A New Conjugate Gradient Algorithm Incorporating Adaptive Ellipsoid Preconditioning

Renato D.C. Monteiro* Jerome W. O'Neal † Arkadi Nemirovski ‡

submitted October 6, 2004

Abstract

The conjugate gradient (CG) algorithm is well-known to have excellent theoretical properties for solving linear systems of equations $Ax = b$ where the $n \times n$ matrix A is symmetric positive definite. However, for extremely ill-conditioned matrices the CG algorithm performs poorly in practice. In this paper, we discuss an adaptive preconditioning procedure which improves the performance of the CG algorithm on extremely ill-conditioned systems. We introduce the preconditioning procedure by applying it first to the steepest descent algorithm. Then, the same techniques are extended to the CG algorithm, and convergence to an ϵ-solution in $\mathcal{O}(\log \det(A) + \sqrt{n} \log \epsilon^{-1})$ iterations is proven, where $\det(A)$ is the determinant of the matrix.

Keywords: Conjugate gradient algorithm, steepest descent algorithm, ellipsoid method, condition number, linear systems of equations, convergence rate, polynomial algorithms.

AMS 2000 subject classifications: 65F10, 65F15, 65F35, 65F40.

1 Introduction

In this paper, we will present a new procedure for determining the solution x_* to the system $Ax = b$, where A is a real, positive definite $n \times n$ matrix and b is a real n-dimensional vector. Solution methods for this classic problem are varied and wide-ranging; some of these methods

*School of Industrial and Systems Engineering, Georgia Institute of Technology, Atlanta, GA, 30332-0205. (email: `monteiro@isye.gatech.edu`). This author was supported in part by NSF Grants CCR-0203113 and CCF-0430644 and ONR grant N00014-03-1-0401.

†School of Industrial and Systems Engineering, Georgia Institute of Technology, Atlanta, GA 30332-0205 (email: `joneal@isye.gatech.edu`). This author was supported in part by the NDSEG Fellowship Program sponsored by the Department of Defense.

‡School of Industrial Engineering and Management, Technion - Israel Institute of Technology, Technion City, Haifa 32000, Israel (email: `nemirovs@ie.technion.ac.il`).

date back centuries (e.g. Gaussian elimination), while others are more recent. However, all methods for solving the above system of equations fall into one of two primary categories: direct methods and iterative methods. Direct methods first build a factorization of A, then perform a series of substitutions to determine x_*. In contrast, iterative methods create a sequence of points $\{x_j\}$ which converge to x_*.

Iterative methods possess several advantages when compared with their direct counterparts, including (1) the development of intermediate, "approximate" solutions, (2) faster performance on sparse, well-conditioned systems, and (3) lower memory storage requirements. On the other hand, iterative methods have a convergence rate which depends on the condition number of the matrix A. This, combined with the cumulative effects of roundoff errors in finite-precision arithmetic, may make iterative methods ineffective when employed on extremely ill-conditioned systems.

The most well-known of the iterative methods is the conjugate gradient (CG) method. This method is known to have excellent theoretical properties, to include n-step finite termination. However, under finite arithmetic these properties are lost, and the CG method behaves similarly to other iterative methods, with a convergence rate proportional to the square root of the condition number of A.

In this paper, we will adapt the CG method so that our convergence rate depends, not on the square root of the condition number of A, but on the logarithm of the determinant of A. We do this by using an adaptive preconditioning strategy, one which first determines the quality of our preconditioner matrix at the current iterate. If the quality of the preconditioner is good, then we use it to perform a standard CG iteration. If the preconditioner is of poor quality at the current iterate, the preconditioner will be updated by multiplying it by a rank-one update matrix. This update matrix, which incorporates ideas from the Ellipsoid Method (see e.g. [5, 8]) and is reminiscent of the update matrices used in space dilation methods (see e.g. [13, 14]), reduces the determinant of the preconditioned matrix, while keeping the minimum eigenvalue bounded away from zero. We note that our algorithm places one key restriction on A, namely that the minimum eigenvalue of A be at least one.

The early development of the CG method dates to the 1950's, particularly to the seminal work of Hestenes and Stiefel [4]. CG methods based on preconditioners, known as preconditioned CG (PCG) methods, we first proposed in the early 1960's. Since then, a wide array of preconditioners have been suggested, many for specific classes of problems. For descriptions of preconditioners currently employed with the PCG algorithm, see [3, 10]. The early history of the PCG method is detailed in the survey work by Golub and O'Leary [2]. The PCG method has been widely used in optimization; see for example [7, 9]. Finally, for those unfamiliar with the PCG method, a good introduction is presented in [12].

Our paper is organized as follows. Subsection 1.1 presents terminology and notation which are used throughout the paper. Section 2 introduces the adaptive preconditioning strategy in the context of the steepest descent method in order to avoid obfuscating its main ideas due to the challenges inherent in the analysis of the PCG method. Section 3 is devoted to two sets of results pertaining to the PCG method. In Subsection 3.1, some classical theoretical results are reviewed, and in Subsection 3.2, some new convergence rate results are obtained in the case where the preconditioner matrix is of good quality over the first j iterates. In Section 4, the adaptive preconditioning strategy is extended to the context of the PCG method and as a result, an adaptive PCG method is developed and corresponding convergence results are derived. Finally, concluding remarks are presented in Section 5.

1.1 Terminology and Notation

Throughout this paper, uppercase Roman letters denote matrices, lowercase Roman letters denote vectors, and lowercase Greek letters denote scalars. The set $\mathbb{R}^{n\times n}$ denotes the set of all $n\times n$ matrices with real components; likewise, the set $\mathbb{R}^n$ denotes the set of n-dimensional vectors with real components. Linear operators (except matrices) will be denoted with script uppercase letters.

Given a linear operator $\mathcal{F} : E \mapsto F$ between two finite dimensional inner product spaces $(E, \langle\cdot,\cdot\rangle_E)$ and $(F, \langle\cdot,\cdot\rangle_F)$, its adjoint is the unique operator $\mathcal{F}^* : F \mapsto E$ satisfying $\langle\mathcal{F}(u), v\rangle_F = \langle u, \mathcal{F}^*(v)\rangle_E$ for all $u \in E$ and $v \in F$. A linear operator $\mathcal{G} : E \mapsto E$ is called self-adjoint if $\mathcal{G} = \mathcal{G}^*$. Moreover, $\mathcal{G}$ is said to be positive semidefinite (resp. positive definite) if $\langle\mathcal{G}(u), u\rangle_E \geq 0$ for all $u \in E$ (resp., $\langle\mathcal{G}(u), u\rangle_E > 0$ for all $0 \neq u \in E$).

Given a matrix $A \in \mathbb{R}^{n\times n}$, we denote its eigenvalues by $\lambda_i(A)$, $i = 1, \ldots, n$; its maximum and minimum eigenvalues are denoted by $\lambda_{\max}(A)$ and $\lambda_{\min}(A)$, respectively. If a symmetric matrix A is positive semidefinite (resp. positive definite), we write $A \succeq 0$ (resp., $A \succ 0$); also, we write $A \succeq B$ to mean $A - B \succeq 0$. Given $A \succ 0$, the condition number of A, denoted $\kappa(A)$, is equal to $\lambda_{\max}(A)/\lambda_{\min}(A)$. The size of the matrix A, denoted size(A), is the number of bits required to store the matrix A. The identity matrix will be denoted by I; its dimensions should be clear from the context. The notation $\|x\|$ denotes Euclidean norm for vectors, i.e. $\|x\| = \sqrt{x^T x}$. The function $\log\alpha$ denotes the natural logarithm of α. Finally, the set $B(0,1)$ denotes the Euclidean ball centered at the origin, i.e. $B(0,1) := \{z : \|z\| \leq 1\}$.

2 The Adaptive Steepest Descent Algorithm

In this section, we introduce the concept of adaptive preconditioning in the context of the preconditioned steepest descent (PSD) algorithm to develop an adaptive PSD algorithm. The section is divided into two subsections. In Subsection 2.1, we motivate the concept of adaptive preconditioning by discussing the PSD algorithm and showing how a generic update matrix F, applied to the original preconditioner matrix C, can improve the convergence of the algorithm. In Subsection 2.2, we present the update matrix F and prove that it has the properties required for the Adaptive PSD Algorithm in Subsection 2.1.

2.1 Motivation from the Steepest Descent Method

In this subsection, we discuss the motivation behind adaptive preconditioning procedures.

Let $x_* = A^{-1}b$ denote the unique solution to $Ax = b$. The following "energy" function will play an important role in the analysis of the algorithms discussed in this paper:

$$\Phi_A(x) \;:=\; \frac{1}{2}(x - x_*)^T A(x - x_*). \tag{1}$$

Notice that the gradient of this function is

$$\nabla\Phi_A(x) \;=\; Ax - b \;=:\; g(x).$$

The methods described in this paper reduce the energy function (1) at each step. We note, however, that this function is never evaluated in the course of our algorithms. Rather, it just serves as an analytic tool in the complexity analysis of the algorithms discussed in this paper.

