j vector is recomputed for a total of $O(k^2 n)$ operations. If the block-LU QP solver uses the product form, the $\{s_j, v_j\}$ pairs can be used without modification. Hence the minimum storage is $(3n_p + 1)n$ vectors.

2. *Store* $\{q_j, y_j\}$. Again, a third set of vectors must be stored, in this case s_j. At each major iteration, each q_j vector is recomputed for a total of $O(k^2 n)$ operations. If the block-LU QP solver uses the summation form, the $\{q_j, y_j\}$ pairs can be used without modification. Hence the minimum storage is again $(3n_p + 1)n$ vectors.

3. *Store* $\{s_j, y_j\}$. The SQP method uses the compact representation. At each major iteration, $S^T B_0 S$ must be recomputed in $O(k^2 n)$ operations. The block-LU QP solver can use either the product or the summation forms; in either case, new pairs $\{u_j, z_j\}$ must be computed in $O(k^2 n)$ operations using (3.25). Hence the minimum storage is $(4n_p + 1)n$ vectors.

In all three cases, the diagonal B_0 can be used without modification in the QP solver. This analysis shows that if memory is shared between the outer SQP and inner QP solvers, either of cases 1 and 2 is optimal.

However, this conclusion may be premature. A block-LU QP solver accesses the Hessian approximation directly rather than through matrix-vector products and so can advantageously use external factorization libraries [29]. It is quite likely that the QP solver must store its own pairs $\{u_j, z_j\}$ and diagonal B_0, as these make up some of the elements of the block-LU matrix. Hence the QP solver requires $(2n_p + 1)n$ vectors of storage independent of the outer SQP solver. Meanwhile, the SQP solver requires the storage indicated in each of the three cases: either $(3n_p + 1)n$ (cases 1 and 2) or $(2n_p+1)n$ (case 3). In all three cases, $O(k^2 n)$ operations must be performed at each major iteration. Hence the compact representation may have the same advantage for an SQP method based on a block-LU QP solver as for one based on a null-space QP solver.

3.4.3 Software

We implemented our method in SNOPT. The user has the choice of setting the parameter `Hessian` either to `Limited`, which is the old limited-memory method, or to `LMCBD`, which is our new method. The default limited-memory method is `Limited`. Except for increasing the number of possible values for the parameter `Hessian`, our method does not change SNOPT's user interface.

We recommend using the new `Hessian` option on problems for which evaluating the user function and gradients is time consuming relative to SNOPT's computations. In particular, for problems having only linear constraints and for which the objective is evaluated quickly, the original `Hessian = Limited` option is probably better.

For our numerical experiments on unconstrained problems, we also implemented our method and several others in the software `lbfgs.f` [50].

3.5 Numerical experiments

Our objective in our numerical experiments is to investigate the effectiveness of our algorithms as well as the robustness of the SNOPT modification.

Framework

We use several test sets:

- COPS 3.0 [20]. The COPS 3.0 test set is a set of high-quality constrained problems implemented in AMPL. We discard the quadratic programs, leaving 20 problems. Each problem but one has three sizes, and one has four.
- CUTEr/Gill and Leonard [24]. A set of 51 unconstrained problems. We include an additional problem, `dixmaanj`, because it is a member of a set of problems Gill and Leonard include.
- CUTEr/Kaustuv [31]. 41 of the 42 constrained problems in Kaustuv's thesis. We discard `degen2` as we did not find it in our CUTEr archive.
- CUTEr/Byrd, Lu, Nocedal, and Zhu [12]. 33 of the 34 bound-constrained problems from their test set. One problem, `hs25`, was removed because for an unresolved reason, the CUTEr framework considered the problem solved after 0 major iterations. The number of function calls indicated in the tables of results in [12] suggest those authors had the same issue.
- GPOPS [58]. 13 optimal control problems distributed with the GPOPS system.
- The 34 general problems from SNOPT's `examples` directory.
- CUTEr/`lbfgs.f`. 152 unconstrained problems.

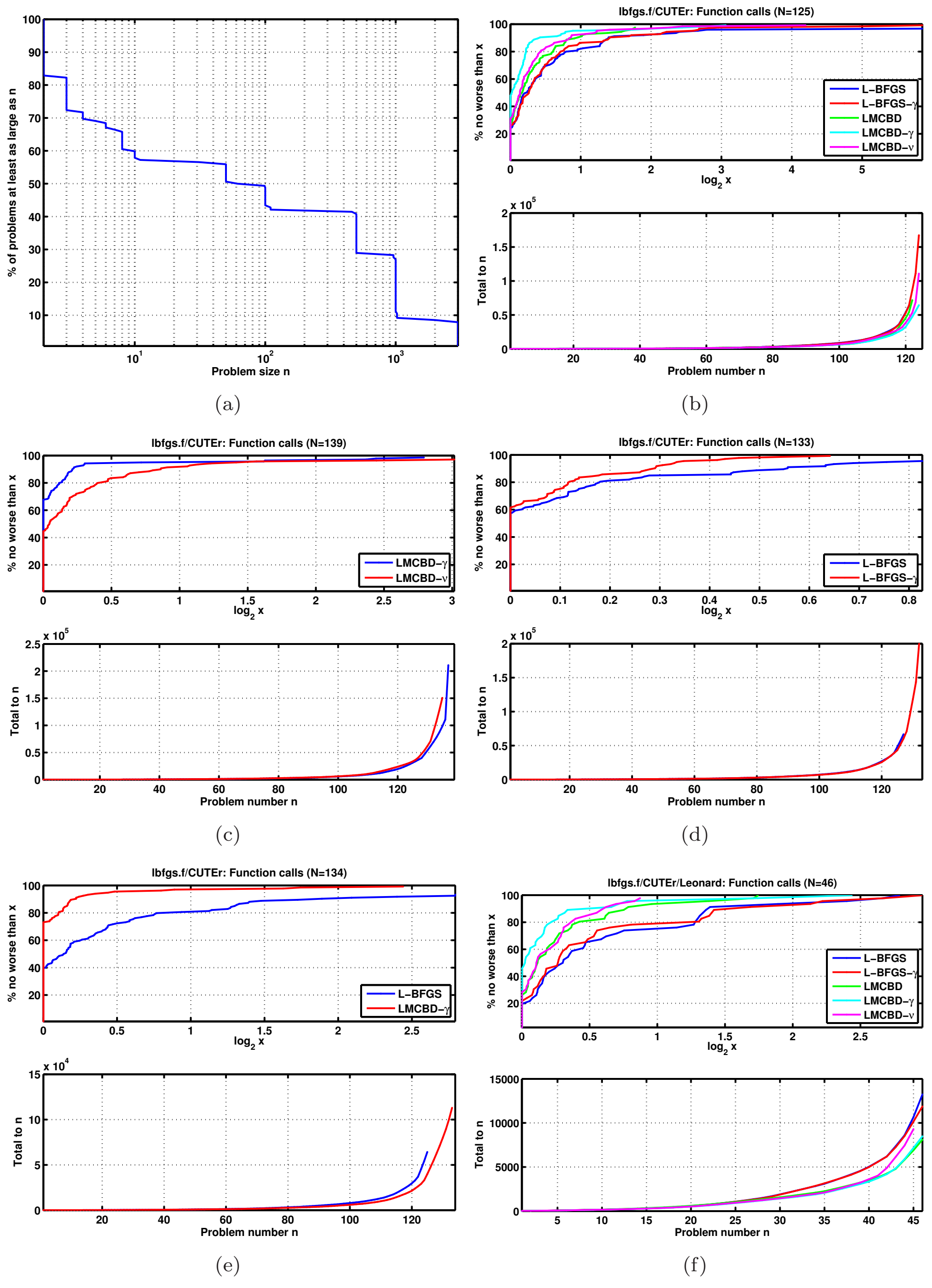


Figure 3.3: Results for the original and modified versions of `lbfgs.f`.

SNOPT is run on all but the final test set; the original and modified forms of `lbfgs.f` are run on the latter.

