SHDQP: AN ALGORITHM FOR CONVEX SET INTERSECTION PROBLEMS BASED ON SUPPORTING HYPERPLANES AND DUAL QUADRATIC PROGRAMMING

C.H. JEFFREY PANG

Abstract. This paper focuses on algorithms for two problems: Finding a point in the intersection of a finite number of closed convex sets, and projecting onto the intersection of a finite number of closed convex sets. We assume that the projections onto each of these closed convex sets are relatively easy to perform. This assumption is also made for projection methods that solve the two problems, namely the method of alternating projections and Dykstra’s algorithm. In an earlier paper [Pan12], we discussed theoretical issues of the insight of projecting onto the intersection of halfspaces generated by the projection process using quadratic programming to accelerate convergence for the two problems. It turns out that the dual quadratic programming algorithm of Goldfarb and Idnani [GI83] is particular suited for projecting onto the polyhedra generated, because it solves the quadratic programs from warm start solutions whenever new constraints are added. The quadratic programs need not be solved to optimality. Reflection operations on the polyhedron generated so far for the first problem are also explored. We describe and show some outputs for our Matlab function SHDQP (Supporting Hyperplanes and Dual Quadratic Programming). We present numerical results for some simple and hard problems. The simple problems achieve superlinear convergence, which is better than the linear convergence typically associated with projection algorithms.

CONTENTS

1. Introduction  2
2. Proposed algorithm  4
3. Description and usage of SHDQP  12
4. Testing SIP algorithms with the DNN cone  15
5. Ill-conditioning in the constraints  16
6. Projecting onto the DNN cone  18
7. Conclusion  20
8. Acknowledgments  22
References  22

Date: May 7, 2014.
2010 Mathematics Subject Classification. 65K05, 65-05, 90C20, 90C30, 90C59, 47J25, 47A46, 47A50, 52A20, 41A50, 41A65, 46C05.

Key words and phrases. best approximation problem, alternating projections, Dykstra’s algorithm, supporting hyperplanes, quadratic programming, active set.
1. Introduction

For finitely many closed convex sets \( K_1, \ldots, K_r \) in a Hilbert space \( X \), the Set Intersection Problem (SIP) is stated as:

\[
\text{(SIP)} \quad \text{Find an } x \in K := \bigcap_{l=1}^{r} K_l, \text{ where } K \neq \emptyset,
\]

A popular method of solving the SIP is the Method of Alternating Projections (MAP), where one iteratively projects a point through the sets \( K_l \) to find a point in \( K \). Another problem related to the SIP is the Best Approximation Problem (BAP): Find \( P_K(y) \), the closest point to \( y \) in \( K \), that is,

\[
\text{(BAP)} \quad \text{Find } P_K(y) := \arg\min_{x \in K} \| y - x \|^2,
\]

where \( K := \bigcap_{l=1}^{r} K_l \).

for closed convex sets \( K_l, l = 1, \ldots, r \).

We quote [Deu01a], where it is mentioned that the MAP has found application in at least ten different areas of mathematics, which include: (1) solving linear equations; (2) the Dirichlet problem which has in turn inspired the “domain decomposition” industry; (3) probability and statistics; (4) computing Bergman kernels; (5) approximating multivariate functions by sums of univariate ones; (6) least change secant updates; (7) multigrid methods; (8) conformal mapping; (9) image restoration; (10) computed tomography. See also [Deu95] for more information. Numerous other applications are also mentioned in [ER11].

One can easily construct an example in \( \mathbb{R}^2 \) involving a circle and a line such that the MAP converges to a point in \( K \) that is not \( P_K(y) \). Fortunately, Dykstra’s algorithm [Dyk83, BD86] reduces the problem of finding the projection onto \( K \) to the problem of projecting onto \( K_l \) individually by adding correction vectors after each iteration. It was rediscovered in [Han88] using mathematical programming duality. For more on the background and recent developments of the MAP and its variants, we refer the reader to [BB96, BR09, ER11], as well as [Deu01b, Chapter 9] and [BZ05, Subsubsection 4.5.4].

Supporting hyperplanes and Quadratic Programming. The projection process generates halfspaces that contain the \( K \) in (1.1) and (1.2), and therefore outer estimates of \( K \). The polyhedron formed by intersecting these halfspaces is also an outer estimate of \( K \). One can make use of quadratic programming (QP) to project onto this polyhedron. We refer to this strategy as the SHQP strategy. In [Pan12], we laid some of the foundations for solving the SIP and the BAP using SHQP. Figure 1.1 illustrates why the SHQP strategy is promising for solving the SIP and BAP.

Parts of this idea had been studied earlier. In [Pie84], Pierra suggested an extrapolation for the SIP by projecting onto a polyhedron produced by two projections. In [CC11], cutters were defined based on the property that the halfspaces generated contain the intersection of the sets, and studied as a generalization to the projection operation. García-Palomares [GP98, GP01] studied the SIP using the SHQP strategy, but limited his analysis to sets of the form \( K_l = \{ x \mid f_l(x) \leq 0 \} \), where each \( f_l : \mathbb{R}^n \to \mathbb{R} \) is smooth.
Figure 1.1. The method of alternating projections on two convex sets $K_1$ and $K_2$ in $\mathbb{R}^2$ with starting iterate $x_0$ arrives at $x_3$ in three iterations. But the point $x_4$, generated by projecting $x_2$ onto the polyhedron formed by the supporting halfspaces of $K_1$ at $x_1$ and $K_2$ at $x_2$, is much closer to the point $\bar{x}$ than $x_3$. On the other hand, the point $x_3$ is ruled out by the supporting hyperplane of $K_2$ passing through $x_2$.

1.1. Contributions of this paper. Consider first the BAP (1.2). The QPs created to solve the BAP have a particular structure. The objective function stays the same throughout because we always project from $y$ onto some outer polyhedral estimate of $K$. New constraints are added as the projection process continues to obtain better outer polyhedral estimates of $K$. One of the contributions of this paper is to notice that one need not solve these QPs from scratch each time new constraints are added, and can instead use Goldfarb and Idnani’s [GI83] dual active set QP algorithm. The dual QP algorithm can take in the solution of the old QP as a warmstart solution to solve the new QP derived from adding the new constraints.