All methods in this paper are gradient methods. Recall that a gradient method (for minimizing $\Phi_A(\cdot)$) is a method which generates a sequence of iterates $\{x_k\}$ according to $x_{k+1} = x_k + \alpha_k d_k$, where d_k is a search direction satisfying $d_k^T g(x_k) < 0$ and $\alpha_k > 0$ is a stepsize chosen so that $\Phi_A(x_{k+1}) < \Phi_A(x_k)$ (see page 25 in Bertsekas [1]). In this paper, we will only discuss gradient methods in which the sequence of stepsizes $\{\alpha_k\}$ is chosen using the minimization rule, i.e. $\alpha_k = \operatorname{argmin}\{\Phi_A(x_k + \alpha d_k) : \alpha \geq 0\}$.

The following definition provides a means for determining "good" search directions in gradient methods.

Definition 1 Given a constant $\zeta > 0$ and a point $x \in \mathbb{R}^n$ such that $g := g(x) \neq 0$, we say that a search direction $0 \neq d \in \mathbb{R}^n$ is *ζ-scaled at x* if

$$\sqrt{(g^T A^{-1} g)(d^T A d)} \leq -\sqrt{\zeta}(g^T d). \tag{2}$$

It turns out that if d is ζ-scaled at x, then minimizing Φ_A along the ray $\{x + \alpha d : \alpha \geq 0\}$ yields a significant reduction in $\Phi_A(\cdot)$. Indeed, Definition 1 ensures that d is a descent direction at x, since the left hand side of (2) is strictly positive. Moreover, it can be shown that

$$\alpha_{\text{new}} \;:=\; \operatorname{argmin}\{\Phi_A(x + \alpha d) : \alpha \geq 0\} \;=\; -\frac{d^T g}{d^T A d}$$

and

$$\frac{\Phi_A(x) - \Phi_A(x_{\text{new}})}{\Phi_A(x)} = \frac{(g^T d)^2}{(g^T A^{-1} g)(d^T A d)},$$

where $x_{\text{new}} = x + \alpha_{\text{new}} d$ (see Chapter 7.6 in Luenberger [6]). Hence, if d is ζ-scaled at x, Definition 1 implies that the next iterate x_{new} satisfies

$$\Phi_A(x_{\text{new}}) \leq \left(1 - \frac{1}{\zeta}\right) \Phi_A(x). \tag{3}$$

Thus, if all search directions of a gradient method are ζ-scaled at their respective iterates, the method will obtain an iterate x_k such that $\Phi_A(x_k) \leq \epsilon \Phi_A(x_0)$ in at most $\mathcal{O}(\zeta \log \epsilon^{-1})$ iterations.

We now give a sufficient condition for a search direction d of the form $d = -CC^T g(x)$, where $C \in \mathbb{R}^{n \times n}$ is an invertible matrix, to be well-scaled at x. We first give a definition.

Definition 2 For a given constant $\nu > 0$, a matrix $C \in \mathbb{R}^{n \times n}$ is called a *ν-preconditioner at x* if C is invertible and $g^T C(C^T AC)C^T g \leq \nu \|C^T g\|^2$, where $g = g(x)$.

Proposition 2.1 *If $C^T AC \succeq \xi I$ for some $\xi > 0$, and if C is a ν-preconditioner at a point x such that $g = g(x) \neq 0$, then $d = -CC^T g$ is (ν/ξ)-scaled at x.*

Proof: Note first that $d \neq 0$ since $g \neq 0$ and C is invertible. The assumption $C^T AC \succeq \xi I$ implies that $(C^T AC)^{-1} \preceq \xi^{-1} I$, and hence

$$g^T A^{-1} g \; = \; (C^T g)^T (C^T AC)^{-1} (C^T g) \; \leq \; \xi^{-1}(g^T CC^T g) \; = \; -\xi^{-1} g^T d.$$

The assumptions that C is a ν-preconditioner at x and $d = -CC^T g$ clearly imply that $d^T Ad \leq -\nu(g^T d)$. The conclusion that $d = -CC^T g$ is (ν/ξ)-scaled at x follows by noting Definition 1 and combining the above two inequalities. ∎

We will now outline a basic step of our adaptive preconditioned steepest descent (APSD) algorithm under the assumption that $A \succeq I$. At every iteration of this method, we generate preconditioners C satisfying Definition 2 and $C^T AC \succeq I$. For reasons that will become apparent later, we assume from now on that $\nu > n$. Suppose that x is the current iterate and C is an invertible matrix such that $C^T AC \succeq I$. (If x is the first iterate, we can set $C = I$ since we are assuming that $A \succeq I$; otherwise, we can let C be the preconditioner used to compute the search direction at the previous iterate.) If C is a ν-preconditioner at x, we can use it to generate a search direction at x according to Proposition 2.1. If C is not a ν-preconditioner at x, we can obtain a ν-preconditioner at x by successively post-multiplying the most recent C by an invertible matrix F satisfying the following three properties:

P1. $F = \sigma_1 I + \sigma_2 pp^T$ for some vector $p \in \mathbb{R}^n$ and constants σ_1 and σ_2,

P2. $F(C^T AC)F \; = \; (CF)^T A(CF) \; \succeq \; I$, and

P3. $\det F \leq \eta(\nu) := \sqrt{n/\nu}\, \exp\{(1 - n/\nu)/2\} < 1$.

We will describe how to construct a matrix satisfying properties P1–P3 in Subsection 2.2. The process of replacing C by CF will be referred to as an *update* of C. For now, we will make a few observations regarding these updates. Property P2 ensures that the updated matrix CF still satisfies the requirement that $(CF)^T A(CF) \succeq I$. Moreover, properties P2 and P3 together ensure that after a finite number of updates of the form $C \leftarrow CF$, a ν-preconditioner C at x will be obtained. Indeed, since

$$\det F(C^T AC)F \; = \; (\det F)^2 \det C^T AC \; \leq \; \eta(\nu)^2 \det C^T AC, \tag{4}$$

we see that $\det C^T AC$ decreases by a factor of $\eta(\nu)^2$ each time an update $C \leftarrow CF$ is performed. Since by property P2, $\det C^T AC \geq 1$ for every preconditioner C generated, it is clear that only a finite number of updates can be performed.

Finally, property P1 ensures that the process of updating C to CF is a simple process requiring $\mathcal{O}(n^2)$ flops if C is kept in explicit form. On the other hand, if C is kept in factored form (i.e., $C = F_1 \cdots F_l$, where l is the total number of updates performed throughout the method), then each matrix F_j requires $\mathcal{O}(n)$ units of storage, and multiplying F_j by a vector requires $\mathcal{O}(n)$ flops.

We are now ready to state the main algorithm of this section using the above ideas.

Algorithm APSD

Start: Given $A \succeq I$, $x_0 \in \mathbb{R}^n$, $b \in \mathbb{R}^n$, and constants $\nu > n$ and $\epsilon > 0$.

1. Set $i = 0$, $g_0 = Ax_0 - b$, and $C = I$.
2. **While** $\Phi_A(x_i) > \epsilon \Phi_A(x_0)$ **do**
   - (a) $d = -CC^T g_i$
   - (b) $\alpha = -g_i^T d/(d^T A d)$
   - (c) **If** $\alpha < \nu^{-1}$ **then**
     - Create an update matrix F satisfying properties P1–P3 above
     - Set $C = CF$ and go to step 2(a)

     end (if)
   - (d) $x_{i+1} = x_i + \alpha d$
   - (e) $g_{i+1} = g_i + \alpha A d$
   - (f) Set $i = i + 1$

   end (while).

Let us make a few observations about the above algorithm. First, if $\nu \geq \lambda_{\max}(A)$, then it is clear that no updates will be performed and that the algorithm reduces to the standard SD algorithm. Hence, the novel and interesting case to consider is when ν is chosen so that $\nu < \lambda_{\max}(A)$. Second, notice that the test $\alpha < \nu^{-1}$ performed in step 2(c) of Algorithm APSD is equivalent to testing whether C is a ν-preconditioner at x_j. This follows by inserting the definition of d given in 2(a) into the formula for α in 2(b). Finally, observe that whenever the test in step 2(c) is satisfied, an update is made to the matrix C.

The following theorem details the main convergence results for Algorithm APSD.

Theorem 2.2 *Assume that $A \succeq I$ and that $\nu < \lambda_{\max}(A)$ in Algorithm APSD. Then, the following statements hold:*

(a) The number of updates is bounded by $N_\psi := (\log \det A)/(\psi^{-1} - 1 + \log \psi)$, where $\psi := \nu/n$.

(b) The number of iterates x_i generated by the algorithm cannot exceed $\lceil \nu \log \epsilon^{-1} \rceil$.

(c) The algorithm has an arithmetic complexity of $\mathcal{O}(n^2(N_\psi + \nu \log \epsilon^{-1}))$ flops if C is kept in explicit form, and $\mathcal{O}(\max\{nN_\psi, n^2\}(N_\psi + \nu \log \epsilon^{-1}))$ flops if C is kept in factored form.

Proof: For the proof of (a), let C_k denote the preconditioner matrix in Algorithm APSD after k updates have been performed. In view of (4), we have that $\det(C_k^T A C_k) \le \eta(\nu)^{2k} \det A$. Also, property P2 implies that $\det(C_k^T A C_k) \ge \det(I) = 1$. Combining these two inequalities yields $1 \le \eta(\nu)^{2k} \det A$. By taking logarithms on both sides of this equation, we get that $k \le [\log \det A]/[2 \log(1/\eta(\nu))]$. Statement (a) follows by substituting the definition of $\eta(\nu)$ given in P3 into this inequality.