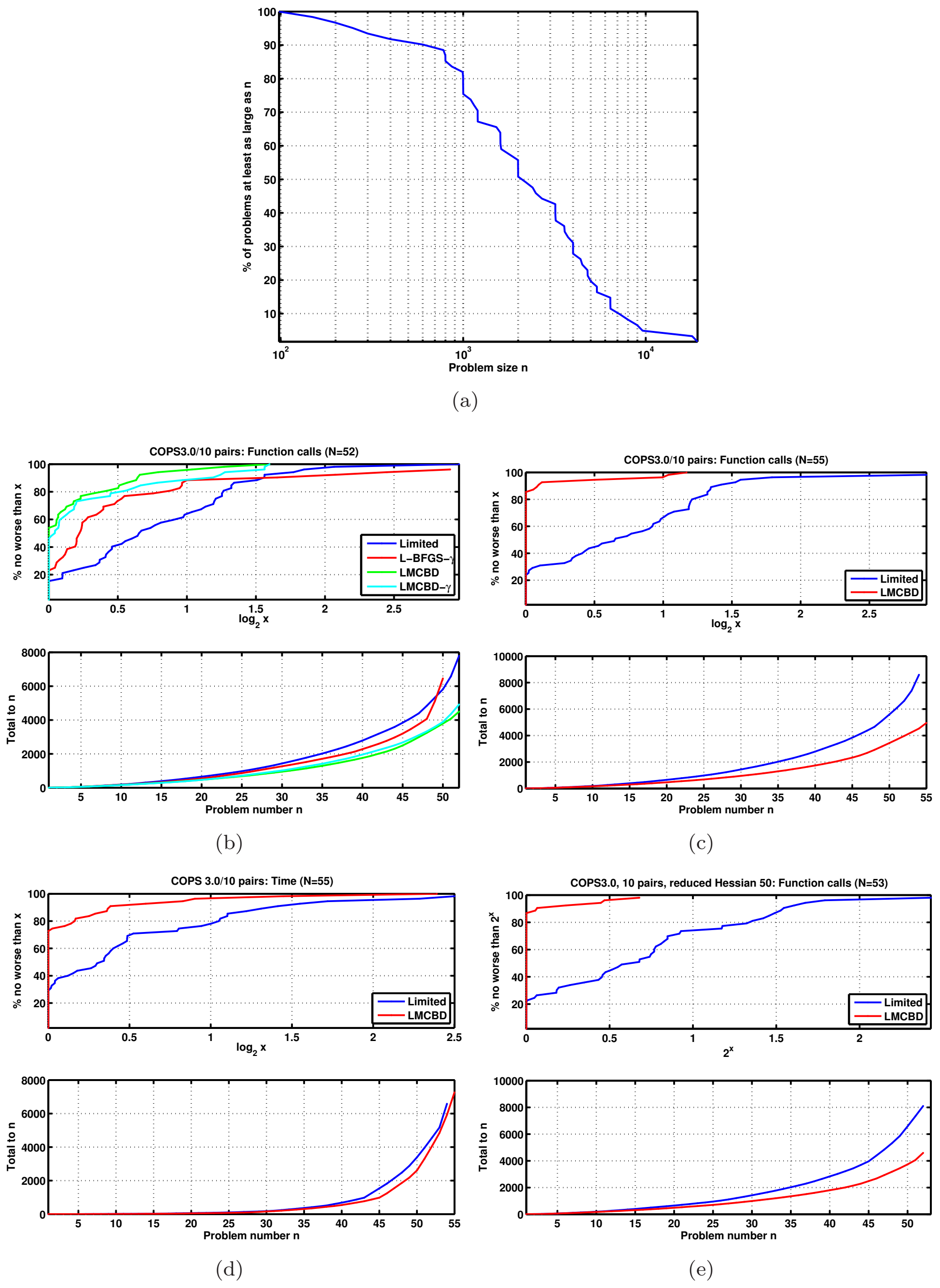
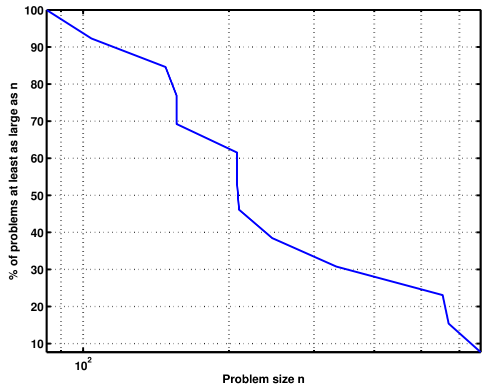
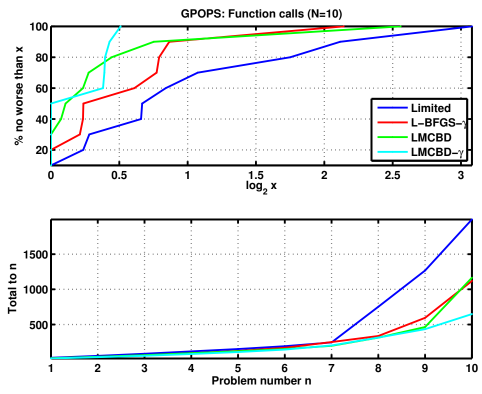


Figure 3.4: Results for SNOPT on the COPS 3.0 test set.

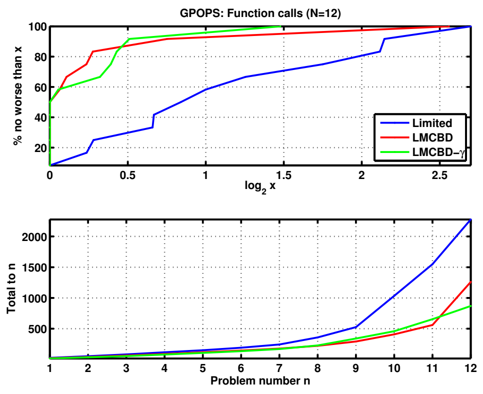
When we plot the results of multiple algorithms together, we have to discard problems for which different solutions are obtained. We also discard any problem on which every algorithm fails. Consequently, the number of problems in the figures may differ from the numbers indicated in this



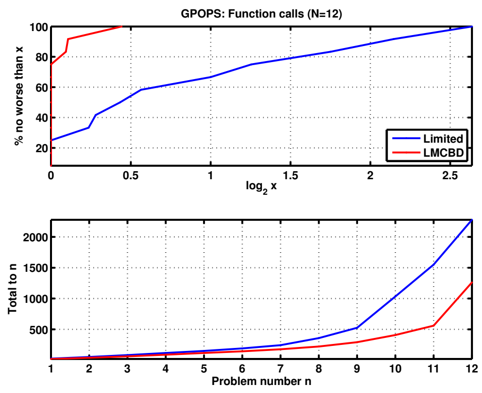
(a)



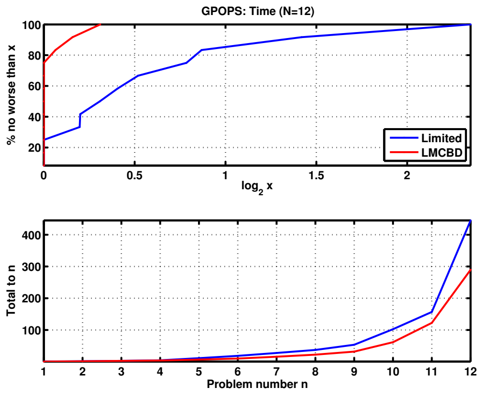
(b)



(c)



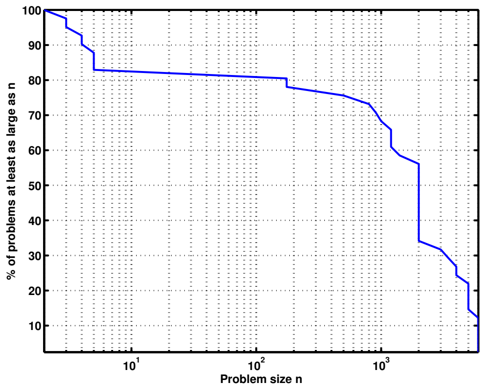
(d)



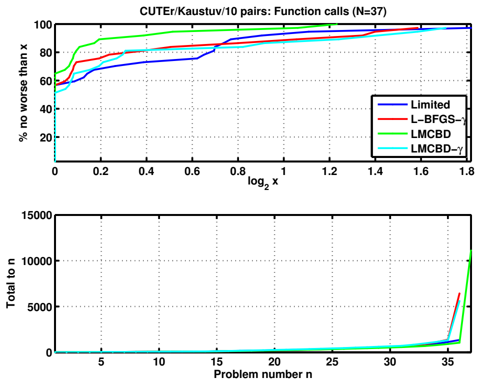
(e)

Figure 3.5: Results for SNOPT on the GPOPS example problems.

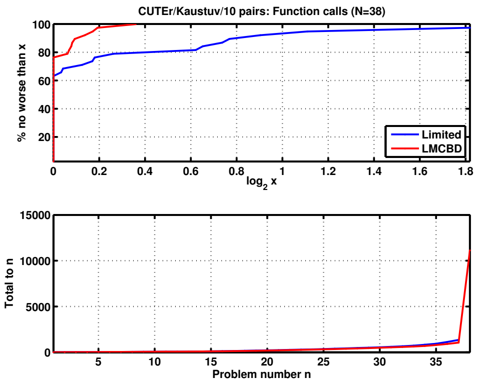
list.



(a)



(b)



(c)

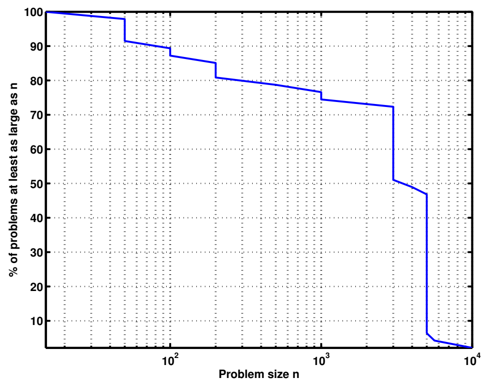
Figure 3.6: Results for SNOPT on the CUTEr/Kaustuv test set.

As in Chapter 2, we use performance profiles. Our primary performance metric is the number of user function evaluations; for two test sets, we also assess CPU time. We use a second plot that shows performance in a different way. Associated with each algorithm is a sorted list of metric values, one element for each problem. The sorted list is cumulatively summed and the result is plotted. The curve associated with a better-performing algorithm increases more slowly than that associated with a worse-performing algorithm.

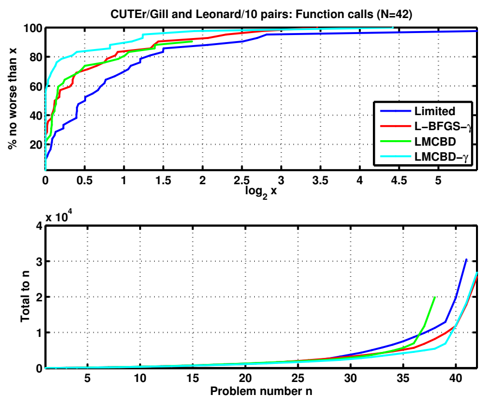
In plots showing problem sizes, the size refers to the number of variables in the problem.

We use the following specification file for SNOPT on the CUTEr problems:

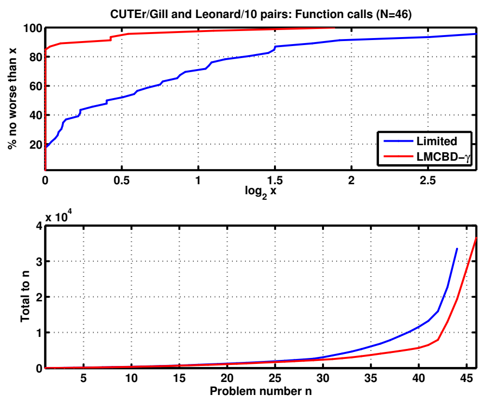
```
Begin SNOPT-Cuter NLP problem
   Major Print level           000001
   Minor print level           0
   Major iteration limit       10000
   Iterations limit            100000
   Solution                    no
   Hessian Limited
```



(a)



(b)



(c)

Figure 3.7: Results for SNOPT on the CUTEr/Gill and Leonard test set.

```
END SNOPT-Cuter NLP problem
```

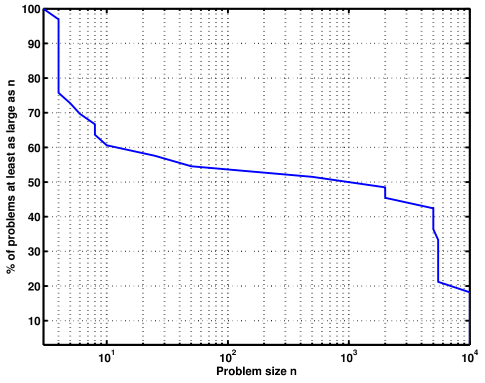
We use the default options for SNOPT when running on the COPS 3.0 problems. For the `lbfgs.f` code in the CUTEr framework, we use the following specification file:

```
      5     M
     10     IPRINT(1)
      0     IPRINT(2)
 100000     MAXIT
      1     diagopt
0.00001     EPS
```

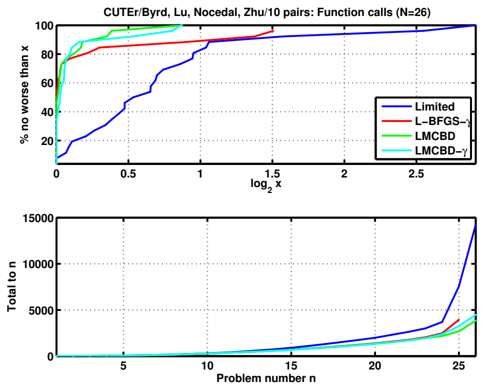
Results for modifications to `lbfgs.f`

Figure 3.3 shows results for `lbfgs.f` on the test set of CUTEr unconstrained problems. We tested the following methods:

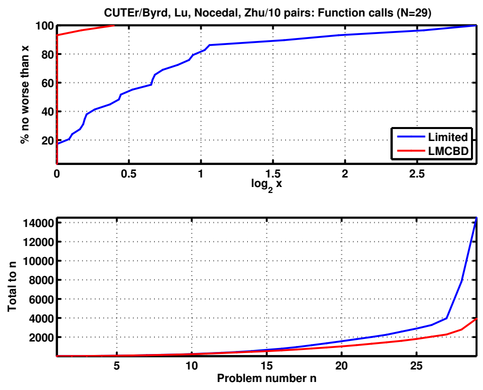
- L-BFGS. The original method.



(a)



(b)



(c)

Figure 3.8: Results for SNOPT on the CUTEr/Byrd, Lu, Nocedal, and Zhu test set.

- L-BFGS-γ. The original method, but σ_k is replaced with γ_k.
- LMCBD. Our primary algorithm.
- LMCBD-γ. Our secondary algorithm in which γ_k is used to moderate the diagonal update.
- LMCBD-ν. Our secondary algorithm, but γ_k is replaced with ν_k.

Every modification performs better on average than the original method. Figure 3.3(c) shows that LMCBD-γ outperforms LMCBD-ν: even in the case of unconstrained problems, the scalar γ is evidently better than ν. Similarly, 3.3(d) shows that L-BFGS-γ slightly outperforms L-BFGS: γ is also better than σ.

Although LMCBD outperforms L-BFGS, LMCBD-γ is the best of the algorithms tested.

Figure 3.3(f) shows results for just the subset of problems that are in the CUTEr/Leonard set.

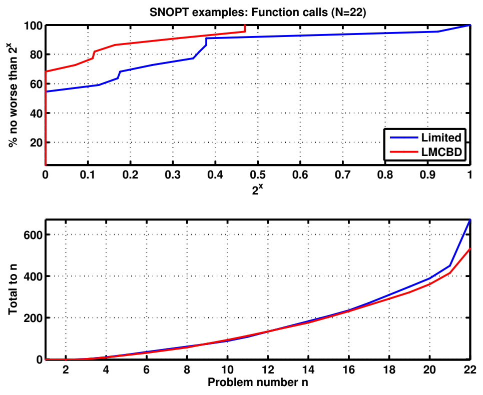


Figure 3.9: Results for SNOPT on the SNOPT example problems.

Results for SNOPT

We tested the modified SNOPT on six test sets having among them four interfaces: AMPL, CUTEr, MATLAB, and Fortran. We tested three modifications to SNOPT:

- LIMITED. SNOPT's current Hessian approximation.
- LMCBD. Our primary algorithm.
- LMCBD-γ. Our secondary algorithm in which γ_k is used to moderate the diagonal update.
- L-BFGS-γ. Our diagonal update is replaced by $\gamma_k I$. Note that γ, rather than σ, is used.

We feel our most important results are those shown in Figure 3.4 for the COPS 3.0 test set. The test set has generally constrained problems of high quality, and so we believe that the results on this set are particularly meaningful. Figure 3.4(b) shows results for all the methods. First, both forms of the LMCBD algorithm outperform the other methods. Second, L-BFGS outperforms LIMITED in general, but it does quite a bit worse on a few problems. We conclude that both a circular buffer and the diagonal update contribute to LMCBD's success. Figures 3.4(c,d) show results just for LMCBD and the original method. The second of the two uses CPU time in seconds as the performance metric. Figure 3.4(e) shows that LMCBD remains effective even when SNOPT is forced to use the conjugate gradient algorithm, rather than the Cholesky factorization, to solve the reduced Hessian system once the reduced Hessian's size exceeds 50. We use the default setting `CG Iterations` = 100. Until seeing the results of this test, one might be concerned that the richer information in LMCBD's Hessian approximation could limit the conjugate gradient algorithm's ability to reduce the residual of the reduced Hessian system in the limited number of iterations it is permitted.

Figure 3.5 shows results for the GPOPS example problems. Figures 3.5(b,c) show the results including and excluding, respectively, the results for L-BFGS, as including this algorithm reduces the relatively small test set size by 2 because of different local minimizers. Figure 3.5(e) uses CPU time as the performance metric. The results are qualitatively similar to those for the COPS 3.0 test set.

Figure 3.6 shows results for the CUTEr/Kaustuv set of constrained problems. LMCBD-γ fares relatively poorly, but LMCBD continues to be successful.

Figure 3.7 shows results for the CUTEr/Gill and Leonard test set of unconstrained problems. Figure 3.7(b) shows results similar to Figure 3.3(b,f): LMCBD-γ is superior to LMCBD on unconstrained problems. We discuss our interpretation of these results in the next section.

Figure 3.8 shows results for the CUTEr/Byrd, Lu, Nocedal, and Zhu test set of bound-constrained problems. Results are similar to those on the other test sets of constrained problems.

Finally, as a further check of the robustness of the software, we test SNOPT using the old and new Hessian approximations on the problems in SNOPT's `examples` directory. The results are shown in Figure 3.9.

3.6 Summary and conclusions

This chapter describes a limited-memory quasi-Newton method, LMCBD, that combines a diagonal update with a limited-memory quasi-Newton method based on a circular buffer.

Theorem 11 and subsequent analysis suggests that the proper interpretation of a diagonal initial matrix B_0 is that it *scales* the problem. Therefore, an effective B_0 is one that reduces the condition number of the Hessian by scaling. Motivated by this observation, we developed a diagonal update that theoretical analysis—albeit quite approximate—and numerical experiments show equilibrates the Hessian matrix. We combined the diagonal update with a limited-memory quasi-Newton method based on a circular buffer to create the method we call LMCBD.

We also discussed the importance of scaling the Hessian approximation by a scalar. This topic has received attention for several decades. σ is often used; and ν was introduced along with a diagonal update like ours by previous authors. σ and ν are interpreted as approximating the magnitude of the curvature in the part of the space for which curvature information is missing. In contrast, we interpret the scalar multiple as calibrating the unit step based on the most recent information. This observation motivates our adopting γ as LMCBD's scalar multiple.

We implemented our algorithms in `lbfgs.f` for experimental purposes and in SNOPT for eventual release. We conducted a broad range of numerical experiments and made several observations:

- γ is an effective scalar multiple and appears to be better than σ and ν.
- LMCBD is an effective algorithm, although LMCBD-γ performs better on the test set of unconstrained problems. We believe the latter holds because γ moderates the change in the diagonal. The unconstrained problems in our test set typically require many more iterations than the other test problems, and so moderate changes in the diagonal are beneficial. In contrast, when

relatively few major iterations are performed on constrained problems, a rapidly changing diagonal is useful.

- Using a circular buffer in SNOPT, regardless of the form of B_0, improves performance on a number of problems, but on some problems, performance is greatly reduced.
- Combining a circular buffer with a diagonal update, as is done in LMCBD, produces a robust Hessian approximation that on average outperforms SNOPT's current method.

Chapter 4

Interpolating quasi-Newton data from a coarse to a fine mesh

A differential equation is often solved by discretizing it on a sequence of increasingly fine meshes. Similarly, an optimization problem involving functionals and differential equations can be solved by discretizing the functionals to produce a sequence of numerical optimization problems defined on increasingly refined meshes. In this chapter, we develop a method to interpolate our limited-memory quasi-Newton matrix from a coarse to a fine mesh for problems of this type.

There is a large literature on differential-equation-constrained optimization problems (DECP). Roughly two types of problems are studied, generally by different researchers: *optimal control problems*, such as problems concerning the trajectory of a rocket, that usually involve the ordinary differential equations of mechanics [10, 3, 58]; and partial differential equation-constrained (*PDE-constrained*) problems [4]. Additionally, many researchers have studied methods that solve problems on multiple meshes of varying fineness: for example, geometric and algebraic multigrid methods [6].

Researchers have used quasi-Newton methods to solve DECPs. Almost 30 years ago, Griewank and Toint [27] proposed a specialized quasi-Newton method for objectives that are a sum of convex element functions. The QN matrix is partitioned according to the objective. They remarked, though did not describe in detail, that one can propagate the element Hessians from one mesh to another. Kelley and Sachs [32] developed rather technical conditions on the initial matrix B_0 so that a quasi-Newton method takes a number of iterations independent of the mesh fineness.

Interpolating a limited-memory quasi-Newton matrix has not received attention for a simple reason: in, say, L-BFGS, the data are not sufficiently valuable to justify the complexity of interpolating them from a coarse grid to a fine one. In contrast, the diagonal matrix B_0 in LMCBD is quite valuable, and so interpolation may be justified. This chapter develops such a method.

4.1 Interpolation rules

The primary tool of our method is a set of three interpolation rules: one each for s, y, and d, the quasi-Newton pair and diagonal of the diagonal matrix B_0. We establish these rules in the simpler setting of approximating integrals, then consider differential-equation-constrained optimization problems.

4.1.1 Integrals

Consider the functional

$$H[u] \equiv \int_\Omega h(x, u, \partial_x u, \partial_x^2 u, \ldots)\ \mathrm{d}x, \tag{4.1}$$

where $\partial_x^i u$ denotes the ith partial derivative of u. The first variation of $H[u]$ is

$$\delta H[u] = \int_\Omega \delta h\ \mathrm{d}x,$$

where δ denotes the variation. δh may be quite complicated, but if h and u are sufficiently smooth, as we assume here, and the variation δu is smooth by construction, then so is δh. (For more on the variational calculus, see, for example, [39].)

In an optimization problem, boundary conditions must be provided. For simplicity in the present exposition, we assume the boundary values of u are fixed and so the variation δu is 0 on the boundary $\partial\Omega$. Consequently,

$$\int_\Omega \delta h\ \mathrm{d}x = \int_\Omega \mathcal{H}u\ \delta u\ \mathrm{d}x$$

for a differential operator $\mathcal{H}$ and a smooth variation δu: the boundary terms that arise from integrating by parts vanish.

We consider discretizations of the integral in which the approximation has one of two forms: either it is centered at the nodes, or it is centered at cell centers. u is approximated by $\bar{u}$ and the variation δu by $\overline{\delta u}$. We assume both barred values are defined at the nodes. The approximation has the form

$$\delta H[u] = \int_\Omega \mathcal{H}u\ \delta u\ \mathrm{d}x \approx \sum_{i\in\mathcal{N}} \bar{\phi}_i \overline{\delta u}_i \mu_i, \tag{4.2}$$

where $\bar{\phi}_i$ is the approximation to $\mathcal{H}u$ at node location x_i, $\mathcal{N}$ is the set of nodes, and μ_i is the weight associated with node i.

From the other direction, $H[u]$ may be approximated by a node-centered or a cell-centered expression. For generality, we write

$$H[u] \approx \sum_{i \in \mathcal{N}} \bar{h}_{1i}(\bar{u})\mu_i + \sum_{i \in \mathcal{C}} \bar{h}_{2i}(\bar{u})\nu_i \equiv \bar{H}(\bar{u});$$

$\mathcal{C}$ is the set of cell indices, and ν_i is the weight associated with cell i. For future use, $\tilde{x}_i$ is the location of the ith cell center. The differential of $\bar{H}(\bar{u})$ is

$$\mathrm{d}_{\bar{u}}\bar{H}(\bar{u}) = \sum_{i \in \mathcal{N}} \partial_{\bar{u}_i}\bar{H}(\bar{u})\mathrm{d}\bar{u}_i.$$

If $\mathrm{d}\bar{u}_i = \overline{\delta u}_i$, then $\delta H[u] \approx \mathrm{d}_{\bar{u}}\bar{H}(\bar{u})$ and

$$\sum_{i \in \mathcal{N}} \bar{\phi}_i \overline{\delta u}_i \mu_i \approx \sum_{i \in \mathcal{N}} \partial_{\bar{u}_i}\bar{H}(\bar{u})\mathrm{d}\bar{u}_i.$$

Consequently, we make the identification $\partial_{\bar{u}_i}\bar{H}(\bar{u}) \approx \bar{\phi}_i\mu_i$. Additionally, by the approximation (4.2),

$$\partial_{\bar{u}_i}\bar{H}(\bar{u})/\mu_i \approx \mathcal{H}u|_{x=x_i}. \tag{4.3}$$

Suppose we have two meshes. The mesh to which a quantity belongs is denoted by a superscript 1 or 2. Let $\mathcal{I}$ interpolate either a node-centered or a cell-centered quantity. For example, we can interpolate the solution $\bar{u}^1$ on the first mesh to the second mesh:

$$\bar{u}^2 \leftarrow \mathcal{I}\bar{u}^1. \tag{4.4}$$

Similarly, we can interpolate $\bar{\phi}^2 \leftarrow \mathcal{I}\bar{\phi}^1$. We refer to (4.4) as the *first interpolation rule*. Both node-centered and cell-centered quantities may appear in a problem; we use the same symbol $\mathcal{I}$ for both cases, though the underlying implementation of $\mathcal{I}$ depends on the case.

To interpolate $\partial_{\bar{u}^1}\bar{H}(\bar{u})$ requires several extra steps. Because u does not vary on $\partial\Omega$—and, more generally, boundary conditions are such that $\bar{u}$ at the boundary nodes must be treated separately—the interpolation operator $\mathcal{I}$ must be modified to exclude the boundary. Generally, extrapolation is necessary for points near the boundary. Let the resulting operator be $\hat{\mathcal{I}}$. Given two vectors a and b, let a/b be element-wise division and ab be element-wise multiplication. Let the operator $\mathcal{R}$ restrict the vector a to interior nodes, and let the operator $\mathcal{E}$ expand the vector $\mathcal{R}a$ to all nodes. The latter operator combines the data in $\mathcal{R}a$ with boundary condition data. $\hat{\mathcal{I}}$ uses only the interior nodes; hence we adopt the convention that $\hat{\mathcal{I}}$ performs its own restriction: for example, $\hat{\mathcal{I}}(\mathcal{R}a)$ is redundant and is instead written simply as $\hat{\mathcal{I}}a$. $\mathcal{E}$ and $\mathcal{R}$ are the identity operators when applied to cell-centered quantities.

As

$$\delta H[u] \approx \sum_{i\in\mathcal{N}^1} \bar{\phi}_i^1 \overline{\delta u}_i^1 \mu_i^1 \approx \sum_{i\in\mathcal{N}^2} \bar{\phi}_i^2 \overline{\delta u}_i^2 \mu_i^2,$$

and recalling $\partial_{\bar{u}_i} \bar{H}(\bar{u}) \approx \bar{\phi}_i \mu_i$ and (4.3),

$$\partial_{\bar{u}^2} \bar{H}^2(\bar{u}^2) \approx \mathcal{E}(\mathcal{R}(\mu^2)\hat{\mathcal{I}}(\partial_{\bar{u}^1} \bar{H}^1(\bar{u}^1)/\mu^1)).$$

Consequently, the interpolation rule for the gradient $g \equiv \partial_{\bar{u}} \bar{H}(\bar{u})$ is

$$g^2 \leftarrow \mathcal{E}(\mathcal{R}(\mu^2)\hat{\mathcal{I}}(g^1/\mu^1)) :$$

in words, divide the gradient by the node weights on mesh 1 to obtain the discretization of a smooth quantity, interpolate the result, then multiply by the node weights on mesh 2. For future use, we need to write this rule slightly more generally. Let $\chi_i = \mu_i$ if g is node-centered; and $\chi_i = \nu_i$, if cell-centered. Then

$$g^2 \leftarrow \mathcal{E}(\mathcal{R}(\chi^2)\hat{\mathcal{I}}(g^1/\chi^1)).$$

This is the *second interpolation rule*.

For convenience, we denote the first rule by $\mathcal{I}_s$: $\mathcal{I}(\bar{u}^1) = \mathcal{I}_s(\bar{u}^1)$; and the second by $\mathcal{I}_y$: $\mathcal{I}_y(g^1) = \mathcal{E}(\mathcal{R}(\mu^2)\hat{\mathcal{I}}(g^1/\mu^1))$.

As a check on our notation, let us evaluate an integral on two meshes. Suppose $\bar{u}^2$ and $\partial_{\bar{u}^2} \bar{H}^2(\bar{u}^2)$ are interpolated from $\bar{u}^1$ and $\partial_{\bar{u}^1} \bar{H}^1(\bar{u}^1)$, respectively, according to the two interpolation rules; and each quantity is node-centered. Let u take prescribed values on the boundary. We have

$$\begin{aligned}
\int_\Omega u\ \mathcal{H}u\ \mathrm{d}x &\approx \sum_{i\in\mathcal{N}^2} \bar{u}_i^2 \bar{\phi}_i^2 \mu_i^2 && \text{(approximation to the integral on mesh 2)} \\
&\approx (\bar{u}^2)^T \partial_{\bar{u}^2} \bar{H}^2(\bar{u}^2) && (\partial_{\bar{u}^2} \bar{H}^2(\bar{u}^2) \approx \bar{\phi}^2 \mu^2) \\
&= \mathcal{I}_s(\bar{u}^1)^T \mathcal{I}_y(\partial_{\bar{u}^1} \bar{H}^1(\bar{u}^1)) && \text{(substitution of the two interpolation rules)} \quad (4.5) \\
&= \mathcal{I}(\bar{u}^1)^T \mathcal{E}(\mathcal{R}(\mu^2)\hat{\mathcal{I}}(\partial_{\bar{u}^1} \bar{H}^1(\bar{u}^1)/\mu^1)) && \text{(definition of } \mathcal{I}_s \text{ and } \mathcal{I}_y) \\
&\approx \mathcal{I}(\bar{u}^1)^T \mathcal{E}(\mathcal{R}(\mu^2)\hat{\mathcal{I}}(\bar{\phi}^1)) && (\partial_{\bar{u}^1} \bar{H}^1(\bar{u}^1) \approx \bar{\phi}^1 \mu^1) \\
&\approx \sum_{i\in\mathcal{N}^1} \bar{u}_i^1 \bar{\phi}_i^1 \mu_i^1 && \text{(approximation on mesh 1).}
\end{aligned}$$

In the final line, we assume each mesh is sufficiently fine so that the approximations to the integral on the two meshes yield approximately the same values.

An example in one dimension

Consider the functional

$$H[u] \equiv \frac{1}{2}\int_{-1}^{1} u^2 + u_x^2 \ \mathrm{d}x$$

subject to Dirichlet boundary conditions on u. The first variation δH for a variation δu for which the end points are 0 is

$$\begin{aligned}
\delta H[u] &= \frac{1}{2}\int_{-1}^{1} \delta u^2 + \delta u_x^2 \ \mathrm{d}x \\
&= \frac{1}{2}\int_{-1}^{1} 2u\delta u + 2u_x\delta u_x \ \mathrm{d}x \\
&= \int_{-1}^{1} u\delta u \ \mathrm{d}x + \int_{-1}^{1} u_x\delta u_x \ \mathrm{d}x \\
&= \int_{-1}^{1} u\delta u \ \mathrm{d}x - \int_{-1}^{1} u_{xx}\delta u \ \mathrm{d}x + u_x\delta u|_{-1}^{1} \\
&= \int_{-1}^{1} (u - u_{xx})\delta u \ \mathrm{d}x = \int_{-1}^{1} \mathcal{H}u\delta u \ \mathrm{d}x,
\end{aligned}$$

where $\mathcal{H}u \equiv u - u_{xx}$.

Now we develop approximations to the integral. Discretize $[-1, 1]$ by N cells with node indices $\mathcal{N} \equiv \{0, 1, \ldots, N\}$; $x_0 = -1$, $x_N = 1$, and $x_{i+1} > x_i$. Let the cell indices be $\mathcal{C} \equiv \mathcal{N} \setminus \{0\}$. The term u^2 of the integrand is approximated by a node-centered sum; the term u_x^2, by a cell-centered sum:

$$\begin{aligned}
\frac{1}{2}\int_{-1}^{1} u^2 \ \mathrm{d}x &\approx \sum_{i\in\mathcal{N}} \bar{u}_i^2 \mu_i \equiv \bar{H}_1(\bar{u}) \\
\frac{1}{2}\int_{-1}^{1} u_x^2 \ \mathrm{d}x &\approx \sum_{i\in\mathcal{C}} \left(\frac{\bar{u}_i - \bar{u}_{i-1}}{x_{i-1} - x_i}\right)^2 \nu_i \equiv \bar{H}_2(\bar{u}),
\end{aligned}$$

where

$$\begin{aligned}
\mu_i &= \frac{1}{2}\begin{cases} x_1 - x_0 & \text{if } i = 0 \\ x_{i+1} - x_{i-1} & \text{if } 0 < i < N \\ x_N - x_{N-1} & \text{if } i = N \end{cases} \\
\nu_i &= x_i - x_{i-1}
\end{aligned}$$

and $\bar{H} \equiv \bar{H}_1 + \bar{H}_2$.

We test our results so far by plotting several quantities for a particular function u discretized on two meshes in Figure 4.1. Mesh 1 is a combination of 20 linearly spaced points and 20 randomly

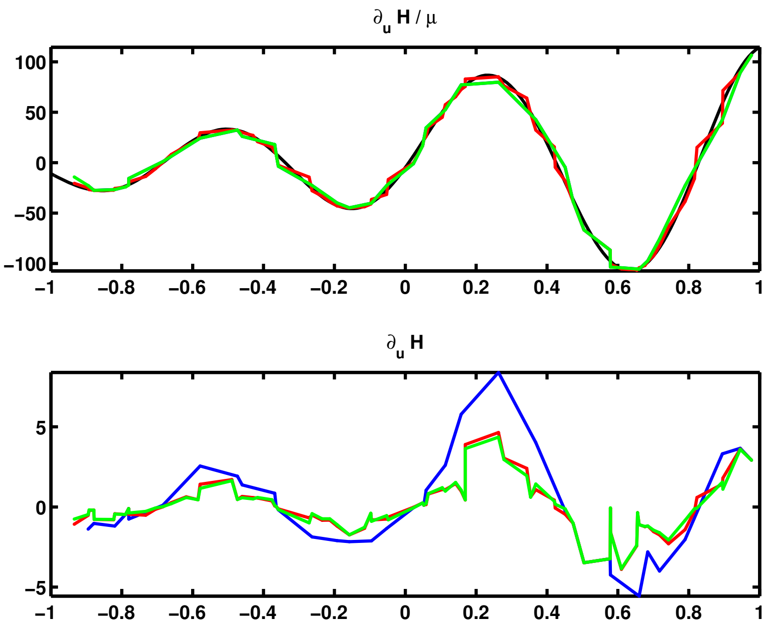


Figure 4.1: Illustration of the interpolation rules.

placed points. Mesh 2 augments mesh 1 by adding 40 more randomly placed points. We use cubic spline interpolation to define $\mathcal{I}$. In the top panel, all other quantities are compared with $\mathcal{H}u = u - u_{xx}$ (black): $\partial_{\bar{u}^i} \bar{H}^i(\bar{u}^i)/\mu^i$ for two meshes $i = 1, 2$ (blue and red, respectively) and $\mathcal{I}_y(\partial_{\bar{u}^1} \bar{H}^1(\bar{u}^1))/\mu^2$ (green). In the bottom panel, the latter three quantities are plotted with their denominators removed: these curves are the gradients for the discrete problem. The green and red curves should be compared: the first is the interpolation of the gradient from mesh 1 to 2, while the second is the exact gradient on mesh 2.

4.1.2 Differential equation constraints

Consider the DECP

$$\begin{aligned} &\inf_u f(p) \\ \text{subject to } &D^k[u; p] = 0 \\ &\text{boundary conditions,} \end{aligned} \tag{4.6}$$

where p is a set of parameters, f is a function of p, u is a function defined over the domain Ω, and D is an operator involving a differential operator. k indexes a particular differential equation in the system of equations; separating the differential equations may be necessary for the discretization step. The objective could be a functional, such as $H[u]$. But one can generally reformulate the problem: introduce a differential equation for the integrand in H and append it to D; introduce an equation that involves a new parameter q, appended to p, and u on the boundary; finally, set the objective to a function of q.

Discretizing (4.6) yields the numerical optimization problem

$$\begin{aligned} \min_{\bar{u}} \; & f(p) \\ \text{subject to } & \bar{D}^k(\bar{u}, p) = 0 \\ & \text{boundary conditions.} \end{aligned} \tag{4.7}$$

Corresponding to (4.6) and (4.7), respectively, are the Lagrangians

$$\begin{aligned} \mathcal{L} &\equiv f(p) - \sum_k \int_\Omega \lambda^k D^k[u] \; \mathrm{d}x + [\text{boundary condition terms}] \\ \bar{\mathcal{L}} &\equiv \bar{f}(p) - \sum_k \sum_i \bar{\lambda}_i^k \bar{D}_i^k(\bar{u}) + [\text{boundary condition terms}]. \end{aligned} \tag{4.8}$$

The index set over which the inner sum occurs in $\mathcal{L}$ depends on the discretization of the differential equations.

$\mathcal{L} \approx \bar{\mathcal{L}}$ and, similarly, corresponding terms in the two Lagrangians are approximately equal. From these relationships, we can obtain the interpolation rules. $\bar{u}$ and $D[\bar{u}]$ are functions and so are interpolated by the first rule:

$$\begin{aligned} \bar{u}^2 &\leftarrow \mathcal{I}_s(\bar{u}^1) \\ D^2(\bar{u}^2) &\leftarrow \mathcal{I}_s(D^1(\bar{u}^1, p)). \end{aligned}$$

Each of the integrals in (4.8) has the form of the integral (4.1). Hence the interpolation rule for gradients applies. Let $\bar{\mathcal{L}}^i$ be discretizations of the continuous Lagrangian on meshes $i = 1, 2$. Then we interpolate by the second interpolation rule:

$$\bar{\mathcal{L}}^2_{\bar{u}^2}(\bar{u}^2) \leftarrow \mathcal{E}(\mathcal{R}(\mu^2)\hat{\mathcal{I}}(\bar{\mathcal{L}}^1_{\bar{u}^1}(\bar{u}^1)/\mu^1)).$$

4.1.3 Diagonal B_0

The primary application of our interpolation rules is to interpolate the quasi-Newton pairs $\{s, y\}$ and the diagonal matrix $B_0 \equiv \operatorname{diag}(d)$ from one mesh to another. In the context of the model problem (4.7), $s = \bar{u}^+ - \bar{u}$ and $y = \bar{\mathcal{L}}^+_{\bar{u}} - \bar{\mathcal{L}}_{\bar{u}}$, and so we set

$$\begin{aligned} s^2 &\leftarrow \mathcal{I}_s(s^1) \\ y^2 &\leftarrow \mathcal{I}_y(y^1). \end{aligned}$$

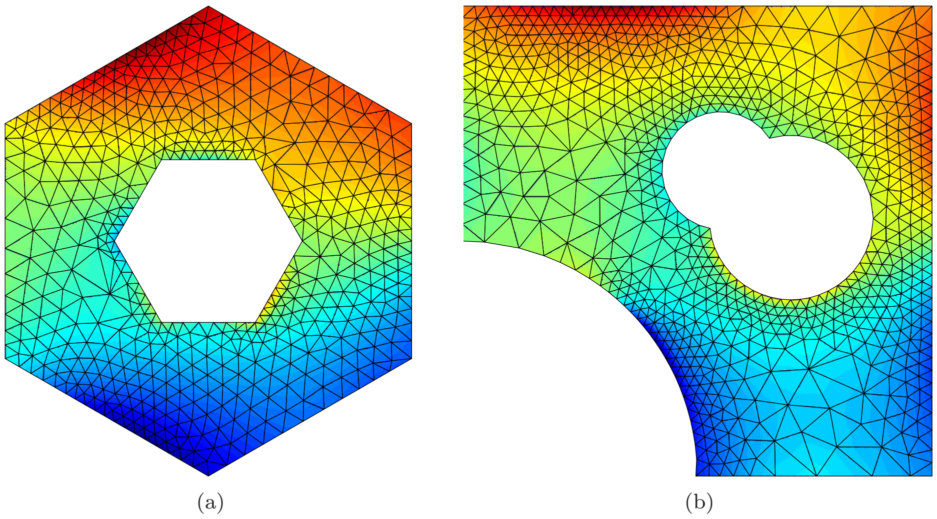


Figure 4.2: Solutions to the two minimal surface problems on intermediate meshes.

These interpolations motivate the shorthand $\mathcal{I}_s$ and $\mathcal{I}_y$.

We can determine an interpolation rule for $B_0 \equiv \operatorname{diag}(d)$ as follows. d is updated by

$$d^+ \leftarrow d - \frac{(ds)^2}{s^T(ds)} + \frac{y^2}{s^T y}. \tag{4.9}$$

Consider the final term, $y^2/s^T y$. From (4.5), we know that $s^T y$ is an approximation to an integral and so varies over different meshes only by the truncation error. y is interpolated according to the rule for gradients. Unfortunately, y itself is lost if one has access only to d. We adopt a third interpolation rule for this case:

$$d^2 \leftarrow \mathcal{E}(\mathcal{R}(\mu^2)\mathcal{I}(\sqrt{d^1}/\mu^1))^2 = \mathcal{I}_y(\sqrt{d^1})^2.$$

First the square root of $(y^1)^2$ is taken; then the result is interpolated according to the second interpolation rule; finally, the result is squared.

4.2 Numerical experiments

We test our interpolation rules on two types of problems: the two-dimensional minimal surface problem and optimal control problems.

	Solver parameters						# nodes / # ufn					Time (s)
	INTERP-SOLN	USE-LMCBD	INTERP-SY	SAVE-SY	INTERP-DIAG	INIT-DIAG	265	373	567	2191	4057	
1							54	74	96	177	242	778.9
2		✓					45	57	69	139	181	583.9
3		✓				✓	45	55	77	130	174	559.0
4	✓						54	59	83	133	187	594.1
5	✓	✓					45	43	58	102	124	412.3
6	✓	✓				✓	45	45	60	84	112	368.1
7	✓	✓			✓		45	47	57	88	110	367.1
8	✓	✓	✓		✓		45	46	53	88	110	368.5
9	✓	✓	✓	✓	✓		45	42	51	81	106	352.1

Table 4.1: Results for the problem MS-1.

	Solver parameters						# nodes / # ufn					Time (s)
	INTERP-SOLN	USE-LMCBD	INTERP-SY	SAVE-SY	INTERP-DIAG	INIT-DIAG	341	198	814	3232	12715	
1							82	64	159	304	539	8374.5
2		✓					60	44	105	190	382	5877.8
3		✓				✓	58	46	121	216	407	6290.4
4	✓						82	48	141	204	319	4995.8
5	✓	✓					60	37	94	134	241	3795.6
6	✓	✓				✓	58	37	95	144	231	3647.8
7	✓	✓			✓		60	39	95	118	198	3084.1
8	✓	✓	✓		✓		60	38	90	124	184	2882.1
9	✓	✓	✓	✓	✓		60	36	83	106	169	2654.1

Table 4.2: Results for the problem MS-2.

4.2.1 The minimal surface problem

The minimal surface problem on a two-dimensional domain minimizes the area of a surface subject to Dirichlet boundary conditions on the boundary:

$$\inf_u \int_\Omega \sqrt{1+|\nabla u|^2}\, \mathrm{d}\Omega.$$

We solve the problem using a finite-element method and first-order triangular elements. We use the MATLAB software `distmesh` [56] to discretize the domain. The objective for the resulting unconstrained numerical optimization problem is

$$F(\bar{u}) \equiv \sum_{i \in \mathcal{C}} \sqrt{1 + (z_x(\bar{u}))_i^2 + (z_y(\bar{u}))_i^2}\ \nu_i,$$

where the index $i \in \mathcal{C}$ is over the triangle (cell) indices, $\bar{u}$ is the vector of surface heights at the nodes, $\nu_i = A_i$ is the area of the ith triangle, and $(z_x)_i$ and $(z_y)_i$ are the components of the gradient of the plane over that triangle. We can reorder the operations so that the objective is written

$$F(\bar{u}) = \sum_{i \in \mathcal{N}} f_i(\bar{u})\ \mu_i \tag{4.10}$$

for some set of nonlinear functions f_i. Here the index i is over the node indices and μ_i is the sum of the areas of the triangles involving node i.

Specifying a problem requires creating a domain and setting the Dirichlet boundary conditions. We consider two problems; we call them MS-1 and MS-2. Their solutions on intermediate meshes are shown in Figure 4.2.

A problem is solved on a sequence of meshes. The first mesh is coarse and uniform. Each subsequent mesh is finer than the previous, and the area of a triangle is adapted to the solution on the preceding mesh. Since the problems are solved to a high precision on each mesh, the sequence of meshes is the same for all parameter variations. Indeed, to save time, we generate the meshes once and then use them for all other parameter variations.

We developed an unconstrained solver in MATLAB that implements the L-BFGS [50] method with our variations and uses the line search routine `cvsrch.m` [53]. The solver parameters are all boolean valued. In all cases, the circular buffer can store 10 pairs. Varying parameters include the following (abbreviations refer to our tables of results):

- INTERP-SOLN. Interpolate the solution $\bar{u}$ using cubic spline interpolation.
- USE-LMCBD. If true, use LMCBD; otherwise, use the standard L-BFGS algorithm.
- INTERP-SY. Interpolate the pairs $\{s, y\}$. The pairs that are interpolated are acquired when the magnitude of the gradient is the square root of the desired tolerance. The pairs obtained in the final iterations tend not to be as helpful.
- INTERP-DIAG. Interpolate the final diagonal d from the iteration on the coarse mesh to the fine mesh.
- INIT-DIAG. Initialize the diagonal d in $B_0 = \text{diag}(d)$ to $\mathcal{R}(\mu^2)$. If INTERP-DIAG is true, then ignore this option except on the coarsest mesh. This option uses exactly the same information and user code as interpolating the QN matrix.

- SAVE-SY. If INTERP-QN is true, retain the interpolated pairs in a static buffer and never discard them. Consequently, the QN matrix for a mesh has up to 10 more pairs in its static buffer than the QN matrix for the previous mesh. When computing a matrix-vector product, the order of the pairs is the static buffer from oldest to newest, then, as usual, the circular buffer from oldest to newest.

Tables 4.1 and 4.2 show our results. The left column numbers the different tests. The check marks indicate which solver options are active. In the "# nodes / # ufn" block, each column shows the number of user function calls required to solve the problem on the mesh having the indicated number of nodes. The right column shows the accumulated time in seconds to solve the problem on all the meshes.

The results accord with the hypothesis that interpolating the solution and quasi-Newton matrix from one mesh to another is useful.

Lines 1 and 5 are controls. In line 1, each problem is solved independently of the others and L-BFGS is used.

Lines 2 and 3 show that using LMCBD improves performance, relative to line 1, on all meshes; and initializing the diagonal is useful.

In line 4, the solution is interpolated. The results shown in all subsequent lines should be compared with this one.

Lines 5 and 6 are to line 4 as lines 2 and 3 are to 1; and the two sets of results have similar behavior.

In line 7, the diagonal matrix B_0 is interpolated in addition to the solution. Lines 6 and 7 suggest that initializing B_0 to $\mathcal{R}(\mu^2)$ or interpolating the previous mesh's B_0 are about equally effective.

Lines 8 and 9 interpolate the $\{s, y\}$ pairs in addition to B_0; line 8 simply initializes the new QN matrix with them, while line 9 saves them in a static buffer that grows with each succeeding mesh.

For MS-1, lines 6 to 9 show about equal performance; for MS-2, these lines show performance that continues to improve significantly as more aggressive methods are applied.

4.2.2 Optimal control

Next we implement the interpolation rules to solve a set of optimal control problems. We examine four problems:

- BRACH. We use the energy formulation of the brachistochrone problem. The position of the ball is given by (x, y). At the initial time, it has zero velocity, height, and position in the x direction; at the final time, it has height -1 and x position 1. The ball's kinetic energy is $E_k \equiv \frac{1}{2}(\dot{x}^2 + \dot{y}^2)$ and its potential energy is $E_p \equiv gy$. The energy of the ball is conserved, and so $\Delta E_k + \Delta E_p = 0$. The ODE relate position and velocity.

- GLIDER. The glider problem from COPS 3.0 [20].

	USE-LMCBD	INTERP-DIAG	# nodes / # ufn								Time (s)
			10	19	37	73	145	289	577	1153	
1			99	47	61	75	100	124	135	166	682.3
2		✓	99	21	24	25	24	34	26	34	80.0
3	✓		286	31	42	59	68	76	90	73	344.2
4	✓	✓	286	14	16	27	45	31	55	39	127.9

	USE-LMCBD	INTERP-DIAG	# nodes / # ufn						Time (s)
			10	20	40	80	160	320	
1			99	57	84	136	180	207	108.3
2		✓	99	42	39	38	55	80	40.3
3	✓		286	44	63	83	120	155	80.5
4	✓	✓	286	21	24	22	32	40	23.5

Table 4.3: Results for the problem BRACH.

	USE-LMCBD	INTERP-DIAG	# nodes / # ufn					Time (s)
			10	19	37	73	145	
1			67	61	41	64	72	22.2
2		✓	67	34	54	48	56	16.7
3	✓		34	62	32	40	73	19.7
4	✓	✓	34	37	29	35	34	12.4

	USE-LMCBD	INTERP-DIAG	# nodes / # ufn					Time (s)
			10	20	40	80	160	
1			67	105	249	224	269	86.8
2		✓	67	60	82	72	88	28.5
3	✓		34	43	207	204	840	180.5
4	✓	✓	34	175	59	43	78	25.4

Table 4.4: Results for the problem GLIDER.

- ROCKET. The rocket problem from COPS 3.0.
- VANDERPOL. A test problem from SNOPT's test set.

We discretize all the problems except GLIDER using the trapezoid rule. We believe because of the formulation of the control, the trapezoid rule for GLIDER can produce a problem having meaningless solutions. As this issue is unrelated to our work, we simply use the implicit Euler rule for this problem. We present results for both uniform and randomly generated nonuniform meshes. The latter is meant to simulate adaptive refinement, for we did not implement true adaptive refinement in these tests.

	USE-LMCBD	INTERP-DIAG	# nodes / # ufn					Time (s)
			10	19	37	73	145	
1			45	25	28	29	9	6.5
2		✓	45	11	10	29	21	7.3
3	✓		27	19	34	73	15	10.6
4	✓	✓	27	27	34	25	8	6.2

	USE-LMCBD	INTERP-DIAG	# nodes / # ufn					Time (s)
			10	20	40	80	160	
1			45	15	25	37	48	14.4
2		✓	45	10	11	20	16	6.6
3	✓		27	33	17	19	34	10.3
4	✓	✓	27	22	19	21	19	7.6

Table 4.5: Results for the problem ROCKET.

	USE-LMCBD	INTERP-DIAG	# nodes / # ufn					Time (s)
			10	19	37	73	145	
1			39	10	9	8	8	3.3
2		✓	39	20	18	13	17	5.3
3	✓		16	9	10	9	9	3.2
4	✓	✓	16	12	12	14	12	4.1

	USE-LMCBD	INTERP-DIAG	# nodes / # ufn					Time (s)
			10	20	40	80	160	
1			39	18	24	26	32	9.0
2		✓	39	27	30	28	51	12.5
3	✓		16	16	22	22	28	7.8
4	✓	✓	16	14	22	19	28	7.6

Table 4.6: Results for the problem VANDERPOL.

Recall that the interpolation rules act on only the interior nodes. We set the value of a boundary node on the fine mesh to its value on the coarse mesh.

We test four SNOPT parameter variations. We use either the original `Hessian = Limited` QN method or the new `Hessian = LMCBD` method; and we either interpolate just the solution, or both the solution and B_0. Enabling interpolation of the QN matrix requires altering SNOPT; unfortunately, these alterations at present are little more than a hack and so cannot be released.

Tables 4.3–4.6 summarize the results. In each figure, the first table shows results for the sequence of uniform meshes; the second, the nonuniform meshes. Our objective in performing these tests is to validate the interpolation rules on generally constrained optimization problems; exact performance details are not of particular interest.

On the problems BRACH and GLIDER, LMCBD generally outperforms SNOPT's original QN method, and interpolating B_0 is beneficial. On the problem ROCKET, interpolating the diagonal improves performance, but LMCBD and SNOPT's original method perform about equally well. On the problem VANDERPOL, all variations perform about equally well.

4.3 Summary and conclusions

We derived three interpolation rules—one each for s, y, and $B_0 = \text{diag}(d)$—for our limited-memory quasi-Newton method based on identifying correspondences in the continuous and discretized optimization problems. Then we tested the rules on two 2D finite-element problems and four optimal control problems. We found that the interpolation rules improve performance (MS-1, MS-2, BRACH, GLIDER, ROCKET)—often markedly (MS-2, BRACH, GLIDER)—or at least do not reduce it (VANDERPOL).

Unlike the methods reported in Chapters 2 and 3, these interpolation rules cannot be implemented as a black box; they require problem-dependent information. One application might be to implement these rules within optimal control or PDE-constrained packages that implement particular discretization methods. Though a package's developer would have to implement the method, the package's users would not.

Bibliography

[1] C. Bekas, E. Kokiopoulou, and Y. Saad. An estimator for the diagonal of a matrix. *Appl. Num. Math.*, 57(11–12):1214–1229, 2007.

[2] D. P. Bertsekas. *Nonlinear Programming.* Athena Scientific, 1995.

[3] J. T. Betts. Survey of numerical methods for trajectory optimization. *J. of Guidance, Control, and Dynamics*, 21(2), 1998.

[4] L. T. Biegler, O. Ghattas, M. Heinkenschloss, and B. van Bloemen Waanders, editors. *Large-Scale PDE-Constrained Optimization*. Springer, 2003.

[5] A. Borobia and R. Canto. Matrix scaling: A geometric proof of Sinkhorn's theorem. *Lin. Alg. Appl.*, 268:1–8, 1998.

[6] W. L. Briggs, V. E. Henson, and S. F. McCormick. 2nd edition, 2000.

[7] R. A. Brualdi. The DAD theorem for arbitrary row sums. *Proc. of the American Math. Soc.*, 45(2):189–194, 1974.

[8] R. A. Brualdi. Matrices of 0's and 1's with total support. *J. of Comb. Theory*, 28(3):249–256, 1980.

[9] R. A. Brualdi, S. V. Parter, and H. Schneider. The diagonal equivalence of a nonnegative matrix to a stochastic matrix. *J. Math. Anal. Appl.*, 16:31–50, 1966.

[10] A. E. Bryson and Y.-C. Ho. *Applied Optimal Control: Optimization, Estimation, and Control*. Halsted Press, 1981.

[11] J. R. Bunch. Equilibration of symmetric matrices in the max-norm. *JACM*, 18(4):566–572, 1971.

[12] R. H. Byrd, P. Lu, J. Nocedal, and C. Y. Zhu. A limited memory algorithm for bound constrained optimization. *SIAM Journal on Scientific Computing*, 16(6):1190–1208, 1995.

[13] R. H. Byrd, J. Nocedal, and R. B. Schnabel. Representations of quasi-Newton matrices and their use in limited-memory methods. *Math. Program.*, (63):129–156, 1994.

[14] T.-Y. Chen and J. W. Demmel. Balancing sparse matrices for computing eigenvalues. *Lin. Alg. Appl.*, 309:261–287, 2000.

[15] T. M. Cover and J. A. Thomas. *Elements of Information Theory.* John Wiley and Sons, 2006.

[16] J. Csima and B. N. Datta. The DAD theorem for symmetric non-negative matrices. *J. Comb. Theory*, 12(1):147–152, 1972.

[17] T. A. Davis. The University of Florida sparse matrix collection. `http://www.cise.ufl.edu/research/sparse/matrices`.

[18] T. A. Davis. *Direct Methods for Sparse Linear Systems.* SIAM, 2006.

[19] E. D. Dolan and J. J. Moré. Benchmarking optimization software with performance profiles. *Math. Program.*, 91(2):201–213, 2002.

[20] E. D. Dolan, J. J. Moré, and T. S. Munson. Benchmarking optimization software with COPS 3.0. Technical Report ANL/MCS-TM-273, Argonne National Lab., 2004.

[21] G. E. Forsythe and E. G. Straus. On best conditioned matrices. *Proc. Amer. Math. Soc.*, 6:340–345, 1955.

[22] J.-Ch. Gilbert and C. Lemaréchal. Some numerical experiments with variable storage quasi-Newton algorithms. *Math. Program.*, 45:407–435, 1989.

[23] P. E. Gill and M. W. Leonard. Reduced-Hessian quasi-Newton methods for unconstrained optimization. *Numerical Analysis Report NA 96-4 (Revised Feb. 1999), U. of California, San Diego*, 1995.

[24] P. E. Gill and M. W. Leonard. Limited-memory reduced-Hessian methods for large-scale unconstrained optimization. *SIAM J. Optim.*, 14(2):380–401, 2003.

[25] P. E. Gill, W. Murray, and M. A. Saunders. SNOPT: An SQP algorithm for large-scale constrained optimization. *SIAM Review*, 47(1):99–131, 2005.

[26] P. E. Gill, W. Murray, and M. H. Wright. *Practical Optimization.* Academic Press, 1981.

[27] A. Griewank and Ph. L. Toint. Partitioned variable metric updates for large structured optimization problems. *Numer. Math.*, 39:119–137, 1982.

[28] R. A. Horn and C. R. Johnson. *Matrix Analysis.* Cambridge, U.K.: Cambridge Univ. Press, 1985.

[29] H. M. Huynh. *A Large-Scale Quadratic Programming Solver Based on Block-LU Updates of the KKT System*. PhD thesis, Stanford University, 2008.

[30] C. Johnson and R. Reams. Scaling of symmetric matrices by positive diagonal congruence. *Lin. and Multilin. Alg.*, 57(2):123–140, 2009.

[31] Kaustuv. *IPSOL: An Interior Point Solver for Nonconvex Optimization Problems*. PhD thesis, Stanford University, 2008.

[32] C. T. Kelley and E. W. Sachs. Quasi-Newton methods and unconstrained optimal control problems. *SIAM J. Control and Optim.*, 25(6):1503–1516, 1987.

[33] L. Khachiyan and B. Kalantari. Diagonal matrix scaling and linear programming. *SIAM J. on Optim.*, 2(4):668–672, 1992.

[34] J. Kleinberg and E. Tardos. *Algorithm Design*. Pearson/Addison-Wesley, 2006.

[35] P. A. Knight. The Sinkhorn-Knopp algorithm: Convergence and applications. *SIMAX*, 30(1):261–275, 2008.

[36] P. A. Knight and D. Ruiz. A fast algorithm for matrix balancing. In *Web Information Retrieval and Linear Algebra Algorithms*, 2007.

[37] T. G. Kolda, D. P. O'Leary, and L. Nazareth. BFGS with update skipping and varying memory. *SIAM J. Optim.*, 8(4):1060–1083, 1998.

[38] H. J. Kushner and G. Yin. Stochastic approximation and recursive algorithms and applications. 2003.

[39] C. Lanczos. *The Variational Principles of Mechanics*. Courier Dover Publications, 1986.

[40] D. C. Liu and J. Nocedal. On the limited memory BFGS method for large-scale optimization. *Math. Program.*, 45:503–528, 1989.

[41] O. E. Livne. Diagonal dominance of SPD matrices. Technical report, Stanford University, 2004.

[42] O. E. Livne and G. H. Golub. Scaling by binormalization. *Numer. Alg.*, 35(1):97–120, January 2004.

[43] A. W. Marshall and I. Olkin. Scaling of matrices to achieve specified row and column sums. *Numer. Math.*, 12:83–90, 1968.

[44] L. Mirsky and L. Perfect. The distribution of positive elements in doubly stochastic matrices. *J. London Math. Soc.*, 40:689–698, 1965.

[45] J. L. Morales. A numerical study of limited memory BFGS methods. *Appl. Math. Lett.*, 15:481–487, 2002.

[46] J. J. Moré, B. S. Garbow, and K. E. Hillstrom. User guide for MINPACK-1. Technical report, Argonne National Lab., 1980.

[47] M. E. Muller. A note on a method for generating points uniformly on n-dimensional spheres. *Comm. ACM*, 2(4), 1959.

[48] J. L. Nazareth. The Newton and Cauchy perspectives on computational nonlinear optimization. *SIAM Review*, 36(2):215–225, 1994.

[49] A. Neumaier. Scaling and structural condition numbers. *Lin. Alg Appl.*, 263:157–165, 1997.

[50] J. Nocedal. Updating quasi-Newton matrices with limited storage. *Math. Comput.*, 35:773–782, 1980.

[51] J. Nocedal and S. Wright. *Numerical Optimization*. Springer, 2nd edition, 2006.

[52] D. P. O'Leary. Scaling symmetric positive definite matrices to prescribed row sums. *Lin. Alg. Appl.*, 370:185–191, 2003.

[53] Dianne O'Leary. Translation of MINPACK subroutine cvsrch, 1991. `http://www.cs.umd.edu/~oleary/a607/cvsrch.m`.

[54] M. Olschowka and A. Neumaier. A new pivoting strategy for Gaussian elimination. *Lin. Alg. Appl.*, 240:131–151, 1996.

[55] J. M. Ortega and W. C. Rheinboldt. *Iterative Solution of Nonlinear Equations in Several Variables*. SIAM, 2000.

[56] G. Strang P.-O. Persson. A simple mesh generator in MATLAB. *SIAM Review*, 46(2):329–345, 2004.

[57] B. N. Parlett and T. L. Landis. Methods for scaling to double stochastic form. *Lin. Alg. Appl.*, 48:53–79, 1982.

[58] A. V. Rao, D. A. Benson, C. L. Darby, C. Francolin, M. A. Patterson, I. Sanders, and G. T. Huntington. Algorithm: GPOPS, a MATLAB software for solving multiple-phase optimal control problems using the Gauss pseudospectral method. *ACM Trans. Math. Software*, Accepted June 2009.

[59] H. Robbins and S. Monro. A stochastic approximation method. *Ann. of Math. Stat.*, 22(3):400–407, 1951.

[60] D. Ruiz. A scaling algorithm to equilibrate both rows and columns norms in matrices. Technical Report RT/APO/01/4, ENSEEIHT-IRIT, 2001.

[61] R. Sinkhorn. A relationship between arbitrary positive matrices and doubly stochastic matrices. *Ann. Math. Statist.*, 35:876–879, 1964.

[62] R. Sinkhorn and P. Knopp. Concerning nonnegative matrices and doubly stochastic matrices. *Pacific J. Math.*, 21(2):343–348, 1967.

[63] A. van der Sluis. Condition numbers and equilibration of matrices. *Numer. Math.*, 14:14–23, 1969.

[64] F. Veersé and D. Auroux. Some numerical experiments on scaling and updating L-BFGS diagonal preconditioners. *INRIA Research Report 3858*, 2000.

[65] F. Veersé, D. Auroux, and M. Fisher. Limited-memory BFGS diagonal preconditioners for a data assimilation problem in meteorology. *Optim. Engrg.*, 1(3), 2000.

[66] M. Zhu, J. L. Nazareth, and H. Wolkowicz. The quasi-Cauchy relation and diagonal updating. *SIAM J. Optim.*, 9(4):1192–1204, 1999.