New Insights and Algorithms for Optimal Diagonal Preconditioning *†

Saeed Ghadimi‡ Woosuk L. Jung§ Arnesh Sujanani§ David Torregrosa-Belén¶
Henry Wolkowicz§

Revising as of September 27, 2025, 12:27 Noon

Key words and phrases: κ,ω-condition numbers, diagonal preconditioning, iterative methods, PCG, linear systems, eigenvalues

AMS subject classifications: 15A12, 65F35, 65F08, 65G50, 49J52, 49K10, 90C32

Contents

*Emails resp.: sghadimi@uwaterloo.ca, w2jung@uwaterloo.ca, a3sujana@uwaterloo.ca, david.torregrosa@ua.es, hwolkowicz@uwaterloo.ca

†This report is available at URL: www.math.uwaterloo.ca/~hwolkowi/henry/reports/ABSTRACTS.html

‡Department of Management Science and Engineering, University of Waterloo, ON, Canada

§Department of Combinatorics and Optimization, University of Waterloo, ON, Canada

¶Department of Mathematics, University of Alicante, Alicante, Spain

List of Algorithms

List of Tables

List of Figures

Abstract

Preconditioning (scaling) is essential in many areas of mathematics, and in particular in optimization. In this work, we study the problem of finding an optimal diagonal preconditioner. We focus on minimizing two different notions of condition number: the classical, worst-case type, κ-condition number, and the more averaging motivated ω-condition number. We provide affine based pseudoconvex reformulations of both optimization problems. The advantages of our formulations are that the gradient of the objective is inexpensive to compute and the optimization variable is just an $n \times 1$ vector. We also provide elegant characterizations of the optimality conditions of both problems.

We develop a competitive subgradient method, with convergence guarantees, for κ-optimal diagonal preconditioning that scales much better and is more efficient than existing SDP-based approaches. We also show that the preconditioners found by our subgradient method leads to better PCG performance for solving linear systems than other approaches. Finally, we show the interesting phenomenon that we can apply the ω-optimal preconditioner to the exact κ-optimally diagonally preconditioned matrix A and get consistent, significantly improved computational performance for PCG methods.

1 Introduction

Recently, there has been great interest in studying preconditioning for gradient-based methods with the focus on designing online algorithms. This is due to their effectiveness in improving the efficiency of solving

optimization problems arising in different applications, specifically machine learning, e.g., [8, 9, 16]. All of these works focus on studying the classical κ-condition number of a matrix A (the ratio of largest to smallest singular values). On the other hand, another line of work has focused on studying the ω-condition number (the ratio of the arithmetic and geometric mean of the singular values) [5, 12].

In our work herein, motivated by the recent studies on the κ-condition number, we provide new insights on the preconditioners obtained from minimizing both of these condition numbers. For simplicity, we restrict the comparisons to diagonal and block diagonal preconditioning for linear systems $Ax = b$, with A, large scale sparse, and symmetric positive definite, denoted $A \succ 0$. Such systems arise from many diverse applications including: finite element analysis, sparse regression, Newton type optimization algorithms, and also from e.g., in [9, 10]: (i) the so-called normal equations from interior point methods in solving linear programs and (ii) the Hessians in minimizing logistic regression.

The κ-condition number and ω-condition number for $A \succ 0$ are, respectively, given by ratios of eigenvalues

$$\kappa(A) = \frac{\lambda_{\max}(A)}{\lambda_{\min}(A)}; \quad \omega(A) = \frac{\mathrm{tr}(A)/n}{\det(A)^{1/n}} = \frac{\sum_i \lambda_i(A)/n}{\prod_i (\lambda_i(A))^{1/n}}. \tag{1.1}$$

(For our purposes we extend these definitions in (1.1) using eigenvalues, rather than singular values, to nonsymmetric matrices with real positive eigenvalues.) Indeed, κ and ω can be considered as *worst-case* and *average-case* condition numbers, respectively. One of the advantages of ω over κ is that finding explicit formulae for optimal preconditioners by minimizing ω with special structure is often possible due to the analyticity and simplicity of differentiation of trace and determinant.

1.1 Main Contributions

Our main contributions in this paper consist of both theoretical and numerical aspects. First, we provide an affine based pseudoconvex reformulation of the optimal diagonal preconditioning problems that allows for elegant characterizations of their optimality conditions. There are three advantages of this reformulation: all its stationary points are global minima, its subgradients are inexpensive to compute, and its optimization variable is just an $n \times 1$ vector rather than a $n \times n$ matrix as in semidefinite programming (SDP) [16].

Furthermore, we show that subgradient methods based on our reformulation can effectively converge to a κ-optimal diagonal preconditioner more efficiently in time and accuracy than current SDP-based approaches in the literature. For the ω case we provide and exploit the ability to find explicit formulae for the optimal block diagonal preconditioners. To the best of our knowledge, this is the first time that such a comprehensive characterization of optimality conditions for both condition numbers is provided. Finally, we conduct extensive numerical experiments that compare the efficiency of the κ- and ω-condition numbers.

We now continue with the organization of the paper and more details on the main contributions. In Section 2 we present the three formulations for minimizing κ along with the derivatives and optimality conditions. The three formulations are presented to take advantage of the affine approach and then avoid the positive homogeneity of the problem in order to improve stability of the problem. In particular, we include Theorem 2.7 that presents a characterization of a κ-optimally diagonally preconditioned A that is based on the *largest/smallest* eigenpairs. This leads to an efficient subgradient algorithm and we include convergence results. A characterization for ω is simpler as it corresponds to the classical Jacobi preconditioner, see Corollary 2.18. We include results on extending from diagonal to block diagonal preconditioning in Section 2.4.1. This shows connections to partial Cholesky preconditioning.

Extensive numerical tests are then conducted in Section 3. We display that our subgradient methods are more scalable and efficient in finding the κ-optimal diagonal preconditioner than existing SDP-based approaches in the literature. We also find that the preconditioners found by our methods improve PCG's performance for solving linear systems more substantially than the preconditioners found by other methods. Finally, we also present comparisons for optimal diagonal preconditioning based on the two condition numbers κ, ω. The comparisons are based on improvements for the PCG algorithm applied to solving a positive definite linear system.

1.2 Background; Preliminaries

Given a linear system $Ax = b$ with $A \succ 0$ and matrices M_1, M_2, preconditioning effectively solves the given system by solving the following system for y:

$$H^{-1}AH^{-T}y = H^{-1}b \text{ for } y,\ H = M^{1/2} = (M_1M_2)^{1/2},\ y = H^Tx.$$

There is no matrix-matrix multiplication in preconditioning algorithms as they do not form $H^{-1}AH^{-T}$ explicitly, and only matrix-vector multiplications/divisions are performed. For example, the well known *Jacobi preconditioner* uses the square root of the diagonal elements, $a = \operatorname{diag}(A)^{1/2}$. The PCG *implicitly* uses the matrix vector multiplications:

$$\operatorname{Diag}(a)^{-1}A\operatorname{Diag}(a)^{-1}x = a.\backslash\left(A(x./a)\right),$$

where $.\backslash, ./$ represent the MATLAB notation for elementwise division, see e.g., PCG matlab, [17, Sect. 12]. A discussion and references on the two condition numbers κ, ω is given recently in [12]. The ω-condition number is introduced in [5] as a measure for nearness to the identity I for finding optimal quasi-Newton updates; see also [7, 21, 22]. It is related to the measure $\operatorname{tr} A - \log\det A$ used in the convergence proofs in [2, 3]. Then, Kaporin [13] used ω to derive new conjugate gradient convergence rate estimates and guarantees. More recently [1] presents further relationships and convergence analyses.

For certain structures, in contrast to κ, one can get explicit formulae for ω-optimal preconditioners. Thus minimizing the measure ω is efficient for finding the preconditioner as no numerical optimization problem needs to be solved. Moreover, the classic Jacobi (diagonal) preconditioner is a multiple of the ω-optimal diagonal preconditioner. And the ω-optimal partial Cholesky at the k-th step (adding the k-th column) but with full diagonal is shown to arise from the Cholesky factorization with the optimal diagonal preconditioner added (see Corollary 2.20 for more details). Both these observations highlight the close connections ω-optimal preconditioners have with classical heuristic preconditioners.

1.2.1 Notation

We work in *real* Euclidean vector spaces. We let $\mathbb{S}^n$ denote the space of symmetric matrices with the trace inner product and corresponding Frobenius norm; $\mathbb{S}^n_+, \mathbb{S}^n_{++}$ are the cones of positive semidefinite, and definite, symmetric matrices of order n, respectively. We let $\mathcal{M}^n$ denote the space of square matrices of order n, also equipped with the trace inner product and Frobenius norm. For $B \in \mathcal{M}^n$ with real eigenvalues we let

$$\lambda_i = \lambda_i(B): \quad \lambda_{\max} = \lambda_1 \geqslant \lambda_2 \geqslant \ldots \geqslant \lambda_{\min} = \lambda_n,$$

be the eigenvalues in nonincreasing order, and we define the functions: $\lambda_1(B) = \max_i \lambda_i(B)$; $\lambda_n(B) = \min_i \lambda_i(B)$. For our purposes, and by abuse of notation, we define the κ-condition number for *matrices with real positive eigenvalues*, that are *not* necessarily symmetric, as $\kappa(B) = \frac{\max_i \lambda_i(B)}{\min_i \lambda_i(B)}$.[1] We note that in this case we also have $\omega(B) = \frac{\sum_{i=1}^n \lambda_i(B)/n}{\prod_{i=1}^n \lambda_i(B)^{\frac{1}{n}}}$.

We let I be the identity matrix, e be the vector of ones and e_i be the i-th unit vector, all of appropriate dimension; we often need a basis for $e^\perp$, and we use the matrix $V = \frac{1}{\sqrt{2}}\begin{bmatrix} I \\ -e^T \end{bmatrix}$; $X \bullet Y$ denotes the *Hadamard product* (elementwise product) of two (compatible) matrices. For a vector $d \in \mathbb{R}^n$, $\operatorname{Diag}(d) \in \mathbb{S}^n$ denotes the diagonal matrix formed using d. The adjoint linear transformation for $S \in \mathbb{S}^n$ is denoted $\operatorname{Diag}^*(S) = \operatorname{diag}(S) \in \mathbb{R}^n$. For two compatible functions f and g, we let $f \circ g$ denote the composite function.

The classical Fenchel subdifferential of a convex function h is defined as

$$\partial h(x) := \{v : \langle v, y - x\rangle \leqslant h(y) - h(x), \quad \forall y\}. \tag{1.2}$$

[1] In the literature, the condition number κ of a nonsymmetric matrix differs as it is found using the singular values.

The Clarke subdifferential of a locally Lipschitz, not necessarily convex, function h is defined as

$$\partial_C h(x) = \operatorname{conv}\left\{s : \exists x^k \to x, \nabla h(x^k) \text{ exists, and } \nabla h(x^k) \to s\right\} \tag{1.3}$$

where conv denotes the convex hull of a set. It is well-known that the Fenchel subdifferential and the Clarke subdifferential coincide for convex functions. Further notation is introduced below as needed.

We now recall the well known facts about the largest and smallest eigenvalues of symmetric matrices and include a proof for completeness as it is useful further below.

Lemma 1.1. *The maximum and minimum eigenvalue functions on $\mathbb{S}^n$, $\lambda_{\max}, \lambda_{\min} : \mathbb{S}^n \to \mathbb{R}$, are convex, concave, respectively.*

Proof. Letting $B \in \mathbb{S}^n$ and using the Rayleigh quotient, we get $\lambda_{\max}(B) = \max\{x^T Bx : \|x\| = 1\}$, which is a maximum of linear (so convex) functions in B, and therefore is convex. We get the concavity part similarly from $\lambda_{\min}(B) = \min\{x^T Bx : \|x\| = 1\}$.

■

Now let

$$A \in \mathbb{S}^n_{++}, d \in \mathbb{R}^n_{++}, D = \operatorname{Diag}(d).$$

We note the following useful property that follows by the commutativity property for eigenvalues.

$$\lambda_i(DAD) = \lambda_i(ADD), \forall i, \quad \kappa(DAD) = \kappa(ADD), \quad \omega(DAD) = \omega(ADD). \tag{1.4}$$

2 Optimal Diagonal Preconditioning

As mentioned above, preconditioning (scaling) is important in many areas of mathematics such as optimization and numerical linear algebra (see a survey in e.g., [20]). For the purposes of comparing the two condition numbers, κ, ω, and for simplicity, we restrict to preconditioning for positive definite linear systems $Ax = b$, and to diagonal and block diagonal preconditioning. We now consider the gradients and optimality conditions.

We begin in Section 2.1 and resort to the eigenvalue relation in (1.4), which allows us to work with the affine mapping AD to minimize $\kappa(D^{1/2}AD^{1/2})$. Note that [9] also considers minimizing $\kappa(D^{1/2}AD^{1/2})$, although there they consider a different problem where they aim to find a diagonal preconditioner that is a convex combination of a small basis or list of diagonal preconditioners. Working with this affine mapping allows us to get expressions for derivatives for the composite functions and design very efficient and scalable algorithms using the derivatives. In addition, our composite functions avoid the positive homogeneity in κ, ω thus improving stability.

We continue in Section 2.4 with the optimality conditions for ω and see that the Jacobi preconditioner is a multiple of the ω-optimal preconditioner. We then extend this to block diagonal preconditioning in Section 2.4.1 and see how the partial Cholesky factorization relates.

2.1 Optimality Conditions for κ Diagonal Preconditioning

We denote the quadratic type mapping in the scaling

$$\mathcal{D} : \mathbb{R}^n_{++} \to \mathbb{S}^n, \quad \mathcal{D}(d) = \operatorname{Diag}(d)^{1/2} A \operatorname{Diag}(d)^{1/2}.$$

By abuse of notation, we let the argument determine the function,

$$\kappa(d) = \kappa(\mathcal{D}(d)) = (\kappa \circ \mathcal{D})(d).$$

Similarly,

$$\lambda_{\max}(d) = \lambda_{\max}(\mathcal{D}(d)) = (\lambda_{\max} \circ \mathcal{D})(d), \quad \lambda_{\min}(d) = \lambda_{\min}(\mathcal{D}(d)) = (\lambda_{\min} \circ \mathcal{D})(d).$$

We use the form that is most appropriate/useful depending on the context. In the literature, e.g., [8, 9, 16], κ-optimal diagonal preconditioning refers to solving

$$d^* \in \operatorname{argmin} \left\{ \kappa(\mathcal{D}(d)) = \frac{\lambda_{\max}(\mathcal{D}(d))}{\lambda_{\min}(\mathcal{D}(d))} : d \in \mathbb{R}^n_{++} \right\}. \tag{2.1}$$

Here we restrict to $d \in \mathbb{R}^n_{++}$ as we are taking square roots. However, if we allowed for a diagonal scaling $DAD, D = \operatorname{diag}(d), d_i \neq 0 \, \forall i$, then changing the signs to ensure $d > 0$ does not change either κ or ω, see (1.4).