For (b), notice that we require $\alpha \ge \nu^{-1}$ at iterate x_j before we generate a new iterate x_{j+1}. Equivalently, we ensure that the matrix C is a ν-preconditioner at iterate x_j before generating x_{j+1}. The assumption that $A \succeq I$ and property P2 ensure that $C^T A C \succeq I$; hence Proposition 2.1 implies that the search direction d used to generate x_{j+1} is ν-scaled at x_j. As a result, $\Phi_A(x_{j+1}) \le (1 - \nu^{-1})\Phi_A(x_j)$ by (3), and the result follows by using standard arguments.

For the proof of (c), we begin by claiming that the process used to create an update matrix F requires $\mathcal{O}(n^2)$ flops if C is kept in explicit form, and $\mathcal{O}(\max\{nN_\psi, n^2\})$ flops if C is kept in factored form. (The proof of this fact follows immediately from Theorem 2.6 and equation (10) in Subsection 2.2.) Based on this result, it is clear that a single iteration or update of Algorithm APSD requires $\mathcal{O}(n^2)$ flops if C is kept explicitly and $\mathcal{O}(\max\{nN_\psi, n^2\})$ flops if C is kept in factored form. The result follows from this observation and statements (a) and (b). ∎

It is interesting to examine how N_ψ varies for $\psi \in (1, \lambda_{\max}(A)/n)$. Note that N_ψ is a strictly decreasing function of ψ, since the function $\psi^{-1} - 1 + \log \psi$ is strictly increasing for all $\psi > 1$. Moreover, it is easy to see that $N_\psi \to \infty$ as $\psi \downarrow 1$. Next, if we denote the eigenvalues of A by $\lambda_i(A)$, we see that

$$\log \det A \;=\; \log\left(\prod_{i=1}^{n} \lambda_i(A)\right) \;=\; \sum_{i=1}^{n} \log \lambda_i(A) \;\le\; n \log \lambda_{\max}(A).$$

Hence, if $\psi \uparrow \lambda_{\max}(A)/n$, then we have

$$N_\psi \;\le\; \frac{n \log \lambda_{\max}(A)}{\log \psi - 1} \;=\; \mathcal{O}\left(\frac{n \log \lambda_{\max}(A)}{\log \lambda_{\max}(A) - \log n - 1}\right) \;=\; \mathcal{O}(n).$$

Hence, the value of N_ψ decreases from infinity to $\mathcal{O}(n)$ as ψ increases from one to $\lambda_{\max}(A)/n$.

While the SD algorithm is not necessarily polynomial in n and the size of A, the following lemma shows that Algorithm APSD is, under some reasonable assumptions on $\psi \in (1, \lambda_{\max}(A)/n)$.

Lemma 2.3 *Let $\psi := \nu/n$, and assume that $\max\{\psi, (\psi-1)^{-1}\} = \mathcal{O}(p(n))$ for some polynomial $p(\cdot)$. Assume also that A is a rational matrix such that $A \succeq I$. Then, the arithmetic complexity of Algorithm APSD is polynomial in n and the sizes of A and ϵ^{-1}.*

Proof: Let us first get an upper bound on $h(\psi) := [\psi^{-1} - 1 + \log\psi]^{-1}$. If $\psi > 3/2$, then it is clear that $h(\psi) = \mathcal{O}(1)$, so assume that $\psi \le 3/2$. Using the assumption that $(\psi-1)^{-1} = \mathcal{O}(p(n))$ and the fact that $\log\psi \ge (\psi-1) - (\psi-1)^2/2$ for all $\psi \ge 1$, we have that

$$\begin{aligned}
[\psi^{-1} - 1 + \log\psi]^{-1} \;&\le\; \left[\frac{1-\psi}{\psi} + (\psi-1) - \frac{(\psi-1)^2}{2}\right]^{-1} \;=\; \left[(\psi-1)^2\left(\frac{1}{\psi} - \frac{1}{2}\right)\right]^{-1} \\
&\le\; \left[(\psi-1)^2\left(\frac{2}{3} - \frac{1}{2}\right)\right]^{-1} \;=\; 6(\psi-1)^{-2} \;=\; \mathcal{O}(p^2(n)).
\end{aligned}$$

Hence, we see that $N_\psi = \mathcal{O}(p^2(n)\log\det A)$. Next, Theorem 3.2 of [11] shows that size$(\det A) \le 2\,$size(A). Using this result along with the fact that $\log\det A = \mathcal{O}(\text{size}(\det A))$ and the bound on N_ψ, we have that $N_\psi = \mathcal{O}(p^2(n)\text{size}(A))$. It is clear that the assumption $\psi = \mathcal{O}(p(n))$ implies that $\nu = \psi n = \mathcal{O}(np(n))$. The result follows from these facts and Theorem 2.2(c). ∎

2.2 The Ellipsoid Preconditioner

In this subsection, we will show how we can construct a matrix F which satisfies properties P1–P3.

Our first lemma gives necessary and sufficient conditions for F to satisfy P2.

Lemma 2.4 *Let $\widehat{A} \succ 0$ and $F \in \mathbb{R}^{n\times n}$ be given. Then, $F\widehat{A}F \succeq I$ if and only if $E(\widehat{A}) \subseteq \mathcal{E}(F)$, where*

$$\begin{aligned}
\mathcal{E}(F) \;&:=\; \{Fu : u \in B(0,1)\}, \\
E(\widehat{A}) \;&:=\; \{z : z^T\widehat{A}z \le 1\}.
\end{aligned}$$

Proof: Let U denote the boundary of $B(0,1)$, i.e. $U := \{u : u^Tu = 1\}$. We have

$$\begin{aligned}
E(\widehat{A}) \subseteq \mathcal{E}(F) \;&\Leftrightarrow\; Fu \notin \operatorname{int} E(\widehat{A}), \quad \forall\, u \in U \\
&\Leftrightarrow\; (Fu)^T\widehat{A}(Fu) = u^T(F\widehat{A}F)u \ge 1, \quad \forall\, u \in U \\
&\Leftrightarrow\; F\widehat{A}F \succeq I.
\end{aligned}$$

∎

Our update matrix F possesses the following special property: its ellipsoid $\mathcal{E}(F)$ is the minimum volume ellipsoid containing a certain "stripe" Π intersected with the unit ball. The next lemma provides the details surrounding the construction of F.

Lemma 2.5 *Let a unit vector $p \in \mathbb{R}^n$ and a constant $\tau < 1$ be given, and consider the stripe $\Pi := \Pi(p,\tau) = \{z : |z^T p| \le \tau\}$. Then, the smallest volume ellipsoid containing $\Pi \cap B(0,1)$ is $\mathcal{E}(F)$, where*

$$F \;=\; F(p,\tau) \;:=\; \mu(I - pp^T) + \theta pp^T, \tag{5}$$

with

$$\theta \;=\; \theta(\tau) \;:=\; \min\{\tau\sqrt{n}, 1\}, \;\; \text{and} \tag{6}$$

$$\mu \;=\; \mu(\tau) \;:=\; \sqrt{\frac{n-\theta^2}{n-1}}. \tag{7}$$

Moreover, if $\tau < n^{-1/2}$, we have

$$\det F \;\le\; \tau\sqrt{n}\; \exp\{(1-\tau^2 n)/2\} \;<\; 1. \tag{8}$$

Proof: For the proof of the first part of the lemma, see Theorem 2(ii) in [15]. We will only prove equation (8). Since p is a unit vector, it is clear that p is an eigenvector of F with eigenvalue θ. In addition, any vector perpendicular to p is also an eigenvector of F with eigenvalue μ. Since $\det F$ is equal to the product of its eigenvalues, it follows that $\det F = \theta\mu^{n-1}$. Also, the assumption that $\tau < n^{-1/2}$ and (6) imply that $\theta = \tau\sqrt{n} < 1$. Using the above observations together with the fact that $1 + \omega \le \exp(\omega)$ for all $\omega \in \mathbb{R}$, we obtain

$$\begin{aligned} \det F \;&=\; \theta\mu^{n-1} \;=\; \theta\left[\frac{n-\theta^2}{n-1}\right]^{\frac{n-1}{2}} \;=\; \theta\left[1+\frac{1-\theta^2}{n-1}\right]^{\frac{n-1}{2}} \;\le\; \theta\left[\exp\left\{\frac{1-\theta^2}{n-1}\right\}\right]^{\frac{n-1}{2}} \\ &=\; \theta\exp\{(1-\theta^2)/2\} \;=\; \tau\sqrt{n}\; \exp\{(1-\tau^2 n)/2\} < 1, \end{aligned}$$

where the last inequality is due to the facts that $f(s) = s\exp[(1-s^2)/2]$ is a strictly increasing function over the interval $[0,1]$ and $f(1) = 1$. ∎

We will now show how to construct a matrix F satisfying P1–P3 under the assumptions that $\widehat{A} = C^T A C \succeq I$, $\nu > n$, and C is not a ν-preconditioner at x. First, we observe that since $\widehat{A} \succeq I$, we have $E(\widehat{A}) \subseteq B(0,1)$. Suppose now that a unit vector p and a scalar τ are chosen so that $E(\widehat{A}) \subseteq \Pi$, where $\Pi = \Pi(p,\tau)$ is the stripe defined in Lemma 2.5. Then, the matrix $F = F(p,\tau)$ given by (5) clearly satisfies property P1 and, by Lemma 2.5, we have

$$E(\widehat{A}) \;\subseteq\; \Pi \cap B(0,1) \;\subseteq\; \mathcal{E}(F).$$

This together with Lemma 2.4 implies that F satisfies property P2.