We examine the inner workings of the dual QP algorithm of [GI83] and identify what we call an inner GI step (Definition 2.2). To solve a newer QP with added constraints from a warmstart solution of an older QP, one can perform the necessary number of inner GI steps to get the optimal solution of the newer QP. The number of inner GI steps needed is always finite, though it can be large. In our algorithm design, the number of inner GI steps taken is flexible. If the inner GI steps do not give much progress, one can choose to project onto the sets $K_l$ to obtain new supporting halfspaces to add to the QP instead.

Some of the classical tools for solving the SIP are projections and reflections, including the method of alternating projections and the Douglas-Rachford algorithm, in part due to the convergence guarantees given by Fejér monotonicity. These ideas are incorporated to our design of our algorithm for the SIP. We call our algorithm SHDQP (Supporting Halfspaces and Dual Quadratic Programming).

A flowchart describing the SHDQP algorithm is presented in Figure 1.2 along with a flowchart describing how an inner GI step is used in the dual QP algorithm of [GI83]. The SHDQP algorithm is described in Section 2. We address infeasibility detection in Subsection 3.2.2.

Next, we describe how SHDQP is implemented in Matlab. We acknowledge [Gas09] for the implementation of the dual QP algorithm in [GI83] in C++. We describe the inputs and outputs of SHDQP in Section 3. We discuss how to choose parameters to handle infeasibility and ill-conditioning of the input problems in Section 5.
We perform tests on our algorithm on some simple and hard SIP and BAP problems. Simple tests for the SIP and BAP involving the intersection of two circles in Section 3 give superlinear convergence, which is better than the linear convergence typically associated with alternating projection algorithms. On some harder test problems involving the DNN cone, the algorithm performs well for the SIP. The algorithm does not perform as well for the BAP. We also test the parameters introduced in Section 5 to handle infeasibility and ill-conditioning in Subsection 6.3.

2. Proposed algorithm

In this section, we propose algorithms for the SIP (1.1) and BAP (1.2) by modifying the dual QP algorithm of Goldfarb and Idnani [GI83]. We shall refer to their algorithm as the GI algorithm to simplify future discussions.

Since the case where the Hessian is the identity matrix is the only case of interest to us, we shall only discuss this case. (The general GI algorithm can work with any positive definite Hessian.) We now state the notations for the QPs we use throughout.

**Definition 2.1.** (Notation for QP) Let \( A \subset \mathbb{N} = \{1, 2, 3, \ldots\} \). Consider the following QP:

\[
QP(A, y) : \min_{x \in \mathbb{R}^n} q(x) := \frac{1}{2} \|y - x\|^2 \\
\text{s.t. } c_i^T x - b_i \geq 0 \text{ for all } i \in A,
\]

where \( c_i \in \mathbb{R}^n \) and \( b_i \in \mathbb{R} \) for all \( i \in \mathbb{N} \). Let the feasible set \( F(A) \) be

\[
F(A) := \{ x \in \mathbb{R}^n | c_i^T x - b_i \geq 0 \text{ for all } i \in A \}.
\]

In other words, \( F(A) \) is the polyhedron formed by intersecting the halfspaces indexed by \( A \).
We say that \((x, A)\) is an S-pair if \(x\) is the unique solution of \(QP(A, y)\) and \(c_i^T x - b_i = 0\) for all \(i \in A\). (In other words, \(A\) is the active set.)

We remark that if \(x\) is the solution of \(QP(A, y)\), then if \(J \subset \mathbb{N}\) is such that \(A \subset J\) and \(c_j^T x - b_j \geq 0\) for all \(j \in J\), then \(x\) is the unique solution of \(QP(J, y)\), with \(A\) as the set of active indices.

The GI algorithm uses an inner step, which we call the inner GI step, and is indicated as follows:

**Definition 2.2.** (Inner GI step) For QP problem \(QP\{1, \ldots, m\}, y\) in (2.1), the Inner GI step in the GI algorithm takes in the following inputs and gives the following outputs.

**Inputs:** An S-pair \((x, A)\), and \(i' \notin A\) such that \(c_i^T x - b_i < 0\).

**Outputs:** An S-pair \((x', A')\) such that \(A' \subset A \cup \{i'\}\) and \(q(x') > q(x)\).

The KKT multipliers \(u_i \geq 0\) for \(i \in A\) such that \(y - x + \sum_{i \in A} u_i c_i = 0\) are also updated.

This inner GI step is step 2 in the description in [GI83]. The infeasibility of the QP is reflected by \(q(x') = \infty\), and the algorithm can end. We will talk more about how infeasibility is detected in Subsection 3.2.2. When there are only a finite number of constraints, say \(m\), we have \(A \subset \{1, \ldots, m\}\). The GI algorithm can be described, in high level terms, as follows:

**Algorithm 2.3.** [GI83] (GI algorithm) This algorithm finds the minimizer of \(QP\{1, \ldots, m\}, y\) in (2.1).

01 Set \(A = \emptyset\) and \(x = y\) so that \((x, A)\) is an S-pair.
02 While \(A\) is not optimal
03 Find \(j' \in \{1, \ldots, m\}\) such that \(c_j^T x - b_j < 0\)
04 Use the inner GI step to find an S-pair \((x', A')\) such that \(q(x') > q(x)\).
05 Let \((x, A) \leftarrow (x', A')\) and update the KKT multipliers \(u_j \geq 0\) for \(j \in A\).
06 end While.
07 Return \(x, A\) and \(u_i\) for \(i \in A\).

It was proved in [GI83] that for a QP with finitely many constraints, the GI algorithm has to terminate. (The value \(q(\cdot)\) is strictly increasing, and the active set \(A \subset \{1, \ldots, m\}\) can take only a finite number of possibilities.) In the GI algorithm, the coefficients \(u_j\) for \(j \in A\) are essential for the running of the algorithm because they help to decide whether a constraint is to be dropped from or added to \(A\). The description in [GI83] also includes some discussion on how to maintain the QR factorization of the matrix formed by concatenating the columns of \(c_i\), where \(i \in A\), in a stable manner using Given’s rotations, and how to update \(u_i\), but we shall suppress these details to highlight only the main ideas related to our algorithms for the BAP and SIP.