We show below, that for our purposes, we can use the equivalent *linear in* d transformation

$$\mathring{\mathcal{D}} : \mathbb{R}^n_{++} \to \mathbb{S}^n, \ \mathring{\mathcal{D}}(d) = A \operatorname{Diag}(d). \tag{2.2}$$

Thus we consider the equivalent simplified view of finding a κ-optimal diagonal preconditioner, i.e., we need to solve the fractional pseudoconvex minimization problem

$$\bar{d} \in \operatorname{argmin} \left\{ \kappa(\mathring{\mathcal{D}}(d)) = \frac{\lambda_{\max}(\mathring{\mathcal{D}}(d))}{\lambda_{\min}(\mathring{\mathcal{D}}(d))} : d \in \mathbb{R}^n_{++} \right\} \tag{2.3}$$

With $D = \operatorname{Diag}(d)$, we first note the relationships in the eigenpairs of $\mathcal{D}(d) = D^{1/2}AD^{1/2}$, $\mathring{\mathcal{D}}(d) = AD$, and $\mathring{\mathcal{D}}(d)^T = DA$, as well as the convexity and concavity of the maximum and minimum eigenvalues of these nonsymmetric matrices AD.[2]

Lemma 2.1. *Let $A, D = \operatorname{Diag}(d) \in \mathbb{S}^n_{++}$, and $(u_i, \lambda_i), i = 1, \ldots, n$, be orthogonal eigenpairs of $\mathcal{D}(d)$. Define*

$$x_i = D^{-1/2}u_i, \forall i, \ X = \begin{bmatrix} x_1 & \ldots & x_n \end{bmatrix}, Y = X^{-T}, \ \Lambda = \operatorname{Diag}(\lambda).$$

Then X and Y are matrices of right and left eigenvectors of $\mathring{\mathcal{D}}(d)$, respectively, with corresponding matrix of eigenvalues Λ; and Y is given by $Y = DX(X^TDX)^{-1}$. Moreover, $\lambda_{\max}(d) = \lambda_1(d)$ (respectively, $\lambda_{\min}(d) = \lambda_n(d)$) is a convex (respectively, concave) function of $d \in \mathbb{R}^n_{++}$.

Proof. Let $U := [u_1 \ \ldots \ u_n]$. We have $X = D^{-1/2}U$ and thus $U^TU = X^TDX$. This brings us to

$$U^{-T} = U(X^TDX)^{-1}.$$

Notice that X^TDX is a diagonal matrix with its diagonal entries correspond to $\|u_i\|^2$, which are strictly positive as eigenvectors are nonzero. Thus the inverse is well-defined. Now, observe that

$$\begin{array}{rcl} Y & = & X^{-T} = D^{1/2}U^{-T} \\ & = & D^{1/2}U(X^TDX)^{-1} \\ & = & DX(X^TDX)^{-1} \end{array}$$

as desired. Now,

$$\begin{array}{rcl} D^{1/2}AD^{1/2}U = U\Lambda & \implies & ADD^{-1/2}U = D^{-1/2}U\Lambda \\ & \implies & ADX = X\Lambda \\ & \implies & X^TDA = \Lambda X^T \\ & \implies & DAX^{-T} = X^{-T}\Lambda \\ & \implies & DAY = Y\Lambda. \end{array}$$

[2] These are a special case of so-called K-pd matrices in the literature, i.e., a product of two symmetric matrices where at least one is positive definite. The product of two positive definite matrices $P = AB$ arises in the optimality conditions in semidefinite programming. One of the difficulties is that the symmetrization of P is *not* necessarily positive definite, see e.g., [18].

The second and last equation show that X and Y are right, and left, eigenvectors of AD, respectively. Moreover, we note that $A \succ 0$ has a positive definite square root and $\lambda_i(A \operatorname{Diag}(d)) = \lambda_i(A^{1/2} \operatorname{Diag}(d) A^{1/2})$. Therefore,

$$\max \lambda_i(\mathring{\mathcal{D}}(d)) = \max \lambda_i(A^{1/2} \operatorname{Diag}(d) A^{1/2}) = \max_{\|x\|=1} x^T A^{1/2} \operatorname{Diag}(d) A^{1/2} x.$$

As in the proof of Lemma 1.1, the latter is a maximum of linear functions in d and therefore is convex. The concavity result follows similarly.

■

Corollary 2.2. *Let $A \in \mathbb{S}^n_{++}, d \in \mathbb{R}^n_{++}$. Then*

$$\lambda_{\max}(d) = \lambda_{\max}(\mathcal{D}(d)) = \lambda_{\max}(\mathring{\mathcal{D}}(d)), \quad \lambda_{\min}(d) = \lambda_{\min}(\mathcal{D}(d)) = \lambda_{\min}(\mathring{\mathcal{D}}(d));$$

and

$$\kappa(d) = \kappa(\mathcal{D}(d)) = \kappa(\mathring{\mathcal{D}}(d)).$$

■

The next result shows that for $A \succ 0$, the matrix AD having all positive eigenvalues is equivalent to positive definiteness of D. This proves useful in the linesearch parts of our algorithms below.

Lemma 2.3. *Let $A \in \mathbb{S}^n_{++}, d \in \mathbb{R}^n, D = \operatorname{Diag}(d)$. Then*

$$\lambda_i(AD) > 0, \forall i \iff d \in \mathbb{R}^n_{++}.$$

Proof. We note that the eigenvalues of AD are the same as the eigenvalues of $A^{1/2} D A^{1/2}$ and Sylvester's Lemma of inertia implies that the number of negative eigenvalues of $A^{1/2} D A^{1/2}$ is the same as that of D so the same as the number of negative elements in d.

■

Remark 2.4. *Note that the two equivalent problems (2.1) and (2.3) are both essentially unconstrained, and the optima are attained and characterized as stationary points in $\mathbb{R}^n_{++}$. This is not obvious but follows since we have the ratios of convex and concave functions using $\lambda_{\max}, \lambda_{\min}$ and this is over the open cone constraint $d \in \mathbb{R}^n_{++}$. If we add the constraints $d^T e = n, d > 0$, or equivalently that $d = e + Vv > 0$, with $V \in \mathbb{R}^{n \times (n-1)}$ a matrix whose columns contain a basis for $e^\perp$ as done above, then we have a bounded problem in $v \in \mathbb{R}^{n-1}$ and $\lambda_{\max}$ is bounded. Bounded below away from 0 follows from applying the greedy solution to the knapsack problem with constraints $\sum_i d_i = n, d \geqslant 0$. We get*

$$\lambda_{\max}(d) \quad \geqslant \quad \operatorname{tr}(AD)/n = \textstyle\sum_i A_{ii} d_i / n \geqslant \min_i A_{ii} > 0.$$

Therefore, with the denominator going to 0, we have κ going to ∞ as $d = e + Vv$ approaches the boundary. That is, minimizing κ provides a self-barrier function *for the boundary.*

Moreover, $\alpha > 0 \implies \kappa(d) = \kappa(\alpha d)$, i.e., we have positive homogeneity. Therefore, the opimal d is not *unique. By pseudoconvexity, the optimal set is a convex set. This set can be large, see Proposition 2.8.*

We now provide the derivative information that we need for the optimality conditions.

Lemma 2.5 ([19, (3.1)]). *Let $B : \mathbb{R} \to \mathcal{M}^n$ be differentiable with derivative $\dot{B} = \dot{B}(t)$, and, at $t = \bar{t}$, let $B(\bar{t})$ have real eigenpairs and be diagonalizable with eigenpairs (ignoring the argument $\bar{t}$)*

$$BX = X\Lambda,\ B^T Y = Y\Lambda, \quad \Lambda = \operatorname{Diag}(\lambda).$$

And make the choice $Y = X^{-T}$. Let $1 \leqslant k \leqslant n$ and λ_k be a singleton eigenvalue with right and left eigenvector x_k, y_k, respectively, taken from the corresponding columns of X, Y, respectively. Then the derivative of eigenvalues is given by

$$\dot{\lambda}_k(\bar{t}) = y_k^T(\bar{t})\dot{B}(\bar{t})x_k(\bar{t}) = \operatorname{tr} \dot{B} x_k y_k^T.^{3} \tag{2.4}$$

Proof. The proof is in [19, Pg 303].[4]

■

Lemma 2.6. *The Fréchet derivative of the linear transformation $\mathring{\mathcal{D}}$ at d acting on Δd is (simply)*

$$\mathring{\mathcal{D}}'(d)(\Delta d) = A \operatorname{Diag}(\Delta d).$$

Now let λ be a singleton eigenvalue of $\mathring{\mathcal{D}}(d)$ with a right eigenvector x. Then the gradient of the composite function at d is

$$\nabla\lambda(\mathring{\mathcal{D}}(d)) = \frac{\lambda}{x^T D x} x \bullet x.$$

Proof. The derivative of the linear transformation is clear.

Suppose as above that we have a linear transformation $\mathring{\mathcal{D}}(d)$ with singleton eigenvalue λ. Let $y = Dx/x^T Dx$ be the corresponding left eigenvector given in Lemma 2.1. Then the derivative of the composite function at d acting on Δd is

$$\begin{aligned}
\langle \nabla(\lambda \circ \mathring{\mathcal{D}})(d), \Delta d\rangle &= \lambda'\left(\mathring{\mathcal{D}}'(d)(\Delta d)\right) = y^T\left(\mathring{\mathcal{D}}'(d)(\Delta d)\right)x \\
&= \tfrac{1}{x^T Dx} x^T \left(DA \operatorname{Diag}(\Delta d)\right) x \\
&= \tfrac{\lambda}{x^T Dx} x^T \operatorname{Diag}(\Delta d) x \\
&= \tfrac{\lambda}{x^T Dx} \operatorname{tr} x x^T \operatorname{Diag}(\Delta d) \\
&= \tfrac{\lambda}{x^T Dx} \langle \operatorname{diag}(x x^T), \Delta d\rangle \\
&= \tfrac{\lambda}{x^T Dx} \langle x \bullet x, \Delta d\rangle.
\end{aligned}$$

■

Theorem 2.7. *Let $A \in \mathbb{S}^n_{++}, d \in \mathbb{R}^n_{++}$ be given; let $D = \operatorname{Diag}(d)$. Then the gradient of the composite function $\kappa(d) := \kappa(\mathring{\mathcal{D}}(d))$ is*

$$\nabla\kappa(d) = \kappa(d)\left(\frac{1}{x_1^T D x_n}(x_1 \bullet x_1) - \frac{1}{x_n^T D x_n}(x_n \bullet x_n)\right).$$

Hence, A is κ-optimally diagonally preconditioned if, and only if, the orthonormal eigenvector pair satisfies

$$x_1 \bullet x_1 = x_n \bullet x_n. \tag{2.5}$$

Equivalently, after a permutation of the elements to account for the sign,

$$x_1 = \begin{pmatrix} u \\ v \end{pmatrix},\ x_n = \begin{pmatrix} u \\ -v \end{pmatrix},\ \|u\| = \|v\|. \tag{2.6}$$

In the nonsmooth case, we can choose the normalized x_1, respectively, x_n, in the eigenspace of $\lambda_{\max}(\mathring{\mathcal{D}}(d))$, respectively $\lambda_{\min}(\mathring{\mathcal{D}}(d))$.

[3] If Y is an invertible matrix of right-eigenvectors not chosen using X^{-T}, then the normalization scaling needs to be added explicitly as $\dot{\lambda}_k(\bar{t}) = y_k^T(\bar{t})\dot{B}(\bar{t})x_k(\bar{t})/(y_k(\bar{t})^T x(\bar{t}))$.

[4] The derivative of eigenvectors is included as is a normalization for stability of evaluation for eigenvectors is included.

Proof. From Lemma 2.6, we can find the gradient as follows:

$$\begin{aligned}\nabla\kappa(d) &= \frac{1}{\lambda_n(d)^2}\left(\lambda_n(d)\dot{\lambda}_1(d) - \lambda_1(d)\dot{\lambda}_n(d)\right) = \frac{1}{\lambda_n^2}\left(\frac{\lambda_n\lambda_1}{x_1^T D x_1}x_1 \bullet x_1 - \frac{\lambda_1\lambda_n}{x_n^T D x_n}x_n \bullet x_n\right)\\ &= \kappa(d)\left((x_1 \bullet x_1)/(x_1^T D x_1) - (x_n \bullet x_n)/(x_n^T D x_n)\right).\end{aligned}$$

The characterization for κ-optimally diagonally preconditioned matrix A follows from solving $\nabla\kappa(e) = 0$.

■

We now use Theorem 2.7 to illustrate that the optimal set can be a large set. A sufficient condition for nonuniqueness is provided in Proposition 2.8.

Proposition 2.8 (Nonuniqueness of κ-optimal diagonal scaling)**.** *Let $A > 0$ be κ-optimal diagonal preconditioned as given in Theorem 2.7 with $x_i, \lambda_i, i = 1, n$, being two, singleton, eigenpairs that satisfy the optimality conditions in Theorem 2.7. Thus we have*

$$\lambda_1 > \lambda_2 \geqslant \ldots \geqslant \lambda_{n-1} > \lambda_n > 0, \quad \lambda = (\lambda_i) \in \mathbb{R}^n_{++},\ \Lambda = \mathrm{Diag}(\lambda),$$

and we let $Q = \begin{bmatrix} x_1 & \bar{Q} & x_n \end{bmatrix}$ be an orthogonal matrix and $A = Q\Lambda Q^T$ from the spectral theorem. Suppose in addition that the lack *of strict complementarity condition holds, i.e., there exists v such that*

$$0 \neq v \in \{u \in \mathbb{R}^{n-1} : 0 = Vu \bullet x_1 = Vu \bullet x_n\}. \tag{2.7}$$

Then with $\Delta D = \mathrm{Diag}(Vv)$, there exists $\epsilon > 0$ such that

$$\kappa(A) = \kappa\left((I + t\Delta D)A(I + t\Delta D)\right),\ \forall |t| \leqslant \epsilon.$$

Proof. The result follows from expanding

$$(I + \epsilon\Delta D)A(I + \epsilon\Delta D) = A + O(\epsilon)$$

and using the continuity of eigenvalues and the fact that the optimality conditions imply

$$\begin{aligned} Vv \bullet x_i = 0, i = 1, n \quad &\Longleftrightarrow \quad \mathrm{Diag}(Vv)x_i = 0, i = 1, n\\ &\Longleftrightarrow \quad \Delta D x_i = 0, i = 1, n.\end{aligned}$$

Specifically, let

$$A_2 = \bar{Q}\,\mathrm{Diag}((\lambda_2, \ldots, \lambda_{n-1})^T)\bar{Q}^T, \quad A_1 = A - A_2.$$

The above equation then implies $\Delta D A_1 = A_1 \Delta D = 0$. Moreover, let $D = (I + \epsilon\Delta D)$, then

$$DAD = A_1 + DA_2D = A + \epsilon\Delta D A_2 + \epsilon A_2 \Delta D + \epsilon^2 \Delta D A_2 \Delta D.$$

Since $\mathrm{range}(\Delta D) \subseteq \mathrm{null}(A_1) = \mathrm{range}(A_2)$, the range of the perturbation of A above is restricted to the eigenspace of A_2, i.e., to the $\mathrm{span}(\{x_2, x_3, \ldots, x_{n-1}\})$. This means that after the perturbation, with $\epsilon > 0$ sufficiently small, λ_1 and λ_n remain the largest and smallest eigenvalues of DAD respectively. As stated above, we are using the continuity of eigenvalues and the orthogonality $A_1 A_2 = 0$ that arises using the spectral theorem and $A_1, A_2 \succeq 0$.

■

Note that if condition (2.7) holds, then we get a nonsingleton set in $\mathbb{R}^{n-1}$ of solutions. Moreover, the structure of V implies that (2.7) restricts the support of the eigenspace $\mathrm{span}(\{x_1, x_n\})$.[5]

[5]Necessity is still an open problem.