We will now show how to construct a unit vector p and a scalar τ so as to ensure that $E(\widehat{A}) \subset \Pi(p,\tau)$ and that property P3 also holds under the assumptions above. Indeed, since C is *not* a ν-preconditioner at x, it follows from Definition 2 that $w := C^T g(x)$ satisfies

$$w^T \widehat{A} w > \nu \|w\|^2. \tag{9}$$

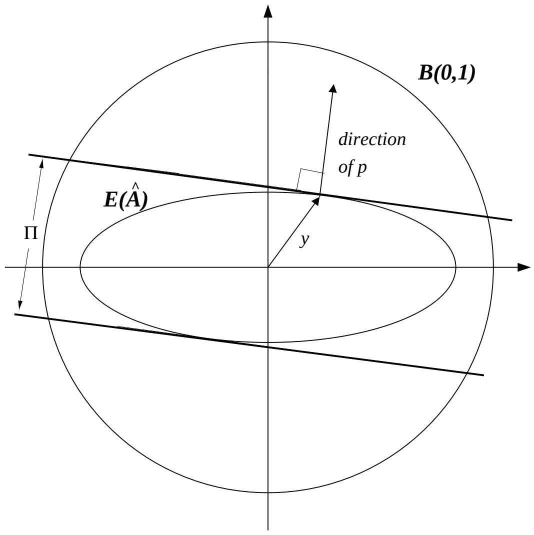

Figure 1: Vector p normal to boundary of $E(\widehat{A})$; stripe Π.

Now, letting $y := w/\sqrt{w^T\widehat{A}w}$, we have that $y^T\widehat{A}y = 1$, that is, y lies on the boundary of $E(\widehat{A})$. As Figure 1 illustrates, the boundary of our stripe Π will consist of the hyperplanes tangent to the boundary of $E(\widehat{A})$ at y and $-y$. It is easy to show that $\Pi = \{z : |z^T p| \le \tau\}$, where

$$p \;:=\; \frac{\widehat{A}w}{\|\widehat{A}w\|} \;=\; \frac{\widehat{A}y}{\|\widehat{A}y\|} \qquad \text{and} \qquad \tau \;:=\; \frac{\sqrt{w^T\widehat{A}w}}{\|\widehat{A}w\|}. \tag{10}$$

We note that the formula for p follows from the fact that the vector $\widehat{A}y$ is normal to the boundary of $E(\widehat{A})$ at the point y, since the gradient of the function $z^T\widehat{A}z$ at y is $2\widehat{A}y$. This construction clearly implies that $E(\widehat{A}) \subseteq \Pi$.

It remains for us to show that the matrix F satisfies property P3. Indeed, by (8) and the fact that $f(s) = s\exp\{(1-s^2)/2\}$ is strictly increasing on the interval $[0,1]$, we conclude that F satisfies property P3 whenever the condition $\tau\sqrt{n} \le \sqrt{n/\nu} < 1$ holds. The latter inequality holds due to the assumption that $\nu > n$, while the first inequality is due to the fact that relations (9) and (10) and the Cauchy-Schwartz inequality imply that

$$\tau \;=\; \frac{\sqrt{w^T\widehat{A}w}}{\|\widehat{A}w\|} \;=\; \frac{w^T\widehat{A}w}{\|\widehat{A}w\|\sqrt{w^T\widehat{A}w}} \;\le\; \frac{\|w\|}{\sqrt{w^T\widehat{A}w}} \;<\; \nu^{-1/2}.$$

Notice that the construction above does not use the facts that $\widehat{A} = C^TAC$ and $w = C^Tg(x)$, but only the facts that $\widehat{A} \succeq I$ and (9) hold. Hence in the discussion above we have established the following more general result.

Theorem 2.6 *Let $\widehat{A} \succeq I$ be given, and let $\nu > n$ be a given constant. Suppose that a vector $w \in \mathbb{R}^n$ satisfies (9), and let p, τ, and $F = F(p,\tau)$ be determined by equations (10) and (5), respectively. Then, the matrix F satisfies properties P1–P3, i.e.*

(P1) $F = \mu I + (\theta - \mu)pp^T$, *where* μ *and* θ *are given by (6) and (7), respectively;*

(P2) $F\widehat{A}F \succeq I$; *and*

(P3) $\det F \leq \sqrt{n/\nu}\exp\{(1-n/\nu)/2\} < 1$.

Before we end this section, we will briefly motivate the remaining part of this paper. Note that in Algorithm APSD, we either perform a standard SD iteration or an update of the preconditioner matrix C at each step. These two sets of computations require roughly the same number of arithmetic operations in view of Theorem 2.2(c), and hence may be considered equivalent from a complexity standpoint. For the purpose of the discussion in this paragraph, we will refer to both sets of computations as *iterations* of the whole algorithm. Recall that the standard SD algorithm has an iteration-complexity of $\mathcal{O}(\kappa(A)\log\epsilon^{-1})$. In view of our assumption that $\lambda_{\min}(A) \geq 1$, this implies that the SD algorithm has an iteration-complexity of $\mathcal{O}(\lambda_{\max}(A)\log\epsilon^{-1})$, and that all search directions in the SD algorithm are $\lambda_{\max}(A)$-scaled. By contrast, Algorithm APSD forces its search directions to be ν-scaled at every iteration; as a result, the algorithm achieves an improved iteration-complexity of $\mathcal{O}(N_\psi + \nu\log\epsilon^{-1})$. On the other hand, the conjugate gradient (CG) algorithm is known to possess an iteration-complexity of $\mathcal{O}(\sqrt{\kappa(A)}\log\epsilon^{-1}) \leq \mathcal{O}(\sqrt{\lambda_{\max}(A)}\log\epsilon^{-1})$. Hence, it is natural to conjecture whether we can reduce its iteration-complexity to $\mathcal{O}(N_\psi + \sqrt{\nu}\log\epsilon^{-1})$ by means of an adaptive preconditioning scheme. We will show that this is indeed possible. The development of an adaptive PCG (APCG) algorithm and the proof of its convergence properties is the subject of the remainder of this paper.

3 The Conjugate Gradient Method Revisited

In this section, we examine the conjugate gradient algorithm in detail. The section is divided into two subsections: Subsection 3.1 is devoted to a review of classical results, while Subsection 3.2 presents new convergence rate results obtained under the assumption that the preconditioner matrix is a good preconditioner at iterates $x_0, \ldots, x_j$.

3.1 Review of the Classical Conjugate Gradient Algorithm

In this subsection, we review the classical preconditioned conjugate gradient (PCG) algorithm and some of its well-known theoretical properties.

The PCG algorithm is an iterative algorithm which generates a sequence $\{x_i\}$ of approximate solutions to the system $Ax = b$, where $A \succ 0$. The PCG algorithm, which is stated next, uses an invertible matrix $Z \in \mathbb{R}^{n\times n}$ as a preconditioner.

PCG Algorithm:

Start: Given $A \succ 0$, $b \in \mathbb{R}^n$, an invertible matrix $Z \in \mathbb{R}^{n\times n}$, and $x_0 \in \mathbb{R}^n$.

1. Set $g_0 = Ax_0 - b$, $d_0 = -ZZ^Tg_0$, and $\gamma_0 = \|Z^Tg_0\|^2$.

2. **For** $i = 0, 1, \ldots$ **do**

   (a) $x_{i+1} = x_i + \alpha_i d_i$, where $\alpha_i = \gamma_i/(d_i^T A d_i)$

   (b) $g_{i+1} = g_i + \alpha_i A d_i$

   (c) $\gamma_{i+1} = \|Z^T g_{i+1}\|^2$

   (d) $d_{i+1} = -ZZ^T g_{i+1} + \beta_{i+1} d_i$, where $\beta_{i+1} = \gamma_{i+1}/\gamma_i$

   end (for).

To present the main theoretical results associated with the PCG algorithm, we introduce the following notation.

$$\begin{aligned} \widehat{A} &:= Z^T A Z, & (11)\\ \hat{g}_i &:= Z^T g_i, & (12)\\ \widehat{S}_i &:= \mathrm{span}\{\hat{g}_0, \ldots, \widehat{A}^i \hat{g}_0\}, & (13)\\ S_i &:= Z\widehat{S}_i = \{Zv : v \in \widehat{S}_i\}. & (14) \end{aligned}$$

A well-known interpretation of the PCG method is that the sequence of points $\{\hat{x}_i\}$ defined as $\hat{x}_i := Z^{-1} x_i$ is the one that is generated by the standard CG algorithm applied to the system $\widehat{A}\hat{x} = \hat{b}$, where $\widehat{A}$ is defined in (11) and $\hat{b} := Z^T b$. Moreover, the gradient of the energy function $\Phi_{\widehat{A}}(\cdot)$ associated with this system at $\hat{x}_i$ is equal to $\hat{g}_i$ as defined in (12).