In order to simplify the presentation of many of the algorithms in this paper, we say that a halfspace \(H = \{x \in \mathbb{R}^n \mid c^T x - b \geq 0\}\), where \(c \in \mathbb{R}^n\) and \(b \in \mathbb{R}\), supports a (closed convex) set \(K' \subset \mathbb{R}^n\) if \(K' \subset H\), and the boundary of \(H\), denoted as \(\partial H\), satisfies \(\partial H \cap K' = \emptyset\).

### 2.1. The Best Approximation Problem (1.2)

We now discuss how to amend Algorithm 2.3 (the GI algorithm) to solve the BAP (1.2).

**Algorithm 2.4.** (SHDQP: BAP) Let \(\{K_i\}_{i=1}^r\) be \(r\) closed convex sets in \(\mathbb{R}^n\), and let \(K = \bigcap_{i=1}^r K_i\). The algorithm below attempts to find the projection of a point \(y\) onto \(K\). Let \(R : \mathbb{N} \to \{1, \ldots, r\}\) be such that the \(m\)th projection is performed onto \(K_{R(m)}\) as described below:
01 Set $A = \emptyset$ and $x = y$ so that $(x, A)$ is an S-pair, and $m = 0$
02 While stopping conditions not satisfied
03 \quad $m \leftarrow m + 1$
04 \quad If $x \notin K_{R(m)}$, then
05 \quad \quad Project $x$ onto $K_{R(m)}$ to get $c_m \in \mathbb{R}^n$ and $b_m \in \mathbb{R}$ such that
06 \quad \quad $\{\tilde{x} \mid c_m^T \tilde{x} - b_m \geq 0\}$ supports $K_{R(m)}$.
07 \quad \quad While $x$ is not optimal enough for $QP$($\{1, \ldots, m\}, y$)
08 \quad \quad \quad Find $v^l \in \{1, \ldots, m\}$ such that $c_{v^l}^T x - b_{v^l} < 0$.
09 \quad \quad \quad Use inner GI step to find an S-pair $(x', A')$ such that $q(x') > q(x)$.
10 \quad \quad \quad Let $(x, A) \leftarrow (x', A')$ and update KKT multipliers $u_i \geq 0$ for $i \in A$.
11 \quad \quad end While.
12 \quad end If.
13 end While.
14 Return $x, A$ and $u_i$ for $i \in A$.

2.1.1. Stopping conditions. We now describe stopping conditions for Algorithm 2.4

Notice that under the Slater conditions, the KKT conditions for the optimality of $x$ is that

$$ x \in K = \bigcap_{l=1}^{r} K_l $$

(2.3a)

$$ y - x = \sum_{l=1}^{r} \tilde{u}_l v_l, \text{ where } v_l \in N_{K_l}(x) $$

(2.3b)

$$ \tilde{u}_l \geq 0 \text{ for all } l \in \{1, \ldots, r\} $$

(2.3c)

The analogue of Condition (2.3c), $u_i \geq 0$, is guaranteed throughout Algorithm 2.4

Algorithm 2.4 also ensures that $y - x = \sum_{i \in A} u_i (-c_i)$, which approximates (2.3b). Therefore, a reasonable stopping condition is that the iterate $x$ approximately satisfies (2.3a). In other words, $x$ is sufficiently close to each of the $K_l$ for $l \in \{1, \ldots, r\}$. (Though we add that a more formal study of such issues involves the study of linear regularity. See for example [BB96, Kru06].)

It is also possible that $x \in K_l$ for all $l \in \{1, \ldots, r\}$. This would mean that $x \in K$, and $x$ is optimal. This stopping condition can be implemented in Algorithm 2.4

Notice that $x$ does not need to solve $QP$($\{1, \ldots, m\}, y$) to optimality in the inner while loop. In fact, one might not want to solve $QP$($\{1, \ldots, m\}, y$) to optimality if the inner GI steps do not bring about much improvement to $q(\cdot)$. After projecting $x$ onto $K_{R(m)}$, we can get a new halfspace containing $K$ that can bring about more movement of $x$ towards $P_K(y)$.

2.2. The Set Intersection Problem (1.1). We discuss some classical strategies for the SIP (1.1) before presenting our algorithm for the SIP in Algorithm 2.10 where we incorporate some of these classical ideas.

The method of alternating projections is a popular method for solving the SIP. We recall Fejér monotonicity, a useful tool for analyzing alternating projections.

**Definition 2.5.** (Fejér attraction and monotonicity) Let $C \subset \mathbb{R}^n$ be a closed convex set and let $\{x_i\}$ be a sequence in $\mathbb{R}^n$. For $x, y \in \mathbb{R}^n$, we say that $(x, y)$ is a Fejér attraction with respect to $C$ if

$$ ||y - c|| \leq ||x - c|| \text{ for all } c \in C, $$
and a sequence \( \{x_i\}_{i=1}^{\infty} \) is \textit{Fejér monotone} with respect to \( C \) if \( (x_i, x_{i+1}) \) are Fejér attractions with respect to \( C \), i.e.,

\[
\|x_{i+1} - c\| \leq \|x_i - c\| \text{ for all } c \in C \text{ and } i = 1, 2, \ldots
\]

A tool for guaranteeing a Fejér attraction is stated below.

**Theorem 2.6.** (Fejér attraction property) For a closed convex set \( C \subset \mathbb{R}^n \), \( x \in \mathbb{R}^n \), \( \lambda \in [0, 2] \), and the projection \( P_C(\cdot) \) onto \( C \), let the relaxation operator \( R_{C, \lambda} : \mathbb{R}^n \rightarrow \mathbb{R}^n \) be defined by

\[
R_{C, \lambda}(x) = x + \lambda(P_C(x) - x).
\]

Then

\[
\|R_{C, \lambda}(x) - c\|^2 \leq \|x - c\|^2 - \lambda(2 - \lambda)d(x, C)^2 \text{ for all } y \in C.
\]

In particular, \( (x, R_{C, \lambda}(x)) \) is a Fejér attraction with respect to \( C \).