2.2 A Projected Subgradient Method for Minimizing $\kappa(\mathring{\mathcal{D}}(d))$

In Theorem 2.7, we derived the gradient and optimality conditions for minimizing the pseudoconvex function $\kappa(\mathring{\mathcal{D}}(d))$ in (2.3), over the open set $d > 0$, where $\mathring{\mathcal{D}}(d) = AD$. Convergence of subgradient methods for pseudoconvex minimization typically requires optimizing over a closed set. Hence, we consider the following optimization problem:

$$\bar{d} \in \operatorname{argmin} \left\{ \kappa(\mathring{\mathcal{D}}(d)) = \frac{\lambda_{\max}(\mathring{\mathcal{D}}(d))}{\lambda_{\min}(\mathring{\mathcal{D}}(d))} : d \in \Omega := \{d : d \geqslant \delta e\} \right\}, \tag{2.8}$$

where in this subsection we define $\kappa(\mathring{\mathcal{D}}(d)) := \infty$ for $d \notin \Omega$ and let $\delta \in (0, 1)$ be a small positive scalar. Clearly then the domain of $\kappa(\mathring{\mathcal{D}}(d))$ is contained in Ω. Also, it is easy to see that Ω is a closed set since it is just a polyhedron. We now present in Algorithm 2.1 our subgradient method for minimizing $\kappa(\mathring{\mathcal{D}}(d))$ over Ω.

Algorithm 2.1 A Subgradient Method for Minimizing $\kappa(\mathring{\mathcal{D}}(d))$ over Ω

Inputs: symmetric positive definite matrix $A > 0$; sequence of positive stepsizes $\{t_k\} \to 0$ with $\sum_{k=1}^{\infty} t_k = \infty$, scalar $\delta \in (0, 1)$; tolerance tol; rule for the stopping criterion, stopcrit.

set $k \leftarrow 0$;

set stopcrit $\leftarrow \infty$;

set $d_1 = e \in \mathbb{R}^n$;

compute minimum eigenpair (λ_n^1, x_n^1) and maximum eigenpair (λ_1^1, x_1^1) of A;

1: **while** stopcrit>tol **do** (main outer loop)

2: set $k \leftarrow k + 1$;

3: compute direction

$$s_k = \frac{\lambda_1^k}{\lambda_n^k} \left(\frac{1}{\langle x_1^k, d_k \bullet x_1^k \rangle} (x_1^k \bullet x_1^k) - \frac{1}{\langle x_n^k, d_k \bullet x_n^k \rangle} (x_n^k \bullet x_n^k) \right); \tag{2.9}$$

4: perform projected gradient step

$$d_{k+1} = \max \left\{ d_k - t_k \frac{s_k}{\|s_k\|}, \delta e \right\}; \tag{2.10}$$

5: compute min eigenpair $(\lambda_n^{k+1}, x_n^{k+1})$ and max eigenpair $(\lambda_1^{k+1}, x_1^{k+1})$ of $A \operatorname{Diag}(d_{k+1})$;

6: update stopcrit;

7: **end while**(main outer loop)

Output: $\hat{D} := \operatorname{Diag}(d_{k+1})$.

Several remarks about Algorithm 2.1 are now given. First, Algorithm 2.1 takes as input a positive definite matrix $A \in \mathbb{S}_{++}^n$, a non-summable sequence of positive stepsizes $\{t_k\}$ converging to 0, and a rule for the stopping criterion, stopcrit. A popular choice for stepsize sequence $\{t_k\}$ in subgradient methods is $t_k = 1/k$ where k is the iteration index. Second, since Algorithm 2.1 is a subgradient method, in general it is not a descent method. However, if Algorithm 2.1 descends at every iteration then the output $\hat{D}$ satisfies $\kappa(A\hat{D}) \leqslant \kappa(A)$. Finally, the algorithm is general in that it does not specify the rule for how stopcrit, which is used to stop the method, is updated in step 6 of the algorithm. There are several possible rules that the user can use in practice for updating stopcrit in step 6. For example, at every iteration the user can take stopcrit= $\|s_k\|$ or they can use stopcrit= $|\kappa(\mathring{\mathcal{D}}(d_{k+1})) - \kappa(\mathring{\mathcal{D}}(d_k))|$.

We now briefly describe each of the steps of Algorithm 2.1. First, it will be shown in Section 2.2.1 that the direction s_k computed in step 3 lies in the quasisubdifferential (also to be defined in Section 2.2.1) of $\kappa(\mathring{\mathcal{D}}(d_k))$. Using this direction, step 4 performs the projected gradient update $d_{k+1} = \Pi_\Omega(d_k - t_k \frac{s_k}{\|s_k\|})$, where $\Pi_\Omega(x) := \operatorname{argmin}_\Omega \|x - \cdot\|$ denotes the orthogonal projection onto Ω. Finally, step 5 computes a maximal and

minimal eigenpair of $A\,\mathrm{Diag}(d_{k+1})$ to construct the next search direction s_{k+1}. It should be noted that the user never has to form the matrix $A\,\mathrm{Diag}(d_{k+1})$ to compute a minimal and maximal eigenpair of it. Instead, the user only has to construct subroutines which compute $A\,\mathrm{Diag}(d_{k+1}) * x$ and $(A\,\mathrm{Diag}(d_{k+1}))\backslash x$ efficiently, where $x \in \mathbb{R}^n$.

2.2.1 Asymptotic Convergence Analysis

Our convergence analysis of Algorithm 2.1 relies mostly on [14] where efficient subgradient methods for minimizing quasiconvex functions are presented. Under various assumptions in addition to quasiconvexity, the author establishes asymptotic convergence of subgradient methods with decaying stepsizes. Since our function $\kappa(\mathring{\mathcal{D}}(d))$ is pseudoconvex, it is quasiconvex. For a more detailed discussion related to the assumptions of [14] and how our set-up satisfies these assumptions, the reader should refer to Appendix A. In the remaining part of this subsection, we show that Algorithm 2.1 asympototically converges by showing that it is an instance of the subgradient framework proposed by Kiwiel.

First, we need to define a special subdifferential called the quasisubdifferential for a quasiconvex function f. Define the strict sublevel set or inner slice of f as:

$$\bar{S}(x) := \big\{y \in \operatorname{int} \bar{D} : f(y) < f(x)\big\} \tag{2.11}$$

where $\operatorname{int} \bar{D}$ denotes the interior of the domain of f. The quasisubdifferential of a quasiconvex function f relative to the above sublevel set is defined as

$$\bar{\partial}^\circ f(x) := \big\{g : \langle g, y - x\rangle < 0, \quad \forall y \in \bar{S}(x)\big\}. \tag{2.12}$$

To minimize a quasiconvex function f over a closed convex set X, Kiwiel proposes in [14] the following basic subgradient algorithm:

$$x_{k+1} := \Pi_X(x_k - t_k \hat{g}_k), \quad \hat{g}_k := g_k/\|g_k\|, \quad g_k \in \bar{\partial}^\circ f(x_k), \quad k = 1, 2, \ldots, \quad x_1 \in X,$$

where $t_k > 0$ are the stepsizes.

Hence, we need to characterize vectors that lie in the quasisubdifferential of $\kappa(\mathring{\mathcal{D}}(d))$ to show that Algorithm 2.1 is an instance of Kiwiel's subgradient framework. The result below presents one characterization of vectors that lie in the quasisubdifferential of fractional programs like the one in our set-up (2.8).

Lemma 2.9. *Suppose $f(x) := a(x)/b(x)$ for all $x \in X$ and $f(x) := \infty$ for $x \notin X$, where $a(x)$ is a convex function that is positive on X, $b(x)$ is a concave function that is positive on X. Let $\bar{D}^b$ denote the domain of b. If for $x \in X \cap \bar{D}^b$, $B := 1/b(x)$ and $A := a(x)/b^2(x)$, then*

$$B[\partial a(x)] + A[\partial(-b)(x)] \subseteq \bar{\partial}^\circ f(x).$$

Proof. Let $x \in X \cap \bar{D}^b$ and consider B and A as in the assumptions of the lemma. It follows from the fact that a and $-b$ are convex functions, B and A are positive scalars, and Fenchel subdifferential calculus rules that

$$B[\partial a(x)] + A[\partial(-b)(x)] = \partial(Ba)(x) + \partial(-Ab)(x) \subseteq \partial(Ba - Ab)(x).$$

Now, let $g \in B[\partial a(x)] + A[\partial(-b)(x)] \subseteq \partial(Ba - Ab)(x)$, and suppose $y \in \bar{S}(x)$, i.e., y is in the interior of the domain of f and $f(y) < f(x)$. It then follows that

$$\begin{aligned}\langle g, y - x\rangle &\leqslant Ba(y) - Ab(y) - Ba(x) + Ab(x)\\ &= \frac{a(y)}{b(x)} - \frac{a(x)b(y)}{b^2(x)} < 0,\end{aligned}$$

where the first inequality follows from the definition of Fenchel subdifferential, the equality follows from the definitions of A and B, and the last inequality follows from the fact that $a(y)/b(y) < a(x)/b(x)$ and $b(y)$ and $b(x)$ are positive. It then follows from the definition of quasisubdifferential in (2.12) that $g \in \bar{\partial}^\circ f(x)$.

The following corollary constructs a vector that lies in the quasisubdifferential of $\kappa(\mathring{\mathcal{D}}(d))$.

Corollary 2.10. *Consider $A \succ 0$ as in (2.8) and $d \in \Omega \cap \operatorname{int} \bar{D}^{\lambda_{\min}}$, where $\bar{D}^{\lambda_{\min}}$ is the domain of $\lambda_{\min}(\mathring{\mathcal{D}}(d))$. Let (λ_1, x_1) and (λ_n, x_n) be a maximal and minimal eigenpair of $\mathring{\mathcal{D}}(d)$, respectively. Then, it holds that*

$$\frac{\lambda_1}{\lambda_n}\left(\frac{1}{x_1^T(d \bullet x_1)}(x_1 \bullet x_1) - \frac{1}{x_n^T(d \bullet x_n)}(x_n \bullet x_n)\right) \in \bar{\partial}^\circ\left(\kappa(\mathring{\mathcal{D}}(d))\right).$$

Proof. It follows from Lemma 2.1 that $\lambda_{\max}(\mathring{\mathcal{D}}(d))$ and $\lambda_{\min}(\mathring{\mathcal{D}}(d))$ are convex and concave functions of d, respectively. Also, it is easy to see that $\lambda_{\max}(\mathring{\mathcal{D}}(d))$ and $\lambda_{\min}(\mathring{\mathcal{D}}(d))$ are positive on Ω. Hence, it follows from Lemma 2.9 that

$$\frac{1}{\lambda_{\min}(\mathring{\mathcal{D}}(d))}\partial\lambda_{\max}(\mathring{\mathcal{D}}(d)) + \frac{\lambda_{\max}(\mathring{\mathcal{D}}(d))}{\lambda^2_{\min}(\mathring{\mathcal{D}}(d))}\partial\left(-\lambda_{\min}(\mathring{\mathcal{D}}(d))\right) \subseteq \bar{\partial}^\circ\left(\kappa(\mathring{\mathcal{D}}(d))\right) \tag{2.13}$$

holds for $d \in \Omega \cap \operatorname{int} \bar{D}^{\lambda_{\min}}$. Moreover, it follows from Lemma 2.6, the definition of Clarke subdifferential in (1.3), and the fact that the Fenchel and Clarke subdifferentials coincide for convex functions that the following inclusions hold.

$$\frac{\lambda_1}{x_1^T(d \bullet x_1)}(x_1 \bullet x_1) \in \partial\lambda_{\max}(\mathring{\mathcal{D}}(d)), \quad -\frac{\lambda_n}{x_n^T(d \bullet x_n)}(x_n \bullet x_n) \in \partial\left(-\lambda_{\min}(\mathring{\mathcal{D}}(d))\right), \tag{2.14}$$

where here (λ_1, x_1) and (λ_n, x_n) are maximal and minimal eigenpairs of $\mathring{\mathcal{D}}(d)$, respectively. The result then follows from (2.13) and (2.14).[6]

Remark 2.11. *It follows from Corollary 2.10 that the update rule (2.10) in step 4 is of the form*

$$d_{k+1} = \Pi_\Omega\left(d_k - t_k\frac{s_k}{\|s_k\|}\right), \text{ where } s_k \in \bar{\partial}^\circ\left(\kappa(\mathring{\mathcal{D}}(d_k))\right).$$

With this lemma now in hand, we are now ready to present the main theorem which shows that Algorithm 2.1 asymptotically converges.

Theorem 2.12. *Let $\kappa_* := \min\left\{\kappa(\mathring{\mathcal{D}}(d)) : d \in \Omega\right\}$ and let $\kappa_*^j = \min_{k=1}^j \kappa(\mathring{\mathcal{D}}(d_k))$. It then holds that $\underline{\lim}_{k\to\infty}\kappa(\mathring{\mathcal{D}}(d_k)) = \kappa_*$; in particular, $\kappa(\mathring{\mathcal{D}}(d_k)) \downarrow \kappa_*$.*

Proof. It follows from last remark in Lemma 2.1 and the definitions of $\kappa(\mathring{\mathcal{D}}(d))$ and Ω in (2.8) that $\lambda_{\max}(\mathring{\mathcal{D}}(d))$ (resp. $\lambda_{\min}(\mathring{\mathcal{D}}(d))$) is a convex (resp. concave) function that is positive on Ω. It then follows from this observation, Lemma A.2, and the definition of $\kappa(\mathring{\mathcal{D}}(d))$ that assumptions **A1-A4** in Appendix A hold for the minimization problem in (2.8). Clearly, also assumption **A5** in Appendix A also holds since Ω in (2.8) is a closed set and the intersection of Ω with the interior of the domain of $\kappa(\mathring{\mathcal{D}}(d))$ is clearly nonempty. The result of the theorem then immediately follows from this observation, Remark 2.11, the facts that $t_k \to 0$, and $\sum_{k=1}^\infty t_k = \infty$, and Theorem 1 in [14].

[6] The detailed proof for the inclusions (2.14) are given in Appendix B.

2.3 Avoiding Positive Homogeneity in Minimizing $\kappa(\mathring{\mathcal{D}}(d))$

As mentioned in Remark 2.4, $\kappa(\mathring{\mathcal{D}}(d))$ is a positively homogenous function. Thus there are multiple optimal solutions and our problem is *Hadamard ill-posed*. From a computational stability perspective, it is more efficient to minimize an equivalent smaller dimensional formulation that is not positively homogeneous. With this in mind, we let $A \succ 0$ and $V \in \mathbb{R}^{n\times(n-1)}$ be a matrix whose columns form a basis for $e^{\perp}$; and we consider the function

$$\mathring{\mathcal{V}}(v) := A\,\mathrm{Diag}(e + Vv). \tag{2.15}$$

In this subsection, we consider the following formulation

$$\min\{\kappa(v) := \kappa(\mathring{\mathcal{V}}(v)) : e + Vv \geqslant \hat{\delta}e,\ v \in \mathbb{R}^{n-1}\}, \tag{2.16}$$

where $\kappa(v) := \infty$ if $e + Vv < \hat{\delta}e$ and $\hat{\delta} \in (0, 1)$ is a scalar. It is easy to see that with the choice of $V = \frac{1}{\sqrt{2}}\begin{bmatrix} I \\ -e^T \end{bmatrix}$, (2.16) can be rewritten as

$$\min\left\{\kappa(v) : v \in \hat{\Omega}\right\}, \tag{2.17}$$

where

$$\hat{\Omega} := \left\{ v \in \mathbb{R}^{n-1} : \sum_{i=1}^{n-1} v_i \leqslant -\sqrt{2}(\hat{\delta} - 1),\quad v_i \geqslant \sqrt{2}(\hat{\delta} - 1),\quad i = 1, \dots n-1 \right\}. \tag{2.18}$$

Note that the set $\hat{\Omega}$ is convex and compact. Projecting onto it is also easy since it is just a simplex. Also, since $\hat{\Omega}$ is bounded, we will be able to get nonasymptotic convergence guarantees for the subgradient method that we propose to minimize (2.17).