The following proposition follows from the above observations and the properties of the standard CG algorithm:

Proposition 3.1 *Each step i of the PCG algorithm possesses the following properties:*

(a) $\widehat{S}_i = \mathrm{span}\{\hat{g}_0, \ldots, \hat{g}_i\}$*;*

(b) $\hat{g}_i^T \hat{g}_j = 0$ *for all* $i < j$*;*

(c) $\hat{g}_i^T \widehat{A} \hat{g}_j = 0$ *for all* $i \leq j - 2$*; and*

(d) $x_i = \mathrm{argmin}\{\Phi_A(x) : x \in x_0 + S_{i-1}\}$.

Proof: See e.g. pages 295-7 of [16]. ∎

We note that these properties may fail to hold under finite arithmetic. Indeed, the PCG method relies heavily on the fact that the search directions $\hat{d}_i$ in the transformed space are conjugate to one another (i.e. $\hat{d}_i^T \widehat{A} \hat{d}_j = 0$ for $i \neq j$). However, under finite-precision arithmetic, it is well-known that the search directions will often lose conjugacy and may become linearly dependent (see e.g. [3]). As a result, the PCG algorithm tends to perform poorly on extremely ill-conditioned systems.

On the other hand, when $\widehat{A}$ is well-conditioned, the PCG algorithm performs reasonably well. As we will see in the next subsection, a weaker condition for the PCG algorithm to perform well at iterates $x_0, \ldots, x_j$ is for Z to be a good preconditioner at these iterates.

3.2 Revisiting the Performance of the PCG Algorithm

In this subsection, we examine the rate of convergence of the iterates $x_0, \ldots, x_j$ of the PCG algorithm under the assumption that Z is a good preconditioner at those iterates.

First, assume that Z is a ν-preconditioner at x_i and that $Z^T AZ \succeq \xi I$ for some positive constants ν and ξ. Let $\tilde{x}_{i+1}$ be the point obtained by taking a step of the PSD algorithm at x_i using Z as preconditioner. Using the fact that, by Lemma 2.1, $d = -ZZ^T g_i$ is ν/ξ-scaled at x_i along with (3) and statements (a) and (d) of Proposition 3.1, we have that

$$\Phi_A(x_{i+1}) \;\le\; \Phi_A(\tilde{x}_{i+1}) \;\le\; \left(1 - \frac{1}{\chi}\right) \Phi_A(x_i), \tag{15}$$

where $\chi := \nu/\xi$. Hence, if Z is a ν-preconditioner at $x_0, \ldots, x_{j-1}$, we obtain

$$\Phi_A(x_j) \;\le\; \left(1 - \frac{1}{\chi}\right)^j \Phi_A(x_0).$$

It turns out that we can derive a convergence rate stronger than the one obtained above, as the following theorem states.

Theorem 3.2 *Assume in Algorithm PCG that Z is a ν-preconditioner at x_i for all $i = 0, \ldots, j$, and that $\widehat{A} \succeq \xi I$. Further, let us define $\chi := \nu/\xi$. Then,*

$$\Phi_A(x_j) \;\le\; 4\chi \left(\frac{\sqrt{3\chi} - 1}{\sqrt{3\chi} + 1}\right)^{2j} \Phi_A(x_0).$$

The proof of this theorem will be given at the end of this subsection after we present some technical results.

We begin with some important definitions. First, consider the linear operator $\mathcal{B}_j : \widehat{S}_j \mapsto \widehat{S}_j$ defined as

$$\mathcal{B}_j(u) \;:=\; P_{\widehat{S}_j}(\widehat{A}u), \tag{16}$$

where $P_{\widehat{S}_j}$ denotes the orthogonal projection operator from $\mathbb{R}^n$ onto $\widehat{S}_j$. Since for all $u, v \in \widehat{S}_j$,

$$u^T \mathcal{B}_j(v) \;=\; u^T P_{\widehat{S}_j}(\widehat{A}v) \;=\; (P_{\widehat{S}_j} u)^T \widehat{A} v \;=\; u^T \widehat{A} v, \tag{17}$$

and $\widehat{A} \succ 0$, it follows that $\mathcal{B}_j$ is self-adjoint and positive definite, and hence invertible. Next, define the function $\Psi_j : x_0 + S_{j-1} \mapsto \mathbb{R}$ as

$$\Psi_j(x) \;:=\; \frac{1}{2}[Z^T g(x)]^T \mathcal{B}_j^{-1} [Z^T g(x)]. \tag{18}$$

Before continuing, we need to show that $\Psi_j(\cdot)$ is well-defined on the affine space $x_0 + S_{j-1}$, i.e. that $Z^T g(x) \in \widehat{S}_j$ for all $x \in x_0 + S_{j-1}$. In fact, this assertion follows from the inclusion $\hat{g}_0 + \widehat{A}\widehat{S}_{j-1} \subseteq \widehat{S}_j$, which holds in view of (13), and the following technical result.

Lemma 3.3 *Define*

$$\mathcal{P}_j \ := \ \{P_j : P_j \text{ is a polynomial of degree at most } j \text{ such that } P_j(0) = 1\}. \tag{19}$$

Then, the following statements are equivalent:

i) $x \in x_0 + S_{j-1}$*;*

ii) $Z^T g(x) \in \hat{g}_0 + \widehat{A}\widehat{S}_{j-1}$*;*

iii) $Z^T g(x) \in P_j(\widehat{A})\, \hat{g}_0$ *for some* $P_j \in \mathcal{P}_j$*.*

Proof: The equivalence between ii) and iii) is obvious in view of (13) and the definition of $\mathcal{P}_j$. Now, (11), (12), and (14) imply that

$$\begin{aligned} x \in x_0 + S_{j-1} \ &\Leftrightarrow \ x - x_0 \in Z\widehat{S}_{j-1} \Leftrightarrow Z^T A(x - x_0) \in \widehat{A}\widehat{S}_{j-1} \\ &\Leftrightarrow \ Z^T(g(x) - g_0) \in \widehat{A}\widehat{S}_{j-1} \ \Leftrightarrow \ Z^T g(x) \in \hat{g}_0 + \widehat{A}\widehat{S}_{j-1}, \end{aligned}$$

i.e. i) and ii) are equivalent. ∎

The relevance of the function Ψ_j is revealed by the following lemma, which relates the functions $\Phi_A(\cdot)$ and $\Psi_j(\cdot)$ on the space $x_0 + S_{j-1}$.

Lemma 3.4 *Let* $x \in x_0 + S_{j-1}$*, and let* x_{j+1} *be the* $j+1$*-st iterate of the PCG method. Then*

$$\Phi_A(x) - \Phi_A(x_{j+1}) \ = \ \Psi_j(x). \tag{20}$$

Proof: Let $u \in \widehat{S}_j$ be given, and define $v := \mathcal{B}_j^{-1}(u) \in \widehat{S}_j$. By the definition of $\mathcal{B}_j$, $u = \mathcal{B}_j(v) = \widehat{A}v + p$ for some unique vector $p = p(u) \in \widehat{S}_j^{\perp}$. Thus,

$$\widehat{A}^{-1}u \ = \ v + \widehat{A}^{-1}p. \tag{21}$$

Multiplying (21) by p^T and using the facts that $v \in \widehat{S}_j$ and $p \in \widehat{S}_j^{\perp}$, we obtain

$$u^T\widehat{A}^{-1}p \ = \ v^Tp + p^T\widehat{A}^{-1}p \ = \ p^T\widehat{A}^{-1}p. \tag{22}$$

On the other hand, multiplying (21) by u^T and using (22), we see that

$$u^T\widehat{A}^{-1}u \ = \ u^Tv + u^T\widehat{A}^{-1}p \ = \ u^T\mathcal{B}_j^{-1}(u) + u^T\widehat{A}^{-1}p \ = \ u^T\mathcal{B}_j^{-1}(u) + p^T\widehat{A}^{-1}p. \tag{23}$$

Now, let $x \in x_0 + S_{j-1}$ be given. We will show that (23) with $u = Z^T g(x)$ implies (20). Indeed, first note that $Z^T g(x_{j+1}) \in \widehat{S}_j^{\perp}$ in view of (12) and statements (a) and (b) of Proposition 3.1. Moreover, since $x, x_{j+1} \in x_0 + S_j$, it follows from Lemma 3.3 that $Z^T g(x) - Z^T g(x_{j+1}) \in \widehat{A}\widehat{S}_j$. These two observations together imply that if $u = Z^T g(x)$ then $p(u) = Z^T g(x_{j+1})$. The result now follows from equality (23) with $u = Z^T g(x)$ and $p = Z^T g(x_{j+1})$, the definition of Ψ_j and the fact that $\Phi_A(x) = \frac{1}{2}g(x)^T A^{-1} g(x) = \frac{1}{2}(Z^T g(x))^T \widehat{A}^{-1}(Z^T g(x))$ for every $x \in \mathbb{R}^n$. ∎

Lemma 3.5 *Assume in Algorithm PCG that Z is a ν-preconditioner at x_j, and that $Z^TAZ \succeq \xi I$. Then,*

$$\frac{\Phi_A(x_j)}{\Phi_A(x_0)} \leq \chi \frac{\Psi_j(x_j)}{\Psi_j(x_0)},$$

where $\chi := \nu/\xi$.

Proof: Since Z is a ν-preconditioner at x_j, equation (15) holds for $i = j$. By rearranging the terms in (15) and invoking Lemma 3.4 with $x = x_j$, we see that

$$\Phi_A(x_j) \;\leq\; \chi[\Phi_A(x_j) - \Phi_A(x_{j+1})] \;=\; \chi\Psi_j(x_j). \tag{24}$$

Moreover, Lemma 3.4 along with the facts that $x_0 \in x_0 + S_{j-1}$ and $\Phi_A(x_{j+1}) \geq 0$ imply that $\Phi_A(x_0) \geq \Psi_j(x_0)$. Combining this inequality with (24) yields the desired result. ∎

Observe that Theorem 3.2 gives an upper bound on the ratio $\Phi_A(x_j)/\Phi_A(x_0)$. In view of Lemma 3.5, such an upper bound can be obtained by simply developing an upper bound for the ratio $\Psi_j(x_j)/\Psi_j(x_0)$, which will be accomplished in Lemma 3.7 below. First, we establish some bounds on the eigenvalues of $\mathcal{B}_j$ in the following result.

Lemma 3.6 *Assume in Algorithm PCG that Z is a ν-preconditioner at every x_i for $i = 0, \ldots, j$, and that $Z^TAZ \succeq \xi I$. Then, all of the eigenvalues of $\mathcal{B}_j$ lie in the interval $[\xi, 3\nu]$.*

Proof: Since $\mathcal{B}_j$ is self-adjoint, its eigenvalues are all real-valued. To prove the conclusion of the lemma, it suffices to prove that $\xi u^Tu \leq u^T\mathcal{B}_j(u) \leq 3\nu(u^Tu)$ for all $u \in \widehat{S}_j$. However, in view of (17), we have that $u^T\mathcal{B}_ju = u^T\widehat{A}u$ for all $u \in \widehat{S}_j$. Thus, it suffices to prove that

$$\xi u^Tu \;\leq\; u^T\widehat{A}u \;\leq 3\nu(u^Tu) \;\text{ for all } u \in \widehat{S}_j. \tag{25}$$

To that end, let $u \in \widehat{S}_j$ be given. Since $\widehat{A} \succeq \xi I$, we have that $\xi u^Tu \leq u^T\widehat{A}u$, proving the first inequality in (25). Next, by Proposition 3.1(a), there exist $\alpha_0, \ldots, \alpha_j \in \mathbb{R}$ such that $u = \sum_{i=0}^j \alpha_i\hat{g}_i$. Using Proposition 3.1(b), we see that

$$u^Tu \;=\; \left(\sum_{i=0}^{j} \alpha_i\hat{g}_i\right)^T \left(\sum_{i=0}^{j} \alpha_i\hat{g}_i\right) \;=\; \sum_{i=0}^{j} \alpha_i^2\|\hat{g}_i\|^2. \tag{26}$$

The fact that Z is a ν-preconditioner at x_i implies that $\hat{g}_i^T\widehat{A}\hat{g}_i \leq \nu\|\hat{g}_i\|^2$ for $i = 0, \ldots, j$. Using this fact, along with Proposition 3.1(c), the Cauchy-Schwartz inequality, and equation

(26), we have that

$$
\begin{aligned}
u^T \widehat{A} u &= \left(\sum_{i=0}^{j} \alpha_i \hat{g}_i\right)^T \widehat{A} \left(\sum_{i=0}^{j} \alpha_i \hat{g}_i\right) = \sum_{i=0}^{j} \alpha_i^2 \; \hat{g}_i^T \widehat{A} \hat{g}_i + 2 \sum_{0 \le i < l \le j} \alpha_i \alpha_l \; \hat{g}_i^T \widehat{A} \hat{g}_l \\
&= \nu \sum_{i=0}^{j} \alpha_i^2 \hat{g}_i^T \hat{g}_i + 2 \sum_{i=0}^{j-1} \alpha_i \alpha_{i+1} \hat{g}_i^T \widehat{A} \hat{g}_{i+1}, \\
&\le \nu \sum_{i=0}^{j} \alpha_i^2 \|\hat{g}_i\|^2 + 2 \sum_{i=0}^{j-1} |\alpha_i| |\alpha_{i+1}| \sqrt{\hat{g}_i^T \widehat{A} \hat{g}_i} \sqrt{\hat{g}_{i+1}^T \widehat{A} \hat{g}_{i+1}}, \\
&\le \nu \sum_{i=0}^{j} \alpha_i^2 \|\hat{g}_i\|^2 + 2 \sum_{i=0}^{j-1} |\alpha_i| \; |\alpha_{i+1}| \; (\sqrt{\nu} \|\hat{g}_i\|)(\sqrt{\nu} \|\hat{g}_{i+1}\|), \\
&\le \nu \sum_{i=0}^{j} \alpha_i^2 \|\hat{g}_i\|^2 + \sum_{i=0}^{j-1} \left(\nu \alpha_i^2 \|\hat{g}_i\|^2 + \nu \alpha_{i+1}^2 \|\hat{g}_{i+1}\|^2\right), \\
&\le 3\nu \sum_{i=0}^{j} \alpha_i^2 \|\hat{g}_i\|^2 \; = \; 3\nu (u^T u),
\end{aligned}
$$

where in the second to last inequality we use the fact that $2\beta\gamma \le \beta^2 + \gamma^2$ for all β and γ. Thus, the second inequality in (25) holds. ∎

The following lemma provides a bound on the ratio $\Psi_j(x_j)/\Psi_j(x_0)$.

Lemma 3.7 *Assume in Algorithm PCG that $\widehat{A} \succeq \xi I$, and that Z is a ν-preconditioner at every x_i for $i = 0, \ldots, j$. Then,*

$$
\frac{\Psi_j(x_j)}{\Psi_j(x_0)} \; \le \; 4 \left(\frac{\sqrt{3\chi} - 1}{\sqrt{3\chi} + 1} \right)^{2j}.
$$

Proof: Let $\lambda_0, \ldots, \lambda_j$ denote the eigenvalues of $\mathcal{B}_j$. We begin by observing that since $\mathcal{B}_j$ is self-adjoint, it has an orthonormal basis of eigenvectors $v_0, \ldots, v_j$ associated with its eigenvalues $\lambda_0, \ldots, \lambda_j$, which all lie in the interval $[\xi, 3\nu]$ by Lemma 3.6. Using the fact that $Z^T g_0 = \hat{g}_0$ clearly belongs to $\widehat{S}_j$ in view of (12) and (13), we conclude that there exist $\alpha_0, \ldots, \alpha_j \in \mathbb{R}$ such that $Z^T g_0 = \sum_{i=0}^{j} \alpha_i v_i$. By equation (18), we have that

$$
\Psi_j(x_0) \; = \; \frac{1}{2} \left(\sum_{i=0}^{j} \alpha_i v_i\right)^T \mathcal{B}_j^{-1} \left(\sum_{i=0}^{j} \alpha_i v_i\right) \; = \; \frac{1}{2} \left(\sum_{i=0}^{j} \alpha_i v_i\right)^T \left(\sum_{i=0}^{j} \frac{\alpha_i}{\lambda_i} v_i\right) \; = \; \frac{1}{2} \sum_{i=0}^{j} \frac{\alpha_i^2}{\lambda_i}.
$$

Next, let $x \in x_0 + S_{j-1}$ be given. In view of Lemma 3.3, there exists $P_j \in \mathcal{P}_j$ such that $Z^T g(x) = P_j(\widehat{A}) \hat{g}_0$. Using (13) and the fact that $\widehat{A} u = \mathcal{B}_j(u)$ for all $u \in \widehat{S}_j$, we easily see that

$$Z^T g(x) \;=\; P_j(\widehat{A})\hat{g}_0 \;=\; P_j(\mathcal{B}_j)(\hat{g}_0) \;=\; P_j(\mathcal{B}_j)\left(\sum_{i=0}^{j}\alpha_i v_i\right) \;=\; \sum_{i=0}^{j}\alpha_i P_j(\lambda_i) v_i.$$

Thus, in view of (18), we have

$$\begin{aligned}
\Psi_j(x) &= \frac{1}{2}\left(\sum_{i=0}^{j}\alpha_i[P_j(\lambda_i)]v_i\right)^T \mathcal{B}_j^{-1}\left(\sum_{i=0}^{j}\alpha_i[P_j(\lambda_i)](v_i)\right)\\
&= \frac{1}{2}\left(\sum_{i=0}^{j}\alpha_i[P_j(\lambda_i)]v_i\right)^T \left(\sum_{i=0}^{j}\alpha_i[P_j(\lambda_i)]\mathcal{B}_j^{-1}(v_i)\right)\\
&= \frac{1}{2}\left(\sum_{i=0}^{j}\alpha_i[P_j(\lambda_i)]v_i\right)^T \left(\sum_{i=0}^{j}\frac{\alpha_i}{\lambda_i}[P_j(\lambda_i)]v_i\right) \;=\; \frac{1}{2}\sum_{i=0}^{j}\frac{\alpha_i^2}{\lambda_i}[P_j(\lambda_i)]^2\\
&\leq \left\{\max_{i=0,\ldots,j}[P_j(\lambda_i)]^2\right\}\Psi_j(x_0) \;\leq\; \max_{\lambda\in[\xi,3\nu]}[P_j(\lambda)]^2\Psi_j(x_0),
\end{aligned}$$

where the last inequality is due to the fact that $\lambda_i \in [\xi, 3\nu]$ for all $i = 0, \ldots, j$. The last relation together with Lemma 3.3 then imply

$$\min_{x\in x_0+S_{j-1}} \Psi_j(x) \;\leq\; \left[\min_{P_j\in\mathcal{P}_j}\left\{\max_{\lambda\in[\xi,3\nu]}[P_j(\lambda)]^2\right\}\right]\Psi_j(x_0) \;\leq\; 4\left(\frac{\sqrt{3\chi}-1}{\sqrt{3\chi}+1}\right)^{2j}\Psi_j(x_0),$$

where the last inequality is well-known (see for example pages 55-56 of [12]). The result now follows by noting that Proposition 3.1(d) and the fact that, by Lemma 3.4, $\Psi_j(\cdot)$ and $\Phi_A(\cdot)$ differ only by a constant on $x_0 + S_{j-1}$, imply that $\Psi_j(x_j) = \min_{x\in x_0+S_{j-1}}\Psi_j(x)$. ∎

We now note that Theorem 3.2 follows as an immediate consequence of Lemma 3.5 and Lemma 3.7.

4 An Adaptive PCG Algorithm

In this section, we will develop an algorithm which incorporates the adaptive preconditioning scheme from Section 2 into the PCG method. We start by discussing a step of our adaptive PCG (APCG) algorithm. A *step* falls into one of the following three categories: (i) a standard PCG iteration, (ii) an update of the preconditioner matrix Z followed by a one-step backtrack in the PCG algorithm if possible, or (iii) an update of the preconditioner matrix Z followed by a restart of the PCG algorithm. To describe a general step of the APCG algorithm, suppose that we have already generated the jth iteration of the PCG algorithm using Z as a preconditioner. Assume that Z satisfies $Z^TAZ \succeq \xi I$ for some $\xi \in (0, 1]$, and that Z is a ν-preconditioner at the PCG iterates $x_0, \ldots, x_{j-1}$ for some constant $\nu > n$. (In the first step of the APCG algorithm, we assume that $A \succeq I$; hence we may choose $Z = I$ and $\xi = 1$.) We will split our discussion into two cases, depending on whether Z is a ν-preconditioner at x_j.

If Z is a ν-preconditioner at x_j, then we simply perform a PCG iteration with Z as the preconditioner, which corresponds to a step of type (i). Assume from now on that Z is not a ν-preconditioner at x_j. In this case, the step of the APCG algorithm consists of updating Z and ξ, then either backtracking one PCG iteration if possible (i.e., a type (ii) step) or restarting the PCG algorithm with ξ reset to one (i.e., a type (iii) step). More specifically, let $C := \xi^{-1/2}Z$, and note that $\widehat{A} := C^TAC \succeq I$. The facts that Z is not a ν-preconditioner at x_j and that $\xi \leq 1$ imply that

$$g_j^TC(C^TAC)C^Tg_j \;=\; \frac{1}{\xi^2}g_j^TZ(Z^TAZ)Z^Tg_j \;>\; \frac{1}{\xi^2}\nu\|Z^Tg_j\|^2 \;=\; \frac{\nu}{\xi}\|C^Tg_j\|^2 \;\geq\; \nu\|C^Tg_j\|^2.$$

Hence $w := C^Tg_j$ satisfies equation (9); as a result, we may use Theorem 2.6 to generate an update matrix F satisfying properties P1–P3. Next, let $H := F/\mu$, where μ is given by (7). It is important to observe that H takes the following form:

$$H = I + \zeta pp^T,$$

where p is parallel to $\widehat{A}\hat{g}_j$ and ζ is a constant. By Proposition 3.1(c), we have that $p \perp \hat{g}_i$ for all $i \leq j-2$, and as a result, $Hw = w$ for all $w \in \widehat{S}_{j-2}$. Using this fact, it is easy to see that

1. The PCG algorithms corresponding to Z and ZH are completely identical up to step $j-2$ and generate the same iterate x_{j-1}, and
2. ZH is a ν-preconditioner at $x_0, \ldots, x_{j-2}$.

Moreover, the preconditioner ZH satisfies

$$(ZH)^TA(ZH) \;=\; \mu^{-2}(ZF)^TA(ZF) \;=\; \xi\mu^{-2}C^TAC \;\succeq\; \xi\mu^{-2}I.$$

Hence, by performing the updates $Z \leftarrow ZH$ and $\xi \leftarrow \xi\mu^{-2}$, we have that $Z^TAZ \succeq \xi I$. Also, by condition 2 above, if we replace j by $\max\{j-1, 0\}$, we have the conditions assumed at the beginning of this step. The process of replacing Z by ZH, ξ by $\xi\mu^{-2}$, and j by $\max\{j-1,0\}$ is a type (ii) step of the APCG algorithm. Note that a backtrack is possible if and only if $j > 0$.

The above description of a step would be complete were it not for the fact that ξ might get too small, which would adversely affect the rate of convergence of the PCG algorithm in view of Theorem 3.2. To prevent this from occurring, we perform a type (iii) step whenever ξ becomes too small. More specifically, fix a constant $\delta \in (0,1)$. Perform the updates $Z \leftarrow ZH$ and $\xi \leftarrow \xi\mu^{-2}$ as before (in the case where Z is not a ν-preconditioner at the current iterate x_j). Next, check to see whether $\xi > \delta$. If it is, we complete a type (ii) step by replacing $j \leftarrow \max\{j-1, 0\}$ as in the previous paragraph. Otherwise, if $\xi \leq \delta$, we complete a type (iii) step by restarting the PCG algorithm (with $j = 0$) using the last PCG iterate as the starting point with the preconditioner Z updated to $\xi^{-1/2}Z$ and ξ updated to 1 in this exact order. Note that these last two updates preserve the fact that $Z^T AZ \succeq \xi I$ and also prevents ξ from becoming too small by resetting it to one. Note also that if j is set to zero in a type (ii) step, the PCG algorithm clearly restarts, but ξ is not reset to one as is done in a type (iii) step.

We are now ready to state our main algorithm.

Algorithm APCG:
Start: Given $A \succeq I$, $b \in \mathbb{R}^n$, $x_0 \in \mathbb{R}^n$, and constants $\nu > n$, $\delta \in (0,1)$, and $\epsilon > 0$.

1. Set $\Phi_0 = \Phi_A(x_0)$ and $Z = I$.
2. Set $i = 0$, $\xi = 1$, $g_0 = Ax_0 - b$, $d_{-1} = 0$, $\beta_0 = 0$, and $\gamma_0 = \|Z^T g_0\|^2$.
3. **While** $\Phi_A(x_i) > \epsilon\Phi_0$ **do**
   (a) **While** $g_i^T Z(Z^T AZ)Z^T g_i \; > \; \nu\gamma_i$ **do**
      i. Build a matrix F per Theorem 2.6 with $w := \xi^{-1/2}Z^T g_i$ and $\widehat{A} := \xi^{-1}Z^T AZ$
      ii. Set $Z = ZF/\mu$ and $\xi = \xi\mu^{-2}$, where μ is given by (7)
      iii. **If** $\xi \le \delta$ **then** go to Step 2 with $Z := \xi^{-1/2}Z$ and $x_0 := x_i$
      iv. Set $i = \max\{i-1, 0\}$
      end (while)
   (b) $d_i = -ZZ^T g_i + \beta_i d_{i-1}$, where $\beta_i = \gamma_i/\gamma_{i-1}$
   (c) $x_{i+1} = x_i + \alpha_i d_i$, where $\alpha_i = \gamma_i/(d_i^T Ad_i)$
   (d) $g_{i+1} = g_i + \alpha_i Ad_i$
   (e) $\gamma_{i+1} = \|Z^T g_{i+1}\|^2$
   (f) Set $i = i+1$

   end (while).

Note that in the above algorithm, x_i may denote several ith iterates of the PCG method, since the latter may be restarted several times during the course of Algorithm APCG. We now present the main convergence result we have obtained for Algorithm APCG, which shows that an ϵ-solution to $Ax = b$ can be obtained in $\mathcal{O}(N_\psi + \sqrt{n}\log\epsilon^{-1})$ steps.

Theorem 4.1 *Assume that the starting conditions of Algorithm APCG are met, and that $\nu = \mathcal{O}(n)$ and $\max\{(1-\delta)^{-1}, \delta^{-1}\} = \mathcal{O}(1)$. Then, Algorithm APCG generates a point x_i satisfying $\Phi_A(x_i) \le \epsilon\Phi_0$ in*

$$\mathcal{O}\left(N_\psi + \sqrt{n}\log\epsilon^{-1}\right)$$

steps, where N_ψ is defined in Theorem 2.2(a).

Proof: For the purposes of this proof, we say that a *cycle* begins whenever step 2 of Algorithm APCG occurs, or equivalently, whenever ξ is reset to one, and that a cycle ends whenever a new one begins. Consider any iterate x_i in Algorithm APCG such that $\Phi_A(x_i) > \epsilon\Phi_0$, and let l denote the cycle number in which this iterate occurs. Also, for any cycle $r < l$, let i_r and y_r denote the last PCG iterate number and last PCG iterate, respectively, of that cycle. Finally, we define

$$t \;:= i + \sum_{r=1}^{l-1} i_r.$$

(The index t denotes the "current" PCG iterate number, if the iterate count is not reset to 0 when a restart occurs.)

Recall that at the beginning of this section, we divided the steps in Algorithm APCG into types (i)–(iii). Our objective is to bound the number of these steps required to get to iterate x_i. Let us first consider the number of type (ii) and (iii) steps. If we define $C := \xi^{-1/2}Z$, it is clear that $C = F_1 \cdots F_k$, where the matrices F_j, $j = 1, \ldots, k$, are the ones obtained via Theorem 2.6. Thus, an argument similar to the one given in Theorem 2.2(a) can be used to show that the number of updates is bounded by N_ψ. Since each type (ii) and (iii) step requires that an update be performed, the number of type (ii) and (iii) steps is bounded by N_ψ.

Let us now consider the number of type (i) steps required to get to iterate x_i. We observe that when a type (ii) step occurs, one PCG iteration may be lost; thus, the total number of type (i) steps cannot exceed t plus the number of type (ii) steps. Hence, we have the following bound on the total number of steps:

$$\text{Total steps} \;\le\; t + (\text{number of type (ii) steps}) + N_\psi \;=\; t + \mathcal{O}(N_\psi). \tag{27}$$

It remains for us to determine a valid bound on t. To that end, let us examine the performance of the PCG iterates within a given cycle. In particular, consider the final step of the first cycle; it is clear that the current PCG iterate for this step is $y_1 = x_{i_1}$. At the beginning of this step, we have a preconditioner Z which is a ν-preconditioner at $x_0, \ldots, x_{(i_1-1)}$ and which satisfies $Z^TAZ \succeq \xi I$. Hence, we apply Theorem 3.2 with $j = i_1 - 1$ and use the fact that $\Phi_A(x_{i_1}) \le \Phi_A(x_{(i_1-1)})$ by Proposition 3.1(d) to obtain

$$\Phi_A(y_1) \;=\; \Phi_A(x_{i_1}) \;\le\; \Phi_A(x_{(i_1-1)}) \;\le\; 4\chi\left(1 - \frac{2}{\sqrt{3\chi}+1}\right)^{2(i_1-1)} \Phi_0.$$

Now y_1 also serves as the starting iterate for the second cycle; as a result, we have that

$$\Phi_A(y_2) \;\le\; 4\chi\left(1 - \frac{2}{\sqrt{3\chi}+1}\right)^{2(i_2-1)} \Phi_A(y_1) \;\le\; (4\chi)^2\left(1 - \frac{2}{\sqrt{3\chi}+1}\right)^{2(i_1+i_2-2)} \Phi_0.$$

By induction, it follows that

$$\Phi_A(x_i) \;\le\; (4\chi)^l\left(1 - \frac{2}{\sqrt{3\chi}+1}\right)^{2t-2l} \Phi_0.$$

We use this result along with the fact that $\Phi_A(x_i) > \epsilon\Phi_0$ to observe that

$$\epsilon \;<\; (4\chi)^l \left(1 - \frac{2}{\sqrt{3\chi}+1}\right)^{2t-2l} \;\leq\; (4\chi)^l \exp\left\{\frac{-4t+4l}{\sqrt{3\chi}+1}\right\}.$$

By taking logarithms on both sides of this inequality, we obtain

$$t \;<\; \frac{1}{4}\left[\left(\sqrt{3\chi}+1\right)\left(l\log(4\chi)+\log\epsilon^{-1}\right)\right] + l \;=\; \mathcal{O}\left(l\sqrt{\chi}\log\chi + \sqrt{\chi}\log\epsilon^{-1}\right). \tag{28}$$

We will now show that $l = \mathcal{O}(N_\psi/n)$. Observe that since $(1-\delta)^{-1} = \mathcal{O}(1)$, we have that $\log\delta^{-1} \geq \omega$ for some constant $\omega > 0$. Suppose that since the beginning of a cycle, we have performed k updates on the preconditioner Z, and assume that $k < \omega(n-1)$. It follows that $k < (n-1)\log\delta^{-1}$, and by rearranging terms, we have that $\delta < \exp\{-k/(n-1)\}$. However, notice that in step 3(a)ii of Algorithm APCG, we have by (7) that

$$\mu^2 \;=\; \frac{n-\theta^2}{n-1} \;\leq\; \frac{n}{n-1} \;=\; 1 + \frac{1}{n-1} \;\leq\; \exp\left\{\frac{1}{(n-1)}\right\},$$

i.e., $\mu^{-2} \geq \exp\{-1/(n-1)\}$. Hence, our current $\xi \geq \exp\{-k/(n-1)\}$, which implies that $\xi > \delta$. Thus, we can still perform another update within the same cycle. This shows that the number of updates in a complete cycle is $\geq \omega(n-1)$, which implies that $l \leq N_\psi/[\omega(n-1)] + 1 = \mathcal{O}(N_\psi/n)$.

To obtain a bound on χ, we observe that at each type (i) step, $\xi > \delta$. In view of the assumptions that $\delta^{-1} = \mathcal{O}(1)$ and $\nu = \mathcal{O}(n)$, it follows that $\chi = \nu\xi^{-1} < \nu\delta^{-1} = \mathcal{O}(n)$. We incorporate the bounds on l and χ into (28) to conclude that

$$t \;=\; \mathcal{O}\left(\frac{N_\psi \log n}{\sqrt{n}} + \sqrt{n}\log\epsilon^{-1}\right) \;=\; \mathcal{O}(N_\psi + \sqrt{n}\log\epsilon^{-1}). \tag{29}$$

The result follows by incorporating this bound into (27). ∎

It is important to observe that by using analysis similar to Lemma 2.3, it can be shown that Algorithm APCG is also a polynomial-time algorithm under the same assumptions as those given in that lemma.

We conclude the section by discussing some computational aspects of Algorithm APCG. It is important to note that if step 3(a) occurs repeatedly, we may find ourselves regressing through the PCG iterates. As a result, we need to either keep all of the PCG data in memory or determine a way to recreate the data as needed. When lack of memory is an issue, the latter option is the only viable alternative. In such a case, it is easy to see that all of the iterates and search directions generated by the PCG algorithm can be recreated by only storing the constants β_i in memory and by simply reversing the PCG algorithm.

5 Concluding Remarks

It is well-known that under exact arithmetic, the standard CG algorithm terminates in at most n iterations. However, in finite-precision arithmetic, the standard CG algorithm loses this property and instead possesses an iteration-complexity bound of $\mathcal{O}(\sqrt{\kappa(A)}\log \epsilon^{-1})$. Our algorithm also loses its theoretical properties under finite arithmetic; indeed, Lemma 3.6 relies heavily on statements (b) and (c) of Proposition 3.1, which only hold under exact arithmetic. Nevertheless, one may hope to gain significant reductions in the number of CG iterations using our algorithm in finite-precision arithmetic, since the update matrices F_j have the effect of making the preconditioned matrix Z^TAZ better conditioned as the algorithm progresses.

One important assumption in our algorithm is the requirement that $A \succeq I$. It is possible to ensure that this assumption holds for matrices for which $A \succ 0$, simply by premultiplying A by some large positive constant $\omega \geq (\lambda_{\min}(A))^{-1}$. In a future paper, we wish to relax the requirement that $A \succeq I$, as well as provide computational results measuring the performance of our approach.

References

[1] D.P. Bertsekas. *Nonlinear Programming.* Athena, 2nd edition, 1995.

[2] G.H. Golub and D.P. O'Leary. Some history of the conjugate gradient and lanczos algorithms: 1948-1976. *SIAM Review*, 31:50–102, 1989.

[3] A. Greenbaum. *Iterative Methods for Solving Linear Systems.* SIAM, 1997.

[4] M.R. Hestenes and E. Stiefel. Methods of conjugate gradients for solving linear systems. *Journal of Research of the National Bureau of Standards*, 49:409–436, 1952.

[5] L.G. Khachiyan. A polynomial algorithm in linear programming. *Doklady Akademia Nauk SSSR*, 244:1093–1096, 1979.

[6] D.G. Luenberger. *Linear and Nonlinear Programming.* Addison-Wesley, 2nd edition, 1984.

[7] R.D.C. Monteiro and J.W. O'Neal. Convergence analysis of a long-step primal-dual infeasible interior-point lp algorithm based on iterative linear solvers. Technical report, Georgia Institute of Technology, 2003. Submitted to *Mathematical Programming.*

[8] A.S. Nemirovski and D.B. Yudin. Optimization methods adapting to the "significant" dimension of the problem. *Automatica i Telemekhanika*, 38:75–87, 1977.

[9] M.G.C. Resende and G. Veiga. An implementation of the dual affine scaling algorithm for minimum cost flow on bipartite uncapacitated networks. *SIAM Journal on Optimization*, 3:516–537, 1993.

[10] Y. Saad. *Iterative Methods for Sparse Linear Systems.* SIAM, 2003.

[11] A. Schrijver. *Theory of Linear and Integer Programming.* Wiley, 1986.

[12] J.R. Shewchuk. An introduction to the conjugate gradient method without the agonizing pain. Technical report, Carnegie Mellon University, 1994.

[13] N.Z. Shor. *Minimization Methods for Non-Differentiable Functions.* Springer-Verlag, 1985.

[14] N.Z. Shor. *Nondifferentiable Optimization and Polynomial Problems.* Kluwer, 1998.

[15] M.J. Todd. On minimum volume ellipsoids containing part of a given ellipsoid. *Mathematics of Operations Research*, 7:253–261, 1982.

[16] L.N. Trefethen and D. Bau III. *Numerical Linear Algebra.* SIAM, 1997.