We call \( R_C(\cdot) := R_{C, 2}(\cdot) \) a reflection, since it reflects its input argument about the supporting hyperplane produced by the projection process. The method of alternating projections and its variants are recalled below.

**Algorithm 2.7.** (Alternating projections and variants) For \( K_1, \ldots, K_r \) that are closed convex sets in a Hilbert space \( X \) and starting iterate \( y_0 \in X \), the method of alternating projections can be stated as

\[
y_{i+1} = P_{K_1+i}(y_i) \text{ for } i = 0, 1, 2, \ldots,
\]

where \([i] \) is the integer in \( \{1, \ldots, r\} \) such that \( i - [i] \) is divisible by \( r \). We also define an alternating reflection scheme by

\[
y_{i+1} = R_{K_1+i}(y_i) \text{ for } i = 0, 1, 2, \ldots
\]

If \( r = 2 \), the Douglas-Rachford algorithm is defined by

\[
y_{i+1} = R_{K_1}(y_i),
y_{i+2} = \frac{1}{2}[R_{K_2}(y_{i+1}) + y_i] \text{ for } i = 0, 2, 4, \ldots
\]

In other words, the even iterates of the Douglas-Rachford algorithm can be defined by \( y_{i+2} = \frac{1}{2}[R_{K_2} \circ R_{K_1}(y_i) + y_i] \) for \( i = 0, 2, 4, \ldots \).

The convergence of the alternating projection method to a point in \( K \), if \( K \neq \emptyset \), is well-studied. See for example the discussion in [BZ05, Section 4.5]. If \( K_1 \) and \( K_2 \) are closed convex sets and \( K_1 \cap K_2 \neq \emptyset \), then the Douglas-Rachford algorithm converges weakly to a point in \( K_1 \cap K_2 \). (The basic Douglas-Rachford algorithm originates in [DR56] and convergence was proven as part of [LM79].)

If we can find a sequence of points \( \{y_k\}_{k=0}^{\infty} \) such that \( y_0 = y \), a sequence of polyhedra \( \{F_k\}_{k=1}^{\infty} \) such that \( F_k \supset K \) for all \( k \) and a sequence of numbers \( \{\lambda_k\}_{k=1}^{\infty} \subset [0, 2] \) such that \( y_k = R_{F_k, \lambda_k}(y_{k-1}) \), then the sequence of points \( \{y_k\} \) is a Fejér monotone sequence with respect to \( K \). If the polyhedra \( F_k \) are good approximates of \( K \), then \( y_k \) can be proven to converge to some point in \( K \).

In view of these observations, we propose amendments for the SIP. But first we write down a framework for general SIP algorithms.

**Algorithm 2.8.** (General SIP algorithms) For closed convex sets \( K_1, \ldots, K_r \) in a Hilbert space \( X \) and starting iterate \( y_0 \in X \), we present the general framework for SIP algorithms that include the the method of alternating projections. Let \( \mathcal{R} : \mathbb{N} \rightarrow \{1, \ldots, r\} \) be such that the \( m \)-th projection is onto \( K_{\mathcal{R}(m)} \), as described below:
01 Let \( y = y_0, m = 0 \).
02 While \( y \) not close enough to \( K \)
03 \[ m \leftarrow m + 1 \]
04 If \( y \notin K_{R(m)} \), then
05 Project \( y \) onto \( K_{R(m)} \) to get \( c_m \in \mathbb{R}^n \) and \( b_m \in \mathbb{R} \)
06 such that \( \{ \tilde{x} \mid c_m^T \tilde{x} - b_m \geq 0 \} \) supports \( K_{R(m)} \).
07 Choose \( A \subseteq \{ 1, \ldots, m \} \) and let \( x = P_{F(A)}(y) \) for \( F(A) \) defined in \((2.2)\).
08 Choose \( \lambda \in \{ 0 \} \cup [1, 2] \) and update \( y \leftarrow y + \lambda (x - y) \).
09 end if
10 end While.

We remark that Algorithm \((2.8)\) is a generalization of the method of alternating projections: If \( A = \{ m \} \) and \( \lambda = 1 \), then Algorithm \((2.8)\) reduces to the method of alternating projections. If \( A = \{ m \} \) and \( \lambda = 2 \), then Algorithm \((2.8)\) reduces to the alternating reflection scheme. If \( A \neq \{ m \} \), then one would require a QP algorithm to carry out Algorithm \((2.8)\). The constraints that are the most logical to be introduced in the inner GI step are the ones formed by the most recent projections.

**Remark 2.9.** (Choice of \( \lambda \) in Algorithm \((2.8)\)) We discuss some typical choices of \( \lambda \) in line 8 of Algorithm \((2.8)\). If \( \lambda \) was chosen to be zero, then we are electing to stay with the current point. A reason to stay at the current point \( y \) is because the supporting hyperplanes were generated by projecting points close to \( y \), and it may be better to explore this neighborhood before moving to a new point. (If no new supporting hyperplanes were created from the projection process, then do not move.) If \( \lambda \) is chosen to be 1, then the algorithm is seeking the greatest guaranteed decrease in the distance to \( K \) in view of \((2.4)\). When \( \lambda \) is chosen to be in \([1, 2)\), the algorithm is reflecting (fully when \( \lambda = 2 \), and partly when \( \lambda \in (1, 2) \)) about the supporting hyperplane formed by projecting onto the polyhedron defined by the indices \( A \). Such a reflection chooses exploration of the space rather than pursuing the greatest decrease in distance to \( K \).

In view of the dual QP algorithm, we propose the following algorithm for the SIP.

**Algorithm 2.10.** (SHDQP: SIP) Let \( \{ K_i \}_{i=1}^r \) be \( r \) closed convex sets in \( \mathbb{R}^n \), and let \( K = \cap_{i=1}^r K_i \). The algorithm below attempts to find a point in \( K \) from a starting point \( y \). Let \( R : \mathbb{N} \to \{ 1, \ldots, r \} \) be such that the \( m \)th projection is performed onto \( K_{R(m)} \) as described below:
01 Set \( A = \emptyset \), \( x = y \) and \( m = 0 \)
02 While stopping conditions not satisfied
03 \[ m \leftarrow m + 1 \]
04 If \( x \notin K_{R(m)} \) then
05 Project \( x \) onto \( K_{R(m)} \) to get \( c_m \in \mathbb{R}^n \) and \( b_m \in \mathbb{R} \) so that
06 \[ \{ \tilde{x} \mid c_m^T \tilde{x} - b_m \geq 0 \} \] supports \( K_{R(m)} \).
07 While \( x \) is not optimal enough for QP\((\{1, \ldots, m\}, y)\)
08 Find \( i' \in \{ 1, \ldots, m \} \) such that \( c_{i'}^T x - b_{i'} < 0 \).
09 Use inner GI step to find an S-pair \((x', A')\) such that \( q(x') > q(x) \).
10 Let \((x, A) \leftarrow (x', A')\) and update KKT multipliers \( u_i \geq 0 \) for \( i \in A \).
11 end While.
12 \( \dagger \) Choose \( \lambda \in [1, 2] \), and decide whether to
13 \[ \text{set } A = \emptyset, y \leftarrow y + \lambda (x - y) \text{ and } x \leftarrow y. \]
14 end if.
15 end While.
16 Return \( x \).

2.2.1. Differences between Algorithms 2.4 and 2.10 The differences between Algorithms 2.4 and 2.10 are minimal. We now describe some differences between them. Since the only requirement of the SIP algorithm is to find a point in \( K \), the stopping conditions are adjusted. So in Algorithm 2.10, we now check how close \( x \) is to each of the \( K_t \) as an indicator of how close \( x \) is to \( K \).

The additional step marked with a (\( \dagger \)) indicates another change from Algorithm 2.4 to Algorithm 2.10. Before the step (\( \dagger \)), the point \( x \) is the projection of \( y \) onto the polyhedron defined by the constraints indexed by \( A \) (named as \( F(A) \) earlier in (2.2)). After step (\( \dagger \)), we move \( x \) and \( y \) to a new point.

We refer to [BZ05] Section 4.5] for how Fejér monotonicity is used to prove the convergence of general alternating projection algorithms, and to [Pan12] for how Fejér monotonicity is used to prove the convergence of Algorithm 2.10 when one projects onto a polyhedron generated by the projection process at some stage.

If we never apply step (\( \dagger \)) to move the point \( y \), then Algorithm 2.10 reduces to Algorithm 2.4. The disadvantage of not moving \( y \) is that if the projection of \( y \) onto \( K \) is where \( K \) is nonsmooth, then the convergence can be hindered considerably. See Sections 4 and 6. In other words, moving the point \( y \) may accelerate convergence because the convergence would be at where \( K \) is smoother. This behavior can also be compared with the theoretical results in [Pan12], where it was proved that the algorithm for the SIP there can converge superlinearly to a point in \( K \) under the assumption of linear regularity (though the size of the QPs to be solved can be large), but we did not manage to prove a similar result on any rate of convergence for the corresponding general algorithm for the BAP. Another question that arises is how often do we move \( x \) and \( y \). It might be advantageous to move \( x \) and \( y \) less often so that when \( x \) and \( y \) do move, they move much further, which may mean better convergence to a point in \( K \).

2.2.2. Choice of \( \lambda \) in (\( \dagger \)). We next remark on the choice of \( \lambda \). Based on the remarks after Algorithm 2.8, the choice \( \lambda \equiv 1 \) can be compared to the alternating projection algorithm and the choice \( \lambda \equiv 2 \) can be compared to the alternating reflection scheme. Given that we have stored the parameters \( c_i \) and \( b_i \) of the supporting hyperplanes for \( i \in \{1, \ldots, m\} \), it can be wise to make use of these information to decide which \( \lambda \) to choose. From this point onwards, let \( t = \lambda - 1 \). So we want to choose \( t \in [0, 1] \). From the inequalities \( c_i^T x - b_i \geq 0 \), we would want the inequalities

\[
\max_{c_i^T(x-y) > 0} b_i - c_i^T x, \min_{c_i^T(x-y) < 0} b_i - c_i^T x \geq 0
\]

If \( c_i^T x - b_i \geq 0 \) for all \( i \) such that \( c_i^T(x-y) = 0 \), then the interval of \( t \) for which (2.5) is satisfied is

\[
[\max_{c_i^T(x-y) > 0} b_i - c_i^T x, \min_{c_i^T(x-y) < 0} b_i - c_i^T x],
\]

provided \( \bar{t} \leq \bar{t} \), otherwise the interval is empty. Note that \( \bar{t} \) can be negative if there are still violated constraints to be introduced into the inner GI steps. A natural choice of \( t \) that is easily implementable is

\[
\hat{t} = \max \{0, \min\{1, \bar{t}/2\}\}.
\]

We see that \( \hat{t} \) is designed to lie in the range \([0, 1] \). In the case where \( 0 \leq \hat{t} < 2 \), the value \( \hat{t} \) is chosen such that \( x + \hat{t}(x-y) \) lies in the middle of the line segment
Figure 2.1. This figure explains the calculations in (2.6) and (2.7) when $\bar{t} \in (1, 2)$. The shaded polyhedron denotes the set of all points satisfying the inequality constraints. The point $x$ is the projection of $y$ onto this polyhedron. The value $\bar{t}$, possibly infinite, is the largest value of $t$ such that $x + t(x - y)$ satisfies all constraints. In a strategy where one reflects as much as possible, the value $\hat{t}$ as defined in (2.7) is the most reasonable choice.

$[x, x + \bar{t}(x - y)]$. A reason why this choice of $\hat{t}$ might be better than 1 is that the point $x + \bar{t}(x - y)$ is a point that is not yet ruled out by the supporting hyperplanes previously generated when $\bar{t} < 1$. See Figure 2.1 for an illustration.

2.3. Problems involving a fixed polyhedron. Suppose either the BAP or SIP can be written in such a way that for the sets $K_l \subset \mathbb{R}^n$, where $l \in \{1, \ldots, r\}$, one of the sets, say $K_1$, can be written as a polyhedron defined by some linear inequalities and equations. Let us write $K_1$ as

$$
K_1 = \{ \tilde{x} | \ c_i^T \tilde{x} - \bar{b}_i \geq 0 \text{ for all } i \in \{1, \ldots, m_I\} \\
\text{and } c_i^T \tilde{x} - \hat{b}_i = 0 \text{ for all } i \in \{1, \ldots, m_E\} \},
$$

(2.8)

where $m_I$ is the number of inequalities, $m_E$ is the number of equations, $c_i \in \mathbb{R}^n$ and $\bar{b}_i \in \mathbb{R}$ for all $i \in \{1, \ldots, m_I\}$, and $\hat{c}_i \in \mathbb{R}^n$, and $\hat{b}_i \in \mathbb{R}$ for all $i \in \{1, \ldots, m_E\}$. The case where $K_1$ is polyhedral can arise in a few situations:

1. The nonnegativity constraint $x \geq 0$ is a polyhedral constraint.
2. The linear inequalities that were produced during previous projections onto the $K_l$’s can be used to warm start the algorithm.

We also show how to exploit the added structure in equality constraints.

2.3.1. Equality constraints. While equality constraints in a QP were not explicitly addressed in the GI algorithm in [GI83], the authors must have known how to handle equality constraints. The equality constraints are always active, and the corresponding KKT multipliers $u_i$ are unrestricted in sign. We refer to the implementation in [Gas09] for the details.

Suppose the equality constraints can be easily written as

$$
\hat{C}^T x - \hat{b} = 0,
$$

(2.9)

and we assume that $\hat{C}$ has orthonormal columns. A reasonable heuristic to decide which violated (inequality) constraint to add into the inner GI step is to calculate the distance from the current iterate to the halfspace corresponding to each violated constraint. We
suggest an improvement to this strategy. Suppose $c_1^T x - b_1 \geq 0$ is a violated constraint. Then consider the constraint
\[ [c_1 - \hat{C}\hat{C}^T c_1]^T x - b_1 + c_1^T \hat{C}\hat{b} \geq 0 \] (2.10)

Notice that this constraint is equivalent to $c_1^T x - b_1 \geq 0$ because $c_1^T \hat{C}[\hat{C}^T x - \hat{b}] = 0$.

Proposition 2.11. (Projections involving affine spaces) Let the affine space $S$ be the set of points satisfying (2.9). Suppose $\bar{x} \in S$, and consider the polyhedron $F := \{ x : c_i^T x - b_i \geq 0, i \in \{1, \ldots, k\} \}$. If $\hat{C}^T c_i = 0$ for all $i \in \{1, \ldots, k\}$, then $P_F(\bar{x}) = P_{F \cap S}(\bar{x})$.

Proof. By the KKT conditions, $\bar{x} - P_F(\bar{x})$ is in span$\{c_i\}_{i=1}^k$. Since $\bar{x} \in S$, we have $P_F(\bar{x}) \in S$. Since $P_F(\bar{x}) \in F \cap S$ and $F \cap S \subset F$, we have $P_F(\bar{x}) = P_{F \cap S}(\bar{x})$ as needed. \qed

Proposition 2.11 shows us that once the orthogonalization operation explained in (2.10) is performed on all inequality constraints, then future iterates will lie in the affine subspace. Therefore, after the transformation in (2.10) is performed on a halfspace, the distance to each halfspace either remains the same or increases. The new distances give a better indication of which constraint should be added into the inner GI step, as explained in Figure 2.2.

The transformation in (2.10) is easy to perform once we are able to express the equality constraints so that the corresponding $\hat{C}$ has orthonormal columns.

2.3.2. Inequality constraints. We examine the case where the polyhedral set has only inequality constraints and no equality constraints. First, we identify two strategies on how to deal with polyhedral sets.
**Definition 2.12.** (Constraint collection and naive projection) We refer to the strategy of treating the set $K_1$ in (2.8) by introducing of $m_I$ inequality constraints and $m_E$ equality constraints, instead of handling $K_1$ as a set, as the constraint collection strategy. We refer to the strategy of treating $K_1$ as a set as the naive projection strategy.

To implement the constraint collection strategy when $K_1$ has only inequality constraints, these constraints can be incorporated by changing the "$m = 0$" in the first line of Algorithms 2.4 and 2.10 to "$m = m_I$" and let $c_i = \bar{c}_i$ and $b_i = \bar{b}_i$ for $i \in \{1, \ldots, m_I\}$, where $\bar{c}_i$ and $\bar{b}_i$ were defined in (2.8).

The naive projection strategy ignores the polyhedral structure and treats $K_1$ as just another set that is easy to project onto. The naive projection strategy may not be a bad idea in some cases, though more work should be done to compare different strategies. For example, if $K_1$ is the nonnegative orthant $\mathbb{R}^n_+$, then one projection, which is easy to perform, addresses a few violated nonnegativity constraints at one go.

3. Description and usage of SHDQP

In this section, we describe the basic elements of the Matlab implementation of SHDQP. Our algorithm attempts to solve the BAP or SIP numerically once given basic inputs like the starting iterate and the projection functions. Since Algorithms 2.4 and 2.10 are very similar, they are implemented together in SHDQP. We hope that the SHDQP code is easy to implement. We first test our algorithm for simple problems involving the intersection of two circles in $\mathbb{R}^2$.

We are grateful to [Gas09], whose implementation of the GI algorithm in C++ allowed us to save time programming our algorithm.

The following lines illustrate how to apply the shdqp code.

\begin{verbatim}
f_list={@(x)nmap_cir(x,[1;0],3), @(x)nmap_cir(x,[-1;0],3)};  \hspace{1cm} (3.1a)
strategy=int32([1,1]);               \hspace{1cm} (3.1b)
[x status C b q A r u Q R]=shdqp(y,f_list,strategy,iter,P,w); \hspace{1cm} (3.1c)
\end{verbatim}

3.1. Inputs of SHDQP. We first look at the inputs of shdqp in (3.1).

- **y**: The starting point
- **f_list**: The list of normal mapping functions
- **strategy**: Describes the order in which to project onto the sets $K_l$, $1 \leq l \leq r$. If this variable is a vector of all ones, then the shdqp algorithm would project onto the sets in a cyclic manner.
- **iter**: Maximum number of iterations
- **P**: Indicates whether to solve BAP or SIP. If $P = 0$, then it solves the BAP, and if $P = 1$, then it solves the SIP.
- **w**: The parameter $w \in [0, 1]$ for the SIP such that the actual $t$ that is used in the steps equals $wt$, where $t$ was defined in (2.7). (This parameter $w$ is irrelevant for the BAP.)