The following lemma will be useful for developing our subgradient method. It gives useful characterizations of the derivatives of $\mathring{\mathcal{V}}(v)$ and $\kappa(v)$ in the smooth setting when the maximum and minimum eigenvalues of $\mathring{\mathcal{V}}(v)$ have multiplicity one.

Lemma 2.13 (Derivatives of $\mathring{\mathcal{V}}(v), \kappa(v)$)**.** *Let $A \succ 0, v \in \mathbb{R}^{n-1}$ be given and set $w = e + Vv \in \mathbb{R}^n_{++}$ and $D = \mathrm{Diag}(w)$. Also, let (λ_1, x_1) and (λ_n, x_n) be maximal and minimal eigenpairs of $\mathring{\mathcal{V}}(v)$, respectively, where λ_1 and λ_n have multiplicity one and x_1 and x_n are assumed to be normalized. Then the following holds:*

1 *The derivative at v acting on $\Delta v \in \mathbb{R}^{n-1}$ is*

$$\dot{\mathring{\mathcal{V}}}(v)(\Delta v) := \mathring{\mathcal{V}}'(v)(\Delta v) = A\,\mathrm{Diag}(V\Delta v).$$

2 *The gradient of the composite function $\kappa(v) := \kappa(\mathring{\mathcal{V}}(v))$ is*

$$\nabla\kappa(v) = \kappa(v)V^T\left(\frac{1}{x_1^T(w \bullet x_1)}(x_1 \bullet x_1) - \frac{1}{x_n^T(w \bullet x_n)}(x_n \bullet x_n)\right). \tag{2.19}$$

Proof. 1 The proof follows from the function being affine.

2 For a singleton eigenvalue λ with normalized right eigenvector x, we have the left eigenvector $y = Dx/(x^TDx)$ as in Lemma 2.1, which satisfies $x^Ty = 1$. Then

$$\begin{aligned}
\langle \nabla(\lambda \circ \mathring{\mathcal{V}})(v), \Delta v\rangle &= y^T\dot{\mathring{\mathcal{V}}}(v)(\Delta v)x = y^T(A\,\mathrm{Diag}(V\Delta v))x \\
&= \mathrm{tr}\, xy^T(A\,\mathrm{Diag}(V\Delta v)) \\
&= \langle Ayx^T, (\mathrm{Diag}(V\Delta v))\rangle \\
&= \langle \mathrm{diag}(Ayx^T), V\Delta v\rangle \\
&= \langle V^T\,\mathrm{diag}(Ayx^T), \Delta v\rangle \\
&= \langle V^T\,\mathrm{diag}(\lambda D^{-1}yx^T), \Delta v\rangle \\
&= \tfrac{1}{x^TDx}\langle V^T\,\mathrm{diag}(\lambda D^{-1}Dxx^T), \Delta v\rangle \\
&= \tfrac{1}{x^T(w\bullet x)}\langle V^T\,\mathrm{diag}(\lambda xx^T), \Delta v\rangle \\
&= \tfrac{1}{x^T(w\bullet x)}\langle \lambda V^T(x \bullet x), \Delta v\rangle.
\end{aligned}$$

Therefore the gradient of $\kappa(v) := \kappa(\mathring{\mathcal{V}}(v))$ is:

$$\begin{aligned} \nabla\kappa(v) &= \frac{\lambda_n\dot{\lambda}_1 - \lambda_1\dot{\lambda}_n}{\lambda_n^2} = \frac{1}{\lambda_n^2}\left(\frac{\lambda_n}{x_1^T(w\bullet x_1)}\lambda_1 V^T(x_1\bullet x_1) - \frac{\lambda_1}{x_n^T(w\bullet x_n)}\lambda_n V^T(x_n\bullet x_n)\right) \\ &= \kappa(v)V^T\left(\frac{1}{x_1^T(w\bullet x_1)}(x_1\bullet x_1) - \frac{1}{x_n^T(w\bullet x_n)}(x_n\bullet x_n)\right). \end{aligned} \tag{2.20}$$

■

The following Remark 2.14 shows that, as in Lemma 2.13, $\nabla\kappa(v)$ lies in the quasisubdifferential of $\kappa(v)$.

Remark 2.14. *Let $A \succ 0$ and $v \in \mathbb{R}^{n-1}$. Also, suppose that $w = e + Vv$ and let (x_i, λ_i), $i = 1, n$, be eigenpairs of $\mathring{\mathcal{V}}(v)$, where $\|x_i\| = 1$. It then follows from Lemma 2.9 and a similar argument as in the proof of Corollary 2.10 that*

$$\kappa(v)V^T\left(\frac{1}{x_1^T(w\bullet x_1)}(x_1\bullet x_1) - \frac{1}{x_n^T(w\bullet x_n)}(x_n\bullet x_n)\right) \in \bar{\partial}^\circ\left(\kappa(v)\right) \tag{2.21}$$

where recall that $\kappa(v) := \kappa(\mathring{\mathcal{V}}(v))$.

We now present our subgradient algorithm for minimizing (2.17).

Algorithm 2.2 A Subgradient Method for Minimizing $\kappa(v) := \kappa(\mathring{\mathcal{V}}(v))$ over $\hat{\Omega}$

Inputs: symmetric positive definite matrix $A \succ 0$; $V \in \mathbb{R}^{n\times(n-1)}$ a basis matrix for the orthogonal complement $e^\perp$; sequence of stepsizes $\{t_k\} = 1/\sqrt{k}$; scalar $\hat{\delta} \in (0,1)$; tolerance tol; a rule for the stopping criterion, stopcrit.

set $k \leftarrow 0$;

set stopcrit $\leftarrow \infty$;

set $v_1 = 0 \in \mathbb{R}^{n-1}$ and $w_1 = e \in \mathbb{R}^n$;

compute min eigenpair (λ_n^1, x_n^1) and max eigenpair (λ_1^1, x_1^1) of A;

1: **while** stopcrit>tol **do** (main outer loop)

2: set $k \leftarrow k + 1$.

3: compute direction

$$g_k = \frac{\lambda_1^k}{\lambda_n^k}V^T\left(\frac{1}{\langle x_1^k, w_k\bullet x_1^k\rangle}\left(x_1^k\bullet x_1^k\right) - \frac{1}{\langle x_n^k, w_k\bullet x_n^k\rangle}\left(x_n^k\bullet x_n^k\right)\right) \tag{2.22}$$

4: perform projected gradient step

$$v_{k+1} = \Pi_{\hat{\Omega}}\left(v_k - t_k\frac{g_k}{\|g_k\|}\right) \tag{2.23}$$

where $\hat{\Omega}$ is as in (2.18) and set

$$w_{k+1} = e + Vv_{k+1}; \tag{2.24}$$

5: compute min eigenpair $(\lambda_n^{k+1}, x_n^{k+1})$ and max eigenpair $(\lambda_1^{k+1}, x_1^{k+1})$ of $A\,\mathrm{Diag}(w_{k+1})$;

6: update stopcrit;

7: **end while**(main outer loop)

Output: $\hat{D} := \mathrm{Diag}(w_{k+1})$.

Several remarks about Algorithm 2.2 are now given. First, the matrix V does not need to be stored in memory and is actually not needed as input. All the user needs to input is a subroutine that outputs Vv given a vector $v \in \mathbb{R}^{n-1}$. Likewise, the matrix $A\,\mathrm{Diag}(w_{k+1})$ does not need to be stored. Second, it follows

from Remark 2.14 that $g_k \in \bar{\partial}^\circ(\kappa(v_k))$, which is defined in (2.12). Finally, the projected gradient step in (2.23) can be performed very efficiently since projecting onto $\hat{\Omega}$ just involves computing the root of a simple equation.

2.3.1 Nonasymptotic Convergence Rate Analysis for Minimizing $\kappa(v)$ on $\hat{\Omega}$

In this subsection, we show a nonasymptotic convergence rate for Algorithm 2.2 for minimizing $\kappa(v)$ over $\hat{\Omega}$. More specifically, we show that

$$\min_{1 \leqslant k \leqslant K} \kappa(v_k) - \kappa_* \leqslant \epsilon$$

holds for $K = \mathcal{O}(1/\epsilon^2)$, where $\kappa_* = \min_{v \in \hat{\Omega}} \kappa(v)$. Our nonasymptotic convergence analysis of Algorithm 2.2 relies mostly on the results from [14] and [11]. Like $\kappa(\mathring{\mathcal{D}}(d))$, the function $\kappa(\mathring{\mathcal{V}}(v))$ is pseudoconvex since it is the ratio of a convex function and a positive concave function. Also, it is easy to see that the set $\hat{\Omega}$ is just a box so it is a convex compact set that is easy to project onto. The only other assumption that needs to be verified to be able to show a nonasymptotic convergence rate of Algorithm 2.2 is that the function $\kappa(v) := \kappa(\mathring{\mathcal{V}}(v))$ is Lipschitz continuous on $\hat{\Omega}$. This result is proved in the following proposition.

Proposition 2.15. *The function $\kappa(v) := \kappa(\mathring{\mathcal{V}}(v))$ is Lipschitz continuous on $\hat{\Omega}$, i.e.,*

$$\|\kappa(v_1) - \kappa(v_2)\| \leqslant L\|v_1 - v_2\|, \quad \forall v_1, v_2 \in \hat{\Omega}$$

where $L > 0$ and $\hat{\Omega}$ is as in (2.18).

Proof. Suppose that $A \succ 0$ and $v \in \mathbb{R}^{n-1}$. Let $\lambda_{\max}(v) := \lambda_{\max}(A\,\mathrm{Diag}(e + Vv))$ and let $\lambda_{\min}(v) := \lambda_{\min}(A\,\mathrm{Diag}(e + Vv))$ so that $\kappa(v) = \lambda_{\max}(v)/\lambda_{\min}(v)$. Since $\lambda_{\max}(v)$ is a convex function, it is locally Lipschitz continuous on the interior of its domain. Hence, it follows from Proposition A.48(b) in [15] that since $\hat{\Omega}$ is a compact set which is contained in the interior of the domain of $\lambda_{\max}(v)$, $\lambda_{\max}(v)$ is L_1-Lipschitz continuous on $\hat{\Omega}$ where $L_1 > 0$. Likewise, by a similiar argument since $\lambda_{\min}(v)$ is a concave function, it is L_2 Lipschitz continuous on $\hat{\Omega}$ where $L_2 > 0$. Consider now the composite function $h(v) = g(\lambda_{\min}(v))$ where $g(y) = 1/y$. We claim that $h(v)$ is Lipschitz continuous on $\hat{\Omega}$. First, it is easy to see that $g(y) = 1/y$ is Lipschitz continuous on any interval (K, ∞) where $K > 0$. Let $L_3 > 0$ be the Lipschitz constant of $g(y)$ for $y \in \mathrm{range}(\lambda_{\min}(v))$, where $v \in \hat{\Omega}$, i.e.,

$$\|g(y_1) - g(y_2)\| \leqslant L_3\|y_1 - y_2\|, \quad \forall y_1, y_2 \in \mathrm{range}(\lambda_{\min}(v)) \text{ for } v \in \hat{\Omega}.$$

Now, let v_1 and $v_2 \in \hat{\Omega}$. We have that

$$\|h(v_1) - h(v_2)\| = \|g(\lambda_{\min}(v_1)) - g(\lambda_{\min}(v_2))\| \leqslant L_3\|\lambda_{\min}(v_1) - \lambda_{\min}(v_2)\| \leqslant L_3L_2\|v_1 - v_2\|.$$

Thus, we have shown that $h(v) = 1/\lambda_{\min}(v)$ is L_3L_2 Lipschitz continuous on $\hat{\Omega}$.

We now show that $\kappa(v)$ is Lipschitz continuous. First observe that any Lipschitz continuous function on a compact set is bounded. Hence, it holds that $|\lambda_{\max}(v)| \leqslant M_1$ and $|h(v)| \leqslant M_2$, where $v \in \hat{\Omega}$ and $M_1 \geqslant 0$ and $M_2 \geqslant 0$. Then, for any $v_1 \in \hat{\Omega}$ and $v_2 \in \hat{\Omega}$, the following holds:

$$\begin{aligned}
\|\kappa(v_1) - \kappa(v_2)\| &= \left\| \frac{\lambda_{\max}(v_1)}{\lambda_{\min}(v_1)} - \frac{\lambda_{\max}(v_2)}{\lambda_{\min}(v_2)} \right\| = \|\lambda_{\max}(v_1)h(v_1) - \lambda_{\max}(v_2)h(v_2)\| \\
&= \|\lambda_{\max}(v_1)h(v_1) - \lambda_{\max}(v_1)h(v_2) + \lambda_{\max}(v_1)h(v_2) - \lambda_{\max}(v_2)h(v_2)\| \\
&\leqslant M_1\|h(v_1) - h(v_2)\| + M_2\|\lambda_{\max}(v_1) - \lambda_{\max}(v_2)\| \\
&\leqslant M_1L_3L_2\|v_1 - v_2\| + M_2L_1\|v_1 - v_2\| = (M_1L_3L_2 + M_2L_1)\|v_1 - v_2\|,
\end{aligned}$$

where the first inequality follows from the fact that $\lambda_{\max}(v)$ and $h(v)$ are bounded and the second inequality follows from the fact that $\lambda_{\max}(v)$ and $h(v)$ are Lipschitz continuous. Hence, it follows from the above relations that the statement of the proposition holds with $L = (M_1L_3L_2 + M_2L_1)$.

■

We now state Theorem 2.16, which displays that Algorithm 2.2 is able to find an ϵ-approximate optimal solution of (2.17) with a sublinear $\mathcal{O}(1/\epsilon^2)$ rate of convergence. Both [14] and [11] prove a $\mathcal{O}(1/\epsilon^2)$ rate of convergence of subgradient methods for minimizing quasiconvex functions. In [14], Kiwiel also proves a $\mathcal{O}(1/\epsilon^2)$ rate when the constraint set $\hat{\Omega}$ is convex and compact and the objective is quasiconvex and Lipschitz continuous on $\hat{\Omega}$. Paper [11], also obtains a similar $\mathcal{O}(1/\epsilon^2)$ complexity result when the constraint set is closed and convex and the objective is quasiconvex and satisfies $\|\kappa(v)-\kappa_*\| \leqslant L\|v-v_*\|$ for all $v \in \mathbb{R}^{n-1}$. However, since the analysis of [11] relies on [14, Lemmas 6 and 14], it can be easily seen that the convergence analysis of [11] still holds if $\|\kappa(v)-\kappa_*\| \leqslant L\|v-v_*\|$ is assumed to hold for just $v \in \hat{\Omega}$ and not all $v \in \mathbb{R}^{n-1}$. Hence, the proof of our Theorem 2.16 will rely mostly on results from [11] since both the algorithm and complexity results presented in [11] are easier to digest. Theorem 2.16 is now presented.

Theorem 2.16. *Let $\epsilon > 0$ be a given tolerance and suppose that $K = \mathcal{O}(1/\epsilon^2)$. It then holds that*

$$\min_{1 \leqslant k \leqslant K} \kappa(v_k) - \kappa_* \leqslant \epsilon, \tag{2.25}$$

where $\kappa_* = \min_{v \in \hat{\Omega}} \kappa(v)$ and $\kappa(v) := \kappa(\mathring{\mathcal{V}}(v))$.

Proof. It is immediate to see that $\hat{\Omega}$ is a convex compact set and that $\kappa(v) := \kappa(\mathring{\mathcal{V}}(v))$ is a quasiconvex function since it is the ratio of a convex and a positive concave function. It follows from Proposition 2.15 that $\kappa(v)$ is Lipschitz continuous on $\hat{\Omega}$. Finally, it follows from Remark 2.14 that the update (2.23) in step 4 of Algorithm 2.2 is of the form $v_{k+1} = \Pi_{\hat{\Omega}}\left(v_k - t_k \frac{g_k}{\|g_k\|}\right)$ where $g_k \in \bar{\partial}^\circ(\kappa(v_k))$. Hence, it follows from these observations, the fact that $\{t_k\} = 1/\sqrt{k}$, and Theorem 3.2(ii) in [11] with $s = 1/2$, $\delta = \epsilon$, and $p = 1$ that the statement of Theorem 2.16 holds.

■

2.4 Optimality Conditions for ω Diagonal Preconditioning

We have an equivalent optimal diagonal scaling problem for ω. We do not bother with stating two equivalent problems since both the trace and determinant functions satisfy commutativity which clearly makes the two formulations equivalent.

We consider the problem of finding the ω-optimal diagonal preconditioner, i.e., we need to solve the fractional/pseudoconvex minimization problem

$$d_\omega^* \in \operatorname{argmin}\left\{\omega(\mathring{\mathcal{D}}(d)) = \frac{1}{\det(A)^{1/n}} \frac{\operatorname{tr}(\mathring{\mathcal{D}}(d))/n}{\det(\operatorname{Diag}(d))^{1/n}} : d \in \mathbb{R}^n_{++}\right\}.^{7} \tag{2.26}$$

One of the advantages of ω is that it is differentiable on $\mathbb{S}^n_{++}$ (see Theorem 2.17). Moreover, the ω-optimal scaling is the Jacobi scaling, i.e., the inverse of the diagonal matrix formed from the diagonal of A, see [5,12]:

$$d_\omega^* := \operatorname{diag}(\operatorname{Diag}(\operatorname{diag}(A))^{-1}) \in \operatorname{argmin}\{\omega(\mathring{\mathcal{D}}(\,d)) : d \in \mathbb{R}^n_{++}\}.$$

As for κ-diagonal preconditioning, the problem (2.26) is positively homogeneous and any positive multiple of d_ω^* does not change the optimal value. In the next result, we characterize its optimality condition.

[7] We note the interesting paradox that $\det(A)$ is ignored in the optimization problem as it is a constant.

Theorem 2.17. *The gradient for ω is given by*

$$\nabla\omega(A) = \frac{1}{n\det(A)^{1/n}}\left(I - \frac{\operatorname{tr} A}{n}A^{-1}\right).$$

And stationarity (optimality) of $\omega(\mathring{\mathcal{D}}(d))$ at $d = e$ is characterized by the gradient being zero, or equivalently by

$$0 \quad = \quad \frac{1}{n\det(A)^{1/n}}\left(\operatorname{diag} A - \frac{\operatorname{tr} A}{n}e\right). \tag{2.27}$$

Proof. We modify the result and proof in [12, Lemma 2.4]. We let $\operatorname{geom}(d)$ denote the geometric mean of the vector d. Then $\nabla \operatorname{geom}(d) = \frac{1}{n}\operatorname{geom}(d)\operatorname{diag}\left(\operatorname{Diag}(d)^{-1}\right)$. We first note that

$$\omega(\mathring{\mathcal{D}}(d)) \quad = \quad \frac{\operatorname{tr}(A\operatorname{Diag}(d))/n}{\det(A\operatorname{Diag}(d))^{1/n}} = \frac{1}{n\det(A)^{1/n}}\frac{\operatorname{tr}(A\operatorname{Diag}(d))}{\operatorname{geom}(d)}.$$

Therefore, the gradient at $d = e$ acting on the direction Δd is

$$(\nabla\omega(\mathring{\mathcal{D}}(e))^T\Delta d \quad = \quad \frac{1}{n\det(A)^{1/n}}\frac{\operatorname{geom}(e)\operatorname{tr}(A\operatorname{Diag}(\Delta d)) - \operatorname{tr}(A\operatorname{Diag}(e))\frac{1}{n}e^T\Delta d}{\operatorname{geom}(e)^2}. \tag{2.28}$$

That this being zero for all Δd yields (2.27).

■

Corollary 2.18. *Let $A \in \mathbb{S}^n_{++}$. Then the ω-optimal diagonal scaling $D^{1/2}AD^{1/2}$ is given by $D = \alpha\operatorname{Diag}(\operatorname{diag}(A))^{-1}$, for any $\alpha > 0$. The gradient $\nabla w(A) = 0$ if, and only if, $A = \alpha I$ for some $\alpha > 0$; i.e., A is ω-optimal if, and only if, it is a multiple of the identity.*

Proof. This follows from the equality conditions in the harmonic-arithmetic mean inequalities. We recall that the ω-optimal diagonal scaling is called the Jacobi scaling in the literature.

■

2.4.1 Optimal Block Diagonal Preconditioning

We now show that the above results extend directly to block diagonal preconditioning and this also results in partial Cholesky preconditioning. This extends the results for ω-optimal preconditioning with partial Cholesky structure in [12, Theorem 2.7] and the block diagonal ω-optimal preconditioner in [7, Prop. 3 part 3]. We let the linear transformation $\operatorname{blkdiag}(B)$ denote the block diagonal matrix formed from the arguments $B = (B_i)$, where $B_i, i = 1, \ldots, k$ are positive definite matrices of order $n_i, \sum_{i=1}^k n_i = n$. If $B_i = R_i^T R_i, i = 1, \ldots, k$ are the Cholesky factorizations, then

$$\mathbf{B} := \operatorname{blkdiag}(B) = \operatorname{blkdiag}(R)^T\operatorname{blkdiag}(R).$$

The adjoint mapping $\operatorname{blkdiag}^*(A)$ is the vector of diagonal blocks $B = (B_i)$.

The following result can be found in [6, Prop. 2.2].

Proposition 2.19 ([7, Prop. 3 part 3], [6, Prop. 2.2])**.** *Let $M = \begin{bmatrix} M_1 & M_2 & \ldots & M_k \end{bmatrix}$ be a full rank $m \times n$ matrix, $n \leqslant m$, where $M_i \in \mathbb{R}^{m\times n_i}$, M_i full rank n_i. Then the optimal block diagonal scaling*

$$\mathbf{B} := \operatorname{blkdiag}(B_1, B_2, \ldots, B_k),\ B_i \in \mathbb{R}^{m\times n_i},$$

that minimizes the measure ω, i.e.,

$$\min\ \omega((M\mathbf{B})^T(M\mathbf{B})),$$

is given by the arguments

$$B_i = (M_i^T M_i)^{-1/2}, \quad i = 1, \ldots, k.$$

Corollary 2.20. *Let $A \in \mathbb{S}^n_{++}$ and let $n_i, \sum_{i=1}^k n_i = n$ be chosen sizes of diagonal blocks A_i of A. Then the ω-optimal block diagonal preconditioner for A is:*

$$\bar{\mathbf{B}} \in \operatorname{argmin}\{\omega(\mathbf{B}^T A \mathbf{B}) : \mathbf{B} = \operatorname{blkdiag}(B_1, \ldots, B_k),\ B_i \in \mathbb{R}^{n_i \times n_i}\},$$

with $B_i = A_i^{-1/2}, \forall i = 1, \ldots, k$.

Proof. Let $A := M^T M$ be a full rank factorization of A. Define $M_i \in \mathbb{R}^{n \times n_i}$ for each $i = 1, \ldots, k$ such that $M = \begin{bmatrix} M_1 & M_2 & \ldots & M_k \end{bmatrix}$. Then the result follows from Proposition 2.19 by noting that $M_i^T M_i = A_i$ for all $i = 1, \ldots, k$.

■

Corollary 2.21. *We get the diagonal/Jacobi by using 1×1 diagonal blocks. We get (an extension) to the partial Cholesky preconditioner in [12] by taking $B_i = R_i^{-T}$, where $A = R_i^T R_i$ is the Cholesky factorization of A_i, and noting that the eigenvalues of $AD = A \operatorname{blkdiag}(R_i) \operatorname{blkdiag}(R_i^T)$ are equal to the eigenvalues of $\operatorname{blkdiag}(R_i^T) A \operatorname{blkdiag}(R_i)$.*

3 Computational experiments

Our computational experiments are divided into several parts. In Section 3.1, we compare the computational efficiency of the algorithm proposed in [16] with our subgradient algorithm for finding an optimal diagonal scaling D that minimizes $\kappa(AD)$.[8] We note that a diagonal scaling E that minimizes $\kappa((AE)^T(AE))$ is obtained in [16]. Hence, when using their code, we input a Cholesky factor B of A, i.e., $A = BB^T$, so that their algorithm finds E that minimizes $\kappa(EAE)$. Note that their code internally forms $A = BB^T$ if B is inputted. We do not compare with the algorithm in [9] since their code is not publicly available. Also, they consider different problem than us. Namely, they aim to find a diagonal preconditioner that is a convex combination of a small basis or list of diagonal preconditioners.

Note that for every experiment, $\kappa(A)$, $\kappa(AD)$, and $\kappa(EAE)$ were each evaluated by computing the matrix's minimum and maximum eigenvalue using MATLAB's eigs function with 10^{-10} precision.

In the second part of Section 3.1, we also illustrate the scalability and efficiency of our subgradient algorithm on large test matrices $A \in \mathbb{S}^n_{++}$ that the code by [16] cannot handle in a reasonable amount of time. Roughly, we consider dimensions n varying from $n = 15,000$ to $n = 150,000$ and consider test matrices taken from the SuiteSparse Matrix Collection [4], and also ones that are randomly generated with very large condition numbers.

In Section 3.2, we find near κ-optimal diagonal preconditioners for a large collection of matrices using our subgradient method and the method in [16]. We then compare the performance of preconditioned conjugate gradient for solving the linear system $Ax = b$ using the preconditioners found by both of the methods. We also compare with the results when no preconditioner is used.

Finally, in Section 3.3, we use the optimality conditions in Theorem 2.7 in order to construct a matrix A that is κ-optimal with respect to diagonal preconditioning, i.e., the κ-optimal diagonal scaling for A is the identity. We then compare with results after applying the Jacobi scaling, the ω-optimal scaling.

All experiments in Sections 3.1 and 3.2 are run on a 2023 Macbook Pro with an 8-core CPU and 128 GB of memory using MATLAB 2024a. The medium experiments in Section 3.3 were also run with this Macbook Pro while the very large instances were run with a large Linux machine from University of Waterloo with 256 GB of memory.

[8] Recall that in this paper $\kappa(AD) = \lambda_{\max}(AD)/\lambda_{\min}(AD) = \kappa(D^{1/2} A D^{1/2})$.

3.1 Minimizing κ Efficiently

In this subsection we compare the algorithm proposed by [16] with our subgradient method for finding a diagonal scaling D that minimizes $\kappa(AD)$.[9] The authors in [16] derive a convex *SDP, semidefinite programming*, relaxation to this nonconvex optimization problem and apply an interior point method to it to find an appropriate diagonal scaling. On the other hand, we work directly with a pseudoconvex vector minimization problem to find a diagonal scaling which reduces κ. More specifically, we apply a subgradient method to the pseudoconvex problem (2.17). As mentioned in Section 2.3, this formulation also has the advantage that the positive homogeneity has been removed, thus leading to increased computational stability.

We find that our subgradient method, Algorithm 2.2, is very efficient in minimizing (2.17). Moreover, we observe that we can get further computational speedups by using a special line-search inside our subgradient method that avoids the need to project onto the simplex $\hat{\Omega}$ and that also tries to maintain descent at every iteration. The details of this line-search algorithm, namely Algorithm C.1, can be found in Appendix C. In our computational experiments Algorithm 2.2 uses a tol of 10^{-4}, sets stopcrit=$2|\kappa(\mathring{\mathcal{D}}(d_{k+1})) - \kappa(\mathring{\mathcal{D}}(d_k))|/|\kappa(\mathring{\mathcal{D}}(d_{k+1})) + \kappa(\mathring{\mathcal{D}}(d_k))|$ at every iteration, and uses a maxiter of 500 and $\hat{\delta} = 10^{-3}$. Algorithm C.1 uses a maxiter of 80 and also takes its main tolerance to be 10^{-4}.

Tables D.1 and D.2 on Pages 25 and 26 in Appendix D.1 compare the computational efficiency of Algorithms 2.2 and C.1 with the algorithm in [16] for minimizing κ. Table D.1 (resp. Table D.2) presents the comparsion on matrices taken from the SuiteSparse Matrix Collection [4]. (resp. on matrices that were randomly generated using the sprandsym function in MATLAB). An extensive list of the SuiteSparse matrices used in Table D.1 can be found in Appendix E.

We now give several remarks about the results presented in Table D.1. The second block of columns of Table D.1 lists the percent reduction in κ that each method achieved for positive definite matrices $A \succ 0$ taken from the SuiteSparse Matrix Collection. More specifically, for our code the column computes the percentage reduction to be

$$\frac{\kappa(A) - \kappa(AD)}{\kappa(A)} * 100$$

while for the code in [16], it computes the reduction to be

$$\frac{\kappa(A) - \kappa(EAE)}{\kappa(A)} * 100$$

where here D and E are the diagonal scalings found by our code and their code, respectively. Hence, a negative value in this column means that the method found an unfavorable diagonal scaling that actually made κ worse, i.e., $\kappa(EAE) > \kappa(A)$ or $\kappa(AD) > \kappa(A)$.

As seen from this column, both Algorithms 2.2 and C.1 found diagonal scalings which reduced κ much more significantly than [16]. Across all 18 SuiteSparse instances, Algorithm 2.2 reduced κ on average by 78.1% while Algorithm C.1 reduced κ on average by 68.6%. On the other hand, [16] reduced κ on average by 26.3%. It is worth noting that Algorithms 2.2 and C.1 reduced κ on every instance while [16] was unable to reduce κ on 11 of the 18 instances.

Not only were Algorithms 2.2 and C.1 able to reduce κ more than [16], both algorithms were also much faster in doing so. The last column of the table shows that Algorithms 2.2 and C.1 were faster than the algorithm in [16] on every instance considered. On average, Algorithm 2.2 was 145.0 times faster than [16] while Algorithm C.1 was 397.8 times faster than their method. On one instance, Algorithms 2.2 and C.1 were even over 1000 times faster than [16].

Similar results are presented in Table D.2 on 27 randomly generated instances. The κ condition number of all matrices was generated, using MATLAB's sprandsym function, to be between $2.0 * 10^6$ and $1.5 * 10^7$. Across the 27 instances, Algorithm 2.2 was able to reduce κ by 27.8% while Algorithm C.1 was able to reduce κ by 22.7%. On the other hand, [16] was unable to reduce κ on all 27 instances tested. This is not too surprising since [16] discusses that their algorithm is less stable if $\kappa(A)$ is too large. Finally, Table D.2

[9] Recall that the method in [16] finds a diagonal scaling E that actually minimizes $\kappa(EAE)$.

shows that Algorithm 2.2 was on average 18.8 times faster in minimizing κ than [16] while Algorithm C.1 was on average 58.2 times faster than their method.

In the last part of this subsection, we illustrate the scalability of our Algorithms 2.2 and C.1 in minimizing κ on large matrices. The dimensions of the matrices that were considered are between $14,800$ and $150,000$. Table D.3 in Appendix D.2 shows the efficiency of Algorithms 2.2 and C.1 on large positive definite test matrices A that were taken from the SuiteSparse Matrix Collection. The list of the specific matrices used from the collection can be found in Appendix E. Table D.4 in Appendix D.2 shows the efficiency of Algorithms 2.2 and C.1 on matrices that were randomly generated using MATLAB's sprandsym function. The matrices A were randomly generated with dimension n varying between $50,000$ and $150,000$. They were also generated to have very large κ values which vary between 10^{11} and $3 * 10^{11}$.

We see in Table D.3 that Algorithms 2.2 and C.1 performed well in finding diagonal scalings D that reduced κ on large matrices that were taken from the SuiteSparse Matrix Collection. On average, across the 11 large instances, Algorithm 2.2 reduced κ by 45.7% while Algorithm C.1 reduced κ by 41.9%. Both algorithms were also very efficient and took approximately just two minutes to minimize κ.

Table D.4 illustrates that Algorithms 2.2 and C.1 also perform very well in reducing κ on large test instances randomly generated to have large κ values. On average, across these 11 hard instances, Algorithm 2.2 reduced κ by 11.6% while Algorithm C.1 reduced κ by 4.4%.

3.2 PCG Comparison for Solving Linear Systems

In this subsection, we compare solving the linear system $Ax = b$ using PCG with the preconditioner found by Algorithm C.1, with solving the system using the preconditioner found by [16]. We also compare solving the linear system using PCG with no preconditioner with solving it using PCG with the preconditioner found by Algorithm C.1. The results are presented in Tables D.5 and D.6 on Pages 27 and 28 in Appendix D.3. All matrices A considered in both tables are randomly generated, using MATLAB's sprandsym function, with varying dimensions, densities, and κ values.

We see in Table D.5 that PCG takes less iterations and CPU time using the preconditioner found by Algorithm C.1 compared to using the preconditioner from [16]. On average, across the 47 instances in Table D.5, PCG takes approximately $436,711$ iterations using the preconditioner in Algorithm C.1 while it takes $548,391$ iterations using the preconditioner from [16]. Moreover, the average CPU time for PCG to solve the system with the preconditioner for Algorithm C.1 is 50.1 seconds; while it is 62.0 seconds for the algorithm in [16].

Table D.6 shows the improvement for PCG from using the preconditioner from Algorithm C.1. Across the 31 instances considered, PCG on average performed $654,318$ iterations with no preconditioning while it performed $518,601$ iterations using our preconditioner. Not only did the number of PCG iterations go down, but the CPU time for solving the linear system also improved using our preconditioner. The average CPU time for PCG to solve the system using no preconditioner was 32.6 seconds while the average CPU time for PCG to solve it using our preconditioner was 28.3 seconds.

3.3 From κ-optimal A to "Improve PCG" using ω-optimal scaling

We now study the effect of applying ω-optimal diagonal scaling to a κ-optimally diagonally scaled matrix A.[10] We find a κ-optimally diagonally scaled matrix $A \succ 0$ using Theorem 2.7, i.e., the maximal and minimal eigenvectors of A, x_1 and x_n, satisfy (2.5) and are chosen using (2.6). We then choose the remaining eigenvalues to be evenly spread out in the open interval $\lambda_i \in (\lambda_n, \lambda_1), i = 2, \ldots, n-1$, with corresponding orthonormal eigenvectors x_i, $i = 2, \ldots n-1$, chosen in the orthogonal complement of $\text{span}\{x_1, x_n\}$, i.e., x_i are orthogonal to both x_1 and x_n. Note that we use a (sparse) QR factorization to obtain the remaining $n-2$ orthonormal eigenvectors. Therefore, the density of A cannot be determined accurately in advance. The time to generate a random problem is typically very small and is hence negligible. For a more precise description of how A is constructed, see Lemma 3.1 below.

[10] For the large problems in Table D.8 we used the so-called fastlinux server, cpu155.math.private, a Dell PowerEdge R650 with two Intel Xeon Gold 6334 8-core 3.6 GHz (Ice Lake) and 256 GB.

Lemma 3.1. *Let* $x = \begin{pmatrix} \alpha \\ \beta \end{pmatrix} \in \mathbb{R}^n, n \geqslant 2$ *with*

$$\|\alpha\| = \|\beta\|, y = \begin{pmatrix} \alpha \\ -\beta \end{pmatrix}.$$

Then the optimality conditions in (2.5)

$$x \bullet x = y \bullet y,\ x^T y = 0,$$

hold. Moreover, we can permute both vectors $x \leftarrow Px, y \leftarrow Py$ *and the result still holds. We then construct eigenvalues* $\lambda_1 \gg \lambda_2 \geqslant \lambda_3 \geqslant \ldots \geqslant \lambda_{n-1} \gg \lambda_n$ *and assign* $x_1 \leftarrow x, x_n \leftarrow y$ *to get the eigenpairs* $(\lambda_i, x_i), i = 1, 2$. *The orthogonal eigenvectors for the eigenpairs* $(\lambda_i, x_i), i = 2, \ldots, n-1$ *are obtained using the last* $n-2$ *columns from a QR factorization (*MATLAB*) starting with the two vectors* x_1, x_n*:*

$$[Q, \sim] = qr\left(\begin{bmatrix} x_1 & x_n \end{bmatrix}\right).$$

Proof. Immediate upon verification and the properties of the QR (modified Gram-Schmidt) factorization. ■

After constructing A, we then apply ω-optimal diagonal preconditioning to A to implicitly get J as described above, i.e., $J = D^{1/2}AD^{1/2}$ where $D = \text{Diag}(1./\,\text{diag}(A))$. We then compare the number of PCG iterations of solving $Ax = b$ and $J(D^{-1/2}x) = D^{1/2}b$. Note that only the vector b and the matrices A and D are inputted to MATLAB's built-in PCG function. The matrix J is not inputted to MATLAB's PCG function. Each linear system that we consider in our experiments is solved by PCG using 5 different initial points so the iteration counts and runtimes reported for each experiment are averaged across these 5 runs. For the computational results showing the comparison between A and J, see Tables D.7 and D.8 in Appendix D.4 on Pages 28 and 29.

We now give several remarks about the results presented in Tables D.7 and D.8. Interestingly, on every instance tested applying the Jacobi or ω-optimal diagonal preconditioning to A led to significant improvement in PCG's performance for solving the linear system even though $\kappa(A) < \kappa(J)$. On average, across the 10 medium sized test instances in Table D.7, PCG performed 38.5 times more number of iterations on the κ-optimal linear system than it did on ω-optimally scaled linear system. Moreover, PCG also on average took 34.2 times longer (in terms of CPU time) on the κ-optimal linear system than it did on the ω-optimally scaled linear system.

Jacobi preconditioning led to even more significant improvements on the larger test instances presented in Table D.8. On average, across the 13 large test instances in Table D.8, PCG performed 47.4 times more number of iterations on the κ-optimal linear system than it did on ω-optimally scaled linear system. Moreover, PCG also on average took 39.3 times longer (in terms of CPU time) on the κ-optimal linear system than it did on the ω-optimally scaled linear system.

4 Conclusion

In this paper, we study the problem of minimizing the condition number of $D^{1/2}AD^{1/2}$ for a given matrix A and a diagonal preconditioner D. In particular, we focus on both the classical κ- and ω-condition numbers. We introduce new theoretical results and algorithms to minimize both condition numbers and compare the efficiency of the resulting preconditioners. For example, we show that the ω-optimal diagonal and block-diagonal preconditioning problems have the advantage that they have closed-form solutions which recover classical preconditioners found in the literature. In addition, we are able to start with a κ-optimally diagonally preconditioned matrix A and then show that applying the ω-optimal diagonal preconditioning on

A significantly improves results for PCG. On large instances, applying the ω-optimal scaling can even lead to a dramatic 40 to 50 times speedup.

Our approach for κ minimization begins with reformulating the condition number minimizations with quadratic argument $D^{1/2}AD^{1/2}$ into an equivalent problem involving the affine argument AD. This simplifies the optimality conditions and leads to a pseudoconvex $n \times 1$ vector minimization problem. Our strategy thus differs from existing approaches in the literature for κ-optimal diagonal preconditioning that turn the problem into an SDP, a $n \times n$ matrix optimization problem, that can be much more expensive to solve when n is large. To solve the more computationally tractable pseudoconvex vector reformulation, we develop a subgradient method whose main computational bottleneck at every iteration is just a minimum and maximum eigenpair computation. Our approach is thus significantly cheaper than the interior-point method proposed by [16], for solving the SDP, which involves inverting a possibly large dense matrix at every iteration.

Moreover, as the κ-condition number is a positively homogeneous function, and thus the minimization problem is Hadamard ill-posed, we present a second reformulation that removes the positive homogeneity. Solving this reformulation in practice leads to enhanced computational stability. We develop an efficient subgradient method which is tailored to solve this more computationally stable formulation. We conduct a careful convergence analysis which also shows that our subgradient method is able to find an ϵ-global optimal solution of this reformulation in at most $\mathcal{O}(1/\epsilon^2)$ iterations.

For instance, when tested on matrices from the *SuiteSparse Matrix Collection*, our two subgradient methods significantly outperformed (over 200 times faster) the SDP-based approach in [16]. Furthermore, our proposed methods consistently achieved much greater reductions in κ, often reducing it by more than half. Finally, our algorithms successfully handled large-scale problem instances involving up to hundreds of thousands of variables, within a matter of minutes, while [16] struggled with solving such problems in a reasonable amount of time. An interesting direction of future study is the development of scalable κ-optimal preconditioners that are block-diagonal, extending beyond the diagonal case.

A Assumptions and Technical Results from [14]

Let $f : \mathbb{R}^n \to \bar{\mathbb{R}} := \mathbb{R} \cup \{-\infty, \infty\}$ be an extended real-valued function with *domain* $\bar{D}$. For any set C, let $\operatorname{int} C$ and $\operatorname{cl} C$ denote its interior and closure, respectively. The following minimization problem is considered in [14]:

$$f_* := \inf \{f(x) : x \in X\}, \tag{A.1}$$

with the following assumptions on f and X:

A1 : $\operatorname{int} \bar{D}$ is nonempty and convex.

A2 : $\overline{\lim}_{t \downarrow 0} f(x + t(y - x)) \leqslant f(x)$, $\forall x \in \bar{D}, y \in \operatorname{int} \bar{D}$.

A3 : f is upper semicontinuous (usc) on $\operatorname{int} \bar{D}$, i.e., $f(x) = \lim_{\epsilon \downarrow 0} \sup_{B(x,\epsilon)} f$, $\quad \forall x \in \operatorname{int} \bar{D}$ where $B(x, \epsilon) := \{y : \|y - x\| \leqslant \epsilon\}$.

A4 : f is quasiconvex on $\operatorname{int} \bar{D}$, i.e., the set $\{x \in \operatorname{int} \bar{D} : f(x) \leqslant \alpha\}$ is convex $\forall \alpha \in \mathbb{R}$.

A5 : The constraint set $X \subset \mathbb{R}^n$ is closed convex, and $X \cap \operatorname{int} \bar{D} \neq \varnothing$.

The following lemma from [14] discusses that fractional programs with certain structures are quasiconvex and also provides one characterization of their quasisubgradients.

Lemma A.1 ([14, Lemma 4]). *Suppose $f(x) = a(x)/b(x)$ for all $x \in \operatorname{int} \bar{D}$, where a is a convex function, b is finite and positive on $\operatorname{int} \bar{D}$, $\operatorname{int} \bar{D}$ is convex, and one of the following conditions holds:*

(a) b is affine;

(b) a is nonnegative on $\operatorname{int} \bar{D}$ and b is concave;

(c) a is nonpositive on $\operatorname{int} \bar{D}$ and b is convex.

Then f is quasiconvex on $\operatorname{int} \bar{D}$ and for each $x \in \operatorname{int} \bar{D}$, if $\alpha := f(x)$ is finite then $a - \alpha b$ is convex and $\partial[a - \alpha b](x) \subset \bar{\partial}^\circ f(x)$.

The following lemma can be found in [14] and is useful for showing that assumptions (A1)-(A4) hold for our set-up in (2.8) since $\kappa(\mathring{\mathcal{D}}(d))$ is the ratio of a convex and concave function that is positive on Ω.

Lemma A.2 ([14, Remark 2]). *Suppose a and $\tilde{b} := -b$ are proper convex functions on $\operatorname{int} \bar{D}^a \cap \operatorname{int} \bar{D}^b$, a is nonnegative on the (open and convex set) $C := \left\{y \in \operatorname{int} \bar{D}^a \cap \operatorname{int} \bar{D}^b : b(y) > 0\right\} \neq \varnothing$, and*

$$f(x) := \begin{cases} a(x)/b(x), & \text{if } x \in C, \\ \sup_{y \in C} \overline{\lim}_{t \downarrow 0} f(x + t(y - x)), & \text{if } x \in \operatorname{bd} C, \\ \infty, & \text{if } x \notin \operatorname{cl} C \end{cases}$$

*where $\operatorname{bd} C$ denotes the boundary of C and $\bar{D}^a$ and $\bar{D}^b$ denote the domains of a and b, respectively. Then assumptions **A1-A4** hold with $\operatorname{int} \bar{D} = C$.*

B Proof of (2.14)

Proof of (2.14). Fix $y \in \mathbb{R}^n$. We want to verify that

$$\left\langle \frac{\lambda_1}{x_1^T (d \bullet x_1)} (x_1 \bullet x_1), y - d \right\rangle \leqslant \lambda_{\max}(\mathring{\mathcal{D}}(y)) - \lambda_{\max}(\mathring{\mathcal{D}}(d)). \tag{B.1}$$

Denote $D = \mathrm{Diag}(d)$. Note that $\lambda_{\max}(\mathring{\mathcal{D}}(d)) = \lambda_1$ and that

$$\begin{array}{rcl}\left\langle \frac{\lambda_1}{x_1^T(d\bullet x_1)}\left(x_1\bullet x_1\right), d\right\rangle &=& \frac{\lambda_1}{x_1^T(d\bullet x_1)}\left\langle \mathrm{diag}(x_1x_1^T), d\right\rangle \\ &=& \frac{\lambda_1}{x_1^TDx_1}\left\langle x_1x_1^T, \mathrm{Diag}(d)\right\rangle \\ &=& \frac{\lambda_1}{x_1^TDx_1}x_1^TDx_1 = \lambda_1.\end{array}$$

Hence, (B.1) becomes

$$\left\langle \frac{\lambda_1}{x_1^T\left(d\bullet x_1\right)}\left(x_1\bullet x_1\right), y\right\rangle - \lambda_1 \leqslant \lambda_{\max}(\mathring{\mathcal{D}}(y)) - \lambda_1 \iff \left\langle \frac{\lambda_1}{x_1^TDx_1}\left(x_1\bullet x_1\right), y\right\rangle \leqslant \lambda_{\max}(\mathring{\mathcal{D}}(y)).$$

This follows from

$$\begin{array}{rcl}\left\langle \frac{\lambda_1}{x_1^TDx_1}\left(x_1\bullet x_1\right), y\right\rangle &=& \frac{\lambda_1}{x_1^TDx_1}\left\langle \left(x_1\bullet x_1\right), y\right\rangle \\ &=& \frac{\lambda_1}{x_1^TDx_1}\left\langle x_1x_1^T, \mathrm{Diag}(y)\right\rangle \\ &=& \frac{1}{x_1^TA^{-1}x_1}x_1^T\,\mathrm{Diag}(y)x_1 \\ &=& \frac{x_1^TA^{-1/2}A^{1/2}\,\mathrm{Diag}(y)A^{1/2}A^{-1/2}x_1}{\|A^{-1/2}x_1\|^2} \\ &\leqslant& \lambda_{\max}(A^{1/2}\,\mathrm{Diag}(y)A^{1/2}) = \lambda_{\max}(\mathring{\mathcal{D}}(y)).\end{array}$$

The subgradient of $-\lambda_{\min}$ follows similarly.

■

C An Efficient Line-Search Subgradient Method

This section presents an efficient line-search subgradient method for minimizing $\kappa(v) := \kappa(\mathring{\mathcal{V}}(v))$ where $\mathring{\mathcal{V}}(v)$ is as in (2.15).

Let $w = e + Vv \in \mathbb{R}^n_{++}$ correspond to the current iterate v. Also, let σ be a small scalar between 0 and 1 and let $\Delta w = V\Delta v$ where $\Delta v = -\nabla\kappa(v)$ where $\kappa(v)$ is as in (2.16). From Lemma 2.3, we can use a ratio test and guarantee positive definiteness from $w + tV\Delta v = w + t\Delta w > \sigma w$, or equivalently $t\Delta w > (\sigma - 1)w$. Therefore, we conclude that the maximum step with safeguarding, so as not to get too close to the boundary, is

$$t_{\max}(\sigma) \cong t_{\max} := \min\left\{\frac{(\sigma-1)w_i}{\Delta w_i} : \Delta w_i < 0\right\} > 0. \tag{C.1}$$

Recall the gradient $\nabla\kappa(v) = \nabla(\kappa\circ\mathring{\mathcal{V}}(v))$ in Lemma 2.13. Our line search in Algorithm C.1 maintains:

LineSearch C.1 (for Algorithm C.1)**.** *Backtrack and maintain: (i) $t \leqslant t_{\max}$ from (C.1); (ii) monotonic nonincrease of the objective $\kappa(v+t\Delta v)$; and nonpositivity of the directional derivative $\nabla\kappa(v+t\Delta v)^T\Delta v \leqslant 0$.*

Algorithm C.1 is presented below.

Algorithm C.1 An Efficient Line-Search Subgradient Method for Minimizing $\kappa(v)$.

Inputs: A symmetric positive definite matrix $A > 0$, a max iteration count mainmaxiter, stopping tolerances maintoler> 0 and linesrchtoler> 0, and a small scalar $0 < \sigma \ll 1$.

set $k \leftarrow 0$, $v_1 = 0 \in \mathbb{R}^{n-1}$, $w_1 = e \in \mathbb{R}^n$, and $t_k \leftarrow \infty$;

compute a min eigenpair (λ_n^1, x_n^1) and max eigenpair (λ_1^1, x_1^1) of A;

compute $\kappa(v_1)$ and $\nabla\kappa(v_1)$ according to (2.16) and (2.19), respectively;

set relnormg$= \|\nabla\kappa(v_1)\|^2/(1+\kappa(v_1)^2)$;

while relnormg $>$ maintoler & $t_k >$ linesrchtoler & $k \leqslant$ mainmaxiter **do** (main outer loop)

  set $k \leftarrow k+1$;

  set $\Delta v_k \leftarrow -\nabla\kappa(v_k)$;

  use (C.1) with $\Delta w := V\Delta v_k$ and $w := w_k$ to evaluate $t_{\max}(\sigma)$;

  perform LineSearch C.1 to find t_k;

  set $v_{k+1} = v_k + t_k\Delta v_k$ and $w_{k+1} = e + Vv_{k+1}$;

  compute min eigenpair $(\lambda_n^{k+1}, x_n^{k+1})$ and max eigenpair $(\lambda_1^{k+1}, x_1^{k+1})$ of $A\,\mathrm{Diag}(w_{k+1})$;

  compute $\kappa(v_{k+1})$ and $\nabla\kappa(v_{k+1})$ according to (2.16) and (2.19), respectively;

  update relnormg$= \|\nabla\kappa(v_{k+1})\|^2/(1+\kappa(v_{k+1})^2)$;

end while(main outer loop)

Output: $\hat{D} := \mathrm{Diag}(w_{k+1})$.

In practice, we take linesearchtol to be 10^{-7} and maintoler to be 10^{-4}.

D Tables

D.1 Minimizing κ Efficiently: Section 3.1

Table D.1: Comparison of Algorithm in [16] and Algorithm C.1 for Min κ on SuiteSparse

Dim & Density, & $\kappa(A)$			% Reduction in κ			CPU Time (seconds)			CPU Ratios	
n	density	$\kappa(A)$	Alg. in [16]	Alg 2.2	Alg. C.1	Alg. in [16]	Alg 2.2	Alg C.1	(Alg. in [16])/Alg 2.1	(Alg. in [16])/AlgC.1
1074	1.1e-02	2.6e+07	0.0e+00	9.9e+01	9.9e+01	47.345	7.750	4.769	6.1	9.9
2003	2.1e-02	1.1e+10	0.0e+00	9.8e+01	6.8e+01	330.150	11.170	0.106	29.6	3127.6
3600	2.1e-03	1.8e+07	0.0e+00	4.5e+01	6.1e+01	535.706	6.547	9.983	81.8	53.7
3134	4.6e-03	2.6e+12	-9.7e-04	7.5e+01	7.0e+01	179.904	2.412	0.384	74.6	468.9
3562	1.3e-02	1.9e+11	0.0e+00	8.0e+01	7.8e+01	1620.562	16.285	1.697	99.5	954.7
1922	8.2e-03	1.7e+08	0.0e+00	9.6e+01	8.8e+01	187.552	3.920	0.330	47.8	569.1
4410	1.1e-02	9.5e+08	0.0e+00	9.5e+01	8.3e+01	3640.376	2.524	2.805	1442.2	1297.7
588	6.2e-02	2.8e+04	9.9e+01	9.0e+01	9.3e+01	21.036	0.400	1.254	52.6	16.8
494	6.8e-03	2.4e+06	9.0e+01	9.1e+01	3.3e+01	11.579	0.741	0.101	15.6	115.0
662	5.6e-03	7.9e+05	9.4e+01	7.4e+01	4.5e+01	13.342	0.294	0.215	45.4	62.0
1824	1.2e-02	1.9e+06	0.0e+00	8.6e+01	7.0e+01	190.816	1.820	1.497	104.9	127.5
2146	1.6e-02	1.7e+03	0.0e+00	5.5e+01	4.9e+01	430.804	1.911	13.432	225.5	32.1
2910	2.1e-02	6.0e+06	0.0e+00	8.4e+01	7.5e+01	631.282	2.763	3.509	228.4	179.9
237	1.8e-02	2.0e+07	-5.1e+01	7.9e+01	6.5e+01	0.909	0.556	0.059	1.6	15.4
957	4.5e-03	5.1e+09	2.8e+01	6.1e+01	6.5e+01	9.483	4.814	0.125	2.0	75.6
100	5.9e-02	1.6e+03	3.8e+01	3.5e+01	3.5e+01	0.773	0.031	0.600	24.8	1.3
468	2.4e-02	1.1e+04	8.3e+01	7.9e+01	7.3e+01	12.871	1.221	0.978	10.5	13.2
729	8.7e-03	2.4e+09	9.2e+01	8.3e+01	8.4e+01	38.291	0.327	0.971	117.3	39.5

Table D.2: Comparison of Algorithm in [16] and Algorithm C.1 for Min κ on Random

Dim & Density, & $\kappa(A)$			% Reduction in κ			CPU Time (seconds)			CPU Ratios	
n	density	$\kappa(A)$	Alg. in [16]	Alg 2.2	Alg. C.1	Alg. in [16]	Alg 2.2	Alg C.1	(Alg. in [16])/Alg 2.1	(Alg. in [16])/AlgC.1
2000	1.0e-03	2.0e+06	-4.8e+01	5.3e+01	4.4e+01	16.096	3.723	0.578	4.3	27.9
2500	8.1e-04	2.5e+06	-5.0e+01	5.2e+01	4.7e-13	22.520	7.593	0.093	3.0	241.6
3000	6.7e-04	3.0e+06	-1.8e+01	3.5e+01	1.3e+01	33.939	4.365	0.419	7.8	81.0
3500	5.8e-04	3.5e+06	-1.9e+01	4.3e+01	4.9e+01	43.244	7.401	1.644	5.8	26.3
4000	5.0e-04	4.0e+06	-4.4e+01	3.5e+01	4.8e+00	58.427	8.160	0.339	7.2	172.3
4500	4.5e-04	4.5e+06	-3.0e+01	3.3e+01	4.2e+01	70.001	4.003	2.368	17.5	29.6
5000	4.0e-04	5.0e+06	-4.4e+01	3.9e+01	3.9e+01	85.027	15.789	2.618	5.4	32.5
5500	3.7e-04	5.5e+06	-3.5e+01	2.8e+01	3.6e+01	102.306	3.171	3.028	32.3	33.8
6000	3.3e-04	6.0e+06	-4.3e+01	3.0e+01	3.4e+01	131.942	8.180	4.213	16.1	31.3
6500	3.1e-04	6.5e+06	-2.9e+01	2.7e+01	3.2e+01	150.446	7.180	4.973	21.0	30.3
7000	2.9e-04	7.0e+06	-2.9e+01	3.3e+01	3.0e+01	176.338	19.705	5.695	8.9	31.0
7500	2.7e-04	7.5e+06	-4.1e+01	3.2e+01	2.8e+01	201.914	22.698	6.576	8.9	30.7
8000	2.5e-04	8.0e+06	-1.9e+01	3.2e+01	2.7e+01	213.641	26.429	6.048	8.1	35.3
8500	2.4e-04	8.5e+06	-2.8e+01	2.5e+01	2.6e+01	255.923	10.842	6.544	23.6	39.1
9000	2.2e-04	9.0e+06	-1.5e+01	8.7e+00	9.3e+00	283.656	4.197	3.164	67.6	89.7
9500	2.1e-04	9.5e+06	-4.2e+01	2.6e+01	2.3e+01	326.533	18.083	7.343	18.1	44.5
10000	2.0e-04	1.0e+07	-3.2e+01	2.2e+01	1.8e+01	351.300	12.470	7.560	28.2	46.5
10500	1.9e-04	1.0e+07	-1.2e+01	2.7e+01	2.1e+01	385.074	34.258	9.799	11.2	39.3
11000	1.8e-04	1.1e+07	-4.1e+01	2.2e+01	1.9e+01	459.240	29.053	12.083	15.8	38.0
11500	1.7e-04	1.1e+07	-3.4e+01	2.3e+01	2.0e+01	491.016	21.515	12.149	22.8	40.4
12000	1.7e-04	1.2e+07	-3.0e+01	2.2e+01	1.9e+01	522.207	30.531	21.476	17.1	24.3
12500	1.6e-04	1.3e+07	-3.1e+01	4.4e+00	3.7e+00	554.995	20.494	4.251	27.1	130.6
13000	1.5e-04	1.3e+07	-2.9e+01	2.4e+01	1.8e+01	607.047	110.059	17.378	5.5	34.9
13500	1.5e-04	1.4e+07	-3.0e+01	2.2e+01	1.8e+01	672.009	31.280	24.296	21.5	27.7
14000	1.4e-04	1.4e+07	-2.9e+01	2.2e+01	1.7e+01	719.140	33.481	13.514	21.5	53.2
14500	1.4e-04	1.4e+07	-3.4e+01	1.5e+01	1.2e+01	905.783	19.809	11.997	45.7	75.5
15000	1.3e-04	1.5e+07	-4.1e+01	1.6e+01	1.1e+01	925.496	25.721	11.144	36.0	83.0

D.2 Minimizing κ on Larger Test Instances: Section 3.1

Table D.3: Algorithm C.1 for Min κ on Large SuiteSparse

Dim & Density, & $\kappa(A)$			% Reduction in κ		CPU Time	
n	density	$\kappa(A)$	Alg 2.2	Alg. C.1	Alg. 2.2	Alg C.1
14822	3.3e-03	2.0e+06	3.9e+01	3.0e+01	7.850	283.641
15439	1.1e-03	4.4e+12	8.0e+01	7.2e+01	8.734	2.185
15439	6.5e-05	6.1e+09	4.2e+01	5.4e+01	2.283	2.725
17361	1.1e-03	2.5e+06	1.3e+01	1.2e+01	1.097	9.086
17361	3.4e-03	1.1e+09	5.6e+01	3.0e+01	19.675	9.306
23052	2.2e-03	7.4e+11	9.0e+01	7.2e+01	503.668	6.933
30401	5.1e-04	5.8e+03	9.1e+01	9.3e+01	204.718	14.951
36441	4.3e-04	2.6e+03	8.5e+01	9.2e+01	459.869	59.632
40806	1.2e-04	8.1e+01	4.5e-02	1.1e-01	29.901	16.301
48962	2.1e-04	1.6e+05	6.2e+00	4.1e+00	22.109	43.123
150102	3.2e-05	1.3e+07	4.6e-03	1.5e+00	53.450	864.384

Table D.4: Algorithm C.1 for Min κ on Large Random

Dim & Density, & $\kappa(A)$			% Reduction in κ		CPU Time	
n	density	$\kappa(A)$	Alg 2.2	Alg. C.1	Alg. 2.2	Alg C.1
50000	3.8e-05	1.0e+11	1.7e+01	7.7e+00	244.577	74.798
60000	3.2e-05	1.2e+11	1.5e+01	6.5e+00	400.467	103.404
70000	2.7e-05	1.4e+11	1.4e+01	5.6e+00	492.698	123.284
80000	2.4e-05	1.6e+11	1.1e+01	5.0e+00	436.111	198.894
90000	2.1e-05	1.8e+11	1.0e+01	4.3e+00	550.245	223.299
100000	1.9e-05	2.0e+11	1.2e+01	4.0e+00	986.187	250.367
110000	1.7e-05	2.2e+11	1.1e+01	3.7e+00	1162.321	293.284
120000	1.6e-05	2.4e+11	1.1e+01	3.4e+00	1172.119	332.887
130000	1.5e-05	2.6e+11	8.5e+00	3.1e+00	1274.404	386.352
140000	1.4e-05	2.8e+11	9.1e+00	2.9e+00	1500.467	480.042
150000	1.3e-05	3.0e+11	8.7e+00	2.7e+00	1677.360	498.512

D.3 PCG Comparison for Solving Linear Systems: Section 3.2

Table D.5: PCG Comparison Using Preconditioners Found by Algorithm in [16] and Algorithm C.1

Dim, Density, & $\kappa(A)$			% Reduction in κ		PCG Iterations		PCG CPU Time	
n	density	$\kappa(A)$	Alg. in [16]	Alg. C.1	Alg. in [16]	Alg C.1	Alg. in [16]	Alg. C.1
1000	3.3e-03	1.0e+09	-2.6e+01	3.7e+01	115340	97612	1.02	0.90
1300	2.5e-03	1.3e+09	-2.6e+01	6.8e+01	150922	89341	1.71	1.02
1600	2.0e-03	1.6e+09	-4.3e+01	3.1e+01	178520	154553	2.46	2.13
1900	1.7e-03	1.9e+09	-3.5e+01	7.0e+01	202601	120176	3.29	1.98
2200	1.4e-03	2.2e+09	-4.9e+01	7.1e+01	246489	128340	4.84	2.53
2500	1.2e-03	2.5e+09	-2.3e+01	2.5e+01	251249	220359	5.62	4.95
2800	1.1e-03	2.8e+09	-4.2e+01	1.6e+01	290864	253696	7.19	6.28
3100	9.9e-04	3.1e+09	-2.9e+01	6.6e+01	305755	177081	8.33	4.84
3400	9.0e-04	3.4e+09	-3.3e+01	5.2e+01	327618	222784	9.76	6.62
3700	8.3e-04	3.7e+09	-2.5e+01	1.7e+01	330854	306415	10.72	9.89
4000	7.6e-04	4.0e+09	-4.4e+01	5.7e+01	365179	235355	12.76	8.20
4300	7.1e-04	4.3e+09	-2.9e+01	5.3e+01	376541	257040	14.15	9.69
4600	6.6e-04	4.6e+09	-4.2e+01	3.8e+01	401636	307104	16.18	12.34
4900	6.2e-04	4.9e+09	-1.1e+01	8.7e+00	403730	383093	17.33	16.48
5200	5.8e-04	5.2e+09	-3.7e+01	4.9e+01	458830	302912	20.81	13.77
5500	5.5e-04	5.5e+09	-3.3e+01	3.8e+01	460970	343189	22.21	16.50
5800	5.2e-04	5.8e+09	-3.2e+01	4.5e+01	467859	337635	23.68	17.05
6100	4.9e-04	6.1e+09	-5.0e+01	4.4e+01	507732	352004	26.98	18.66
6400	4.7e-04	6.4e+09	-2.2e+01	4.3e+01	487284	358570	27.18	20.07
6700	4.4e-04	6.7e+09	-4.0e+01	4.1e+01	525544	379455	30.70	22.17
7000	4.2e-04	7.0e+09	-2.9e+01	3.9e+01	529965	394440	32.41	24.04
7300	4.1e-04	7.3e+09	-5.3e+01	2.7e+00	569603	512406	36.14	32.53
7600	3.9e-04	7.6e+09	-2.9e+01	7.4e-13	556700	528549	36.83	34.94
7900	3.7e-04	7.9e+09	-2.2e+01	3.6e+01	571141	437729	39.26	30.09
8200	3.6e-04	8.2e+09	-6.6e+01	3.5e+01	614096	449955	43.92	32.18
8500	3.5e-04	8.5e+09	-2.8e+01	3.5e+01	570400	455269	42.29	33.78
8800	3.3e-04	8.8e+09	-3.6e+01	3.4e+01	626901	476385	48.30	36.86
9100	3.2e-04	9.1e+09	-1.8e+01	2.0e+01	599788	528119	47.93	41.95
9400	3.1e-04	9.4e+09	-2.0e+01	8.8e+00	630375	577302	51.57	47.41
9700	3.0e-04	9.7e+09	-2.9e+01	3.1e+01	640995	508810	54.60	43.39
10000	2.9e-04	1.0e+10	-4.5e+01	3.1e+01	685653	519018	60.32	45.62
10300	2.8e-04	1.0e+10	-2.5e+01	2.9e+01	667930	540192	94.18	73.77
10600	2.8e-04	1.1e+10	-4.4e+01	2.9e+01	715141	546513	100.73	72.12
10900	2.7e-04	1.1e+10	-3.9e+01	1.9e+01	724046	590975	122.17	103.60
11200	2.6e-04	1.1e+10	-2.4e+01	2.8e+01	697926	564635	123.62	98.01
11500	2.5e-04	1.1e+10	-8.9e+00	2.4e+01	674744	591017	123.21	105.38
11800	2.5e-04	1.2e+10	-3.9e+01	1.7e+01	759543	631727	126.21	101.22
12100	2.4e-04	1.2e+10	-2.9e+01	2.6e+01	753466	602102	124.03	102.56
12400	2.3e-04	1.2e+10	-4.3e+01	2.5e+01	776591	613785	123.09	90.23
12700	2.3e-04	1.3e+10	-3.3e+01	2.5e+01	776271	621762	137.31	106.14
13000	2.2e-04	1.3e+10	-3.2e+01	1.1e+01	770468	683497	138.04	122.07
13300	2.2e-04	1.3e+10	-2.7e+01	2.4e+01	763526	646019	138.30	116.34
13600	2.1e-04	1.4e+10	-3.5e+01	1.7e+01	814873	680430	141.08	118.19
13900	2.1e-04	1.4e+10	-5.5e+01	2.3e+01	865682	654026	160.41	125.69
14200	2.0e-04	1.4e+10	-4.4e+01	2.3e+01	857069	666636	162.56	130.88
14500	2.0e-04	1.5e+10	-4.3e+01	1.1e+01	872791	729504	173.93	141.72
14800	2.0e-04	1.5e+10	-3.0e+01	8.5e+00	833167	747884	166.70	150.00

Table D.6: PCG Comparison

Dim & Density		κ condition #		PCG Iterations		PCG CPU	
n	density	A	$D^{1/2}AD^{1/2}$	A	$D^{1/2}AD^{1/2}$	A	$D^{1/2}AD^{1/2}$
1000	1.9e-03	2.5e+09	1.6e+09	180178	152582	1.33	1.24
1300	1.5e-03	3.2e+09	1.0e+09	230315	134756	2.19	1.42
1600	1.2e-03	4.0e+09	2.7e+09	274483	242196	3.30	3.10
1900	1.0e-03	4.7e+09	1.0e+09	320968	156769	4.59	2.37
2200	8.7e-04	5.5e+09	1.6e+09	358721	206184	5.91	3.59
2500	7.7e-04	6.3e+09	2.4e+09	400460	257771	7.28	5.01
2800	6.8e-04	7.0e+09	5.8e+09	428318	391543	8.86	8.51
3100	6.2e-04	7.8e+09	2.6e+09	463490	273294	10.56	6.56
3400	5.6e-04	8.5e+09	3.0e+09	493529	296013	12.52	7.92
3700	5.2e-04	9.3e+09	5.1e+09	526851	397509	14.66	11.60
4000	4.8e-04	1.0e+10	4.1e+09	552169	361977	16.77	11.49
4300	4.4e-04	1.1e+10	4.7e+09	582535	393356	19.21	13.56
4600	4.2e-04	1.2e+10	6.1e+09	603376	449391	21.47	16.76
4900	3.9e-04	1.2e+10	1.2e+10	643577	643577	24.27	25.37
5200	3.7e-04	1.3e+10	6.4e+09	659469	469358	26.72	19.91
5500	3.5e-04	1.4e+10	7.7e+09	684824	517202	29.43	23.26
5800	3.3e-04	1.5e+10	7.7e+09	717033	523850	32.45	24.88
6100	3.1e-04	1.5e+10	8.3e+09	733571	553378	34.93	27.69
6400	3.0e-04	1.6e+10	9.0e+09	758107	579864	37.95	30.51
6700	2.9e-04	1.7e+10	9.6e+09	779761	592643	40.56	32.51
7000	2.7e-04	1.8e+10	1.0e+10	795966	629263	43.34	36.16
7300	2.6e-04	1.8e+10	1.8e+10	830346	828422	47.01	49.64
7600	2.5e-04	1.9e+10	1.2e+10	846830	670583	49.83	41.73
7900	2.4e-04	2.0e+10	1.3e+10	864006	699636	53.01	45.62
8200	2.3e-04	2.1e+10	1.3e+10	887065	718809	56.55	48.40
8500	2.2e-04	2.1e+10	1.4e+10	895498	724188	58.85	50.56
8800	2.2e-04	2.2e+10	1.4e+10	908428	749730	61.86	54.41
9100	2.1e-04	2.3e+10	1.5e+10	938959	769789	66.03	57.60
9400	2.0e-04	2.3e+10	2.0e+10	958843	899589	69.70	69.48
9700	2.0e-04	2.4e+10	2.4e+10	976855	976855	73.30	77.87
10000	1.9e-04	2.5e+10	1.7e+10	989326	816544	76.41	67.23

D.4 From κ-optimal A to "Improve PCG" using ω-optimal scaling: Section 3.3

Table D.7: PCG tol. $1e-7$; Medium, PC; A, κ-opt **VS** J, ω-opt of A

Dim & Density		Ratios in conds (J, A)		J: ω-opt of A		Ratios A/J	
n	density	$\kappa(J)/\kappa(A)$	$\omega(J)/\omega(A)$	iters	cpu	iters	cpu
5000	9.40e-03	2.98e+00	7.368e-01	19.4	3.927e-03	25.2	13.9
10000	7.44e-03	3.31e+00	7.362e-01	18.8	6.698e-03	36.6	30.5
15000	6.02e-03	2.64e+00	7.365e-01	17.0	1.418e-02	41.6	39.0
20000	4.57e-03	3.56e+00	7.363e-01	20.6	2.861e-02	41.6	41.1
25000	3.47e-03	3.71e+00	7.370e-01	20.6	8.006e-02	30.9	29.1
30000	2.42e-03	3.52e+00	7.362e-01	20.2	4.327e-02	48.0	43.0
35000	1.56e-03	2.77e+00	7.371e-01	16.0	4.174e-02	36.5	36.1
40000	8.93e-04	2.89e+00	7.364e-01	18.0	2.535e-02	46.7	38.9
45000	4.15e-04	3.93e+00	7.370e-01	21.8	1.820e-02	28.5	27.0
50000	4.12e-04	4.12e+00	7.361e-01	24.0	2.679e-02	49.5	43.7

Table D.8: PCG tol. $1e-7$; Large, Linux; A, κ-opt **VS** J, ω-opt of A

Dim & Density		Ratios in conds (J, A)		J: ω-opt of A		Ratios A/J	
n	density	$\kappa(J)/\kappa(A)$	$\omega(J)/\omega(A)$	iters	cpu	iters	cpu
60000	9.14e-03	4.27e+00	7.366e-01	18.2	8.894e-01	42.0	36.5
65000	7.78e-03	4.57e+00	7.362e-01	21.0	9.985e-01	52.9	47.9
70000	6.54e-03	3.93e+00	7.365e-01	17.0	8.110e-01	48.3	42.9
75000	5.44e-03	4.18e+00	7.363e-01	18.0	8.253e-01	56.4	49.2
80000	4.36e-03	4.08e+00	7.369e-01	17.0	7.202e-01	38.3	33.8
85000	3.47e-03	3.91e+00	7.362e-01	18.0	7.062e-01	60.9	54.0
90000	2.63e-03	3.92e+00	7.371e-01	17.6	6.050e-01	34.2	29.8
95000	1.91e-03	4.12e+00	7.364e-01	18.0	5.092e-01	48.0	41.6
100000	1.31e-03	4.13e+00	7.369e-01	18.2	4.029e-01	34.4	29.8
105000	8.09e-04	4.41e+00	7.361e-01	22.0	3.342e-01	58.8	50.3
110000	4.29e-04	3.58e+00	7.363e-01	17.0	1.709e-01	55.9	45.2
115000	1.74e-04	3.97e+00	7.369e-01	20.6	1.134e-01	31.6	23.7
120000	3.32e-05	3.14e+00	7.362e-01	20.2	3.648e-02	55.0	26.0

E List of SuiteSparse Matrices Used in Experiments

Table D.1 considers the following matrices from the SuiteSparse Matrix Collection:

"bcsstk08.mat", "bcsstk13.mat", "bcsstk21.mat", "bcsstk23.mat", "bcsstk24.mat"

"bcsstk26.mat", "bcsstk28.mat", "bcsstk34.mat", "494_bus.mat", "662_bus.mat"

"nasa1824.mat", "nasa2146.mat", "nasa2910.mat", "nos1.mat", "nos2.mat", "nos4.mat"

"nos5.mat", "nos7.mat"

Table D.3 considers the following matrices from the SuiteSparse Matrix Collection:

"Pres_Poisson.mat", "bcsstk25.mat", "bcsstm25.mat", "gyro_m.mat"

"gyro.mat", "bcsstk36.mat", "wathen100.mat", "wathen120.mat", "minsurfo.mat"

"gridgena.mat", "G2_circuit.mat"

Index

References

[1] A. Bock and M. Andersen, *Connecting Kaporin's condition number and the Bregman log determinant divergence*, tech. rep., 2025. 4

[2] R. Byrd and R. Nocedal, *A tool for the analysis of quasi-Newton methods with application to unconstrained minimization*, SIAM J. Numer. Anal., 26 (1989), pp. 727–739. 4

[3] R. Byrd, R. Nocedal, and Y.Yuan, *Global convergence of a class of quasi-Newton methods on convex problems*, SIAM J. Numer. Anal., 24 (1987), pp. 1171–1191. 4

[4] T. Davis and Y. Hu, *The University of Florida sparse matrix collection*, ACM Transactions on Mathematical Software (TOMS), 38 (2011), pp. 1–25. 18, 19

[5] J. Dennis Jr. and H. Wolkowicz, *Sizing and least-change secant methods*, SIAM J. Numer. Anal., 30 (1993), pp. 1291–1314. 3, 4, 16

[6] X. Doan, S. Kruk, and H. Wolkowicz, *A robust algorithm for semidefinite programming*, Optim. Methods Softw., 27 (2012), pp. 667–693. 17

[7] X. Doan and H. Wolkowicz, *Numerical computations and the ω-condition number*, Tech. Rep. CORR 2011-03, University of Waterloo, Waterloo, Ontario, 2011. www.math.uwaterloo.ca/~hwolkowi/henry/reports/ONE.pdf. 4, 17

[8] W. Gao, Y.-C. Chu, Y. Ye, and M. Udell, *Gradient methods with online scaling*, tech. rep., 2024. 3, 6

[9] W. Gao, Z. Qu, M. Udell, and Y. Ye, *Scalable approximate optimal diagonal preconditioning*, tech. rep., 2023. arXiv, 2312.15594. 3, 5, 6, 18

[10] J. A. George and J. W.-H. Liu, *Computer Solution of Large Sparse Positive Definite Systems*, Prentice-Hall, Englewood Cliffs, NJ, 1981. 3

[11] Y. Hu, J. Li, and C. Yu, *Convergence rates of subgradient methods for quasi-convex optimization problems*, Computational Optimization and Applications, (2020), pp. 183–212. 15, 16

[12] W. Jung, D. Torregrosa-Belen, and H. Wolkowicz, *The ω-condition number: applications to preconditioning and low rank generalized Jacobian updating*, Comput. Optim. Appl., 91 (2025), pp. 235–282. 3, 4, 16, 17, 18

[13] I. Kaporin, *New convergence results and preconditioning strategies for the conjugate gradient method*, Numer. Linear Algebra Appl., 1 (1994), pp. 179–210. 4

[14] K. Kiwiel, *Convergence and efficiency of subgradient methods for quasiconvex minimization*, Mathematical programming, 90 (2001), pp. 1–25. 1, 11, 12, 15, 16, 23

[15] J. Lee, *Smooth manifolds*, in Introduction to smooth manifolds, Springer, 2003, pp. 1–29. 15

[16] Z. Qu, W. Gao, O. Hinder, Y. Ye, and Z. Zhou, *Optimal diagonal preconditioning*, Operations Research, 73 (2024), pp. 1479–1495. 2, 3, 6, 18, 19, 20, 22, 25, 26, 27

[17] J. Shewchuk, *An Introduction to the Conjugate Gradient Method Without the Agonizing Pain*, Carnegie Mellon University, Pittsburgh, PA 15213, 1994. 4

[18] L. Tunçel and H. Wolkowicz, *Strengthened existence and uniqueness conditions for search directions in semidefinite programming*, Linear Algebra Appl., 400 (2005), pp. 31–60. 6

[19] N. van der Aa, H. ter Morsche, and R. Mattheij, *Computation of eigenvalue and eigenvector derivatives for a general complex-valued eigensystem*, Electron. J. Linear Algebra, 16 (2007), pp. 300–314. 8

[20] A. Wathen, *Preconditioning*, Acta Numer., 24 (2015), pp. 329–376. 5

[21] H. Wolkowicz and Q. Zhao, *An all-inclusive efficient region of updates for least change secant methods*, SIAM J. Optim., 5 (1995), pp. 172–191. 4

[22] Q. Zhao, *Measures for least change secant methods*, Master's thesis, University of Waterloo, 1993. 4