The variable $f_list$ in (3.1a) tells us how the normal mappings are accepted as arguments in shdqp. The sets to be projected onto in (3.1) are the circles with radii $3$ and centers $(\pm 1, 0)^T$ (which can be easily guessed from the code). In general, the normal mappings are specified as follows: $f_list(1)(x)$ equals $x - P_{K_1}(x)$. The vector $x - P_{K_1}(x)$ is the normal vector resulting from projecting $x$ onto $K_1$, which is why we call the functions in $f_list$ normal mappings. We decided to use the normal mappings
instead of using the projections because the parameters $c_m \in \mathbb{R}^n$ and $b_m \in \mathbb{R}$ of the halfspace $\{ \tilde{x} | c_m^T \tilde{x} - b_m \geq 0 \}$ are more directly deduced from the normal mappings than from the projection: The vector $c_m$ would equal $f_{\text{list}(1)}(x)$ divided by its norm, and the distance from $x$ to $y$ is the magnitude of $f_{\text{list}(1)}(x)$. Specifically, if $x$ lies in the set $K_i$, then $f_{\text{list}(1)}(x)$ equals zero. If the projections were used instead, numerical errors arising from cancellations might give a false supporting hyperplane.

3.2. Outputs of SHDQP. The program shdqp first determines $n$ from the size of $y$. Let $m$ be the number of projections performed. The matrix $C \in \mathbb{R}^{n \times m}$ and vector $b \in \mathbb{R}^m$ record the $\{c_i\}_{i=1}^m$ and $\{b_i\}_{i=1}^m$ produced by the projection process. The main output for the shdqp program: For the BAP, $x$ is the projection of $y$ onto $K = \bigcap_{i=1}^r K_i$, and for the SIP, $x$ is a point close to $K$. The other variables depend on the output status.

3.2.1. Feasible BAP. The BAP is feasible if $K \neq \emptyset$.

$q$ The number of active indices

$A$ The vector $A \in \mathbb{N}^q$ indicates the active indices

$Q, R$ The matrices $Q \in \mathbb{R}^{n \times q}$ and $R \in \mathbb{R}^{q \times q}$ are such that $(Q, R)$ is the QR factorization of the matrix $C(:, A(1:q))$.

$u$ The vector $u \in \mathbb{R}^q$ for the KKT multipliers for the active constraints. Specifically,

$$y - x + \sum_{i=1}^q u_i c_{A(i)} = 0. \quad (3.2)$$

Adding to the other KKT conditions $u_i \geq 0$ and $c_i^T x = b_i$ for all $i \in \{1, \ldots, q\}$, we have a certificate for the optimality of $x$.

The variable $r$ is irrelevant in the feasible case.

3.2.2. Infeasible BAP. The BAP is infeasible if $K = \emptyset$.

$q$ Records the number of active indices for the last iteration in which the problem was still not yet declared as infeasible.

$A$ The first $q$ entries of the vector $A \in \mathbb{N}^{q+1}$ records these active indices.

$Q, R$ Unchanged from feasible case.

The vector $r \in \mathbb{R}^{q+1}$, where $r \leq 0$, is a certificate of infeasibility. (The vector $r$ in this subsection is actually $(-1)$ in [G183] and in Section 5) The sum

$$\sum_{i=1}^{q+1} r_i c_{A(i)} \quad (3.3)$$

would be approximately equal to zero. (More precisely, the norm of the formula in (3.3) would be less than the lower tolerance explained in Section 5) Furthermore,

$$\sum_{i=1}^{q+1} r_i b_{A(i)} < 0. \quad (3.4)$$
If (3.3) is exactly zero, then the following simple argument shows that there is no \( \tilde{x} \) such that \( c_{A(i)}^T \tilde{x} \geq b_{A(i)} \) for all \( i \in \{1, \ldots, q+1\} \): If, on the contrary, there is some \( \tilde{x} \) satisfying all these inequalities, then

\[
0 = \sum_{i=1}^{q+1} r_i c_{A(i)}^T \tilde{x} \leq \sum_{i=1}^{q+1} r_i b_{A(i)} < 0,
\]

which is absurd.

3.2.3. The SIP. For the SIP, if \( K \neq \emptyset \), then the output \( x \) is a point close to \( K \). If \( K = \emptyset \), then the vector \( r \) is the same certificate of infeasibility as described in the last subsection.

3.3. Test code. We choose the following Let \( \mathbb{B}_r(x) \) be the ball with center \( x \) and radius \( r \). We carry out the following tests for the shdqp algorithm that are included in the file "test_cir.m". Let the starting variables defined by:

\[
\begin{align*}
&f_{\text{list}1} = \{(x)\text{mmap\_cir}(x,[1;0],3), (x)\text{mmap\_cir}(x,[-1;0],3)\} ; \\
&f_{\text{list}5} = \{(x)\text{mmap\_cir}(x,[5;0],3), (x)\text{mmap\_cir}(x,[-5;0],3)\} ; \\
&f_{\text{strategy}} = \text{int32}(\{1,1\}) ;
\end{align*}
\]

The following tests are performed for the different inputs, though we report only on Test 1 in this paper:

- Test 1: \([\ldots]\)=shdqp([0;10],f_{\text{list}1},f_{\text{strategy}},100,0,0);
- Test 2: \([\ldots]\)=shdqp([0;10],f_{\text{list}5},f_{\text{strategy}},100,0,0);
- Test 3: \([\ldots]\)=shdqp([0;10],f_{\text{list}1},f_{\text{strategy}},100,1,0);
- Test 4: \([\ldots]\)=shdqp([0;10],f_{\text{list}5},f_{\text{strategy}},100,1,0);
- Test 5: \([\ldots]\)=shdqp([0;10],f_{\text{list}1},f_{\text{strategy}},100,1,1);
- Test 6: \([\ldots]\)=shdqp([0;10],f_{\text{list}5},f_{\text{strategy}},100,1,1);

The '\([\ldots]\)' refers to the LHS of (3.1c). Tests 1, 3 and 5 involve feasible problems while Tests 2, 4 and 6 involve infeasible problems. Tests 1 and 2 attempt to solve the BAP, while the other tests attempt to solve the SIP.

We now observe the output for Test 1 more carefully:

** 1st test: Projection onto intersection of circles **
\( x'=[+0.000000 +10.000000], \ ||x-(0,\sqrt{8})||=7.171573e+00 \)
\( x'=[+0.701489 +2.985112], \ ||x-(0,\sqrt{8})||=7.187744e-01 \)
\( x'=[-0.047520 +2.910211], \ ||x-(0,\sqrt{8})||=9.458681e-02 \)
\( x'=[+0.058764 +2.849630], \ ||x-(0,\sqrt{8})||=6.247192e-02 \)
\( x'=[+0.000658 +2.828624], \ ||x-(0,\sqrt{8})||=5.918623e-04 \)
\( x'=[+0.000000 +2.828427], \ ||x-(0,\sqrt{8})||=4.692090e-08 \)
\( x'=[+0.000000 +2.828427], \ ||x-(0,\sqrt{8})||=4.787408e-16 \)
Check KKT conditions: 4.253499e-16
Check positivity of multipliers: 3.803301e+00

An observation that can be made about Test 1 is that the iterates (for that problem) are equivalent to finding the point that lies in the intersection of the supporting hyperplanes generated by each of the last projections onto the two circles. In other words,
the acceleration behavior illustrated in Figure 1.1 is observed. It took 10 projections to get to within machine precision of the projection $\left(0, \sqrt{8}\right)$. (The SHDQP algorithm for the SIP with parameter 0 doesn’t usually converge in finitely many steps, but in this case, the distance to each of the two circles have been reduced to 3 due to numerical underflow.) The distances of the iterates to $(0, \sqrt{8})$ suggests that the convergence is superlinear, as is suggested by the convergence of the Newton’s method and the theory in [GP98, GP01], where the smoothness of the boundaries of the sets were used in the proofs of superlinear convergence. This convergence is much faster than the linear convergence associated with algorithms for the SIP.

Lastly, note that the decrease of the distances $\|x_i - (0, \sqrt{8})\|$ is not even monotone. But if we look at the odd and even iterates, the convergence appears to be quadratic.

4. Testing SIP algorithms with the DNN cone

In this section, we test the speed of the different algorithms for the SIP to find a feasible point in the DNN cone from a random starting point.

Let $n$ be a positive integer. Let the cone of symmetric positive semidefinite matrices of size $\mathbb{R}^{n \times n}$ be $S^n_\circ$. The DNN (doubly nonnegative) cone, $\mathcal{D}_n$, is the set $S^n_\circ \cap \mathbb{R}^{n \times n}_+$. or in other words, the set of all symmetric positive semidefinite matrices such that all entries are nonnegative. It is clear that $\mathcal{D}_n \neq \emptyset$ since $\mathcal{D}_n$ contains the identity matrix, so the purpose of this exercise is just to test the speed of the algorithms for the SIP.

Example 4.1. (SIP for the DNN cone) In the script “test_sip.m”, we consider the DNN cone for the parameter $n = 100$. We use the naive projection strategy on $K_1 = \mathbb{R}^{n \times n}_+$ and $K_2 = S^n_\circ$ and apply different algorithms for the SIP to find a point in $\mathcal{D}_n$. The starting point (a symmetric matrix in $\mathbb{R}^{100 \times 100}$) is generated by applying the randn() function in Matlab to find each entry. The output below shows the typical result of how many iterations (i.e., the number of projections onto $K_1$ and $K_2$) it would take for the different algorithms for the SIP to converge to a point in the DNN cone.

*** Test 1: SHDQP(SIP) with parameter 1 ***
Number of iterations:  5
Min value/ eigenvalue of output: +1.176462e-03 +7.656788e-02

*** Test 2: SHDQP(SIP) with parameter 0 ***
Number of iterations: 23
Min value/ eigenvalue of output: +0.000000e+00 +2.942067e-16

*** Test 3: Alternating reflections ***
Number of iterations:  5
Min value/ eigenvalue of output: +1.176462e-03 +7.656788e-02

*** Test 4: Alternating Projections ***
Number of iterations: 111
Min value/ eigenvalue of output: +0.000000e+00 +1.448226e-17

*** Test 5: Douglas-Rachford ***
Number of iterations: 23
Min value/ eigenvalue of output: +0.000000e+00 +6.902958e-14

Like our tests in Subsection 3.3, the SIP algorithms may not converge in finitely many iterations in exact arithmetic, but finite convergence is still observed most of the time possibly due to numerical underflow. We see that the number of iterations required by
the SHDQP algorithm with parameter 1 and the alternating reflection scheme require the least amount of iterations to converge to a point in the DNN cone, and that the iterations for these two algorithms are identical. The number of iterations required for the SHDQP algorithm with parameter 0 is much less than that for the method of alternating projections. One reason is because the method of alternating projections often lands on a point that can be ruled out by previous supporting hyperplanes, and the extra step in the SHDQP algorithm of projecting onto the polyhedra generated by these supporting hyperplanes would accelerate convergence. The Douglas-Rachford algorithm has faster convergence than the alternating projection algorithm. We have shown an example where the Douglas-Rachford algorithm converges in the same amount of iterations as SHDQP with parameter 0. Sometimes the Douglas-Rachford algorithm is faster, and sometimes SHDQP with parameter 0 is faster, but the Douglas-Rachford algorithm is often faster.

One can easily generate an example where the SHDQP algorithm with parameter 1 would converge faster than the alternating reflection scheme. For example, consider the sets $K_1, K_2 \subset \mathbb{R}$ defined by $K_1 = (-\infty, 0]$ and $K_2 = [0, \infty)$. The SHDQP algorithm would do well, but the alternating reflection scheme would not converge. It remains to be seen how the various algorithms for the SIP would compare for problems typically appearing in practice.

5. Ill-conditioning in the constraints

In this section, we describe how SHDQP handles the ill-conditioning of the matrix formed by concatenating the normal vectors (produced earlier by the projection process) corresponding the active constraints. We defer numerical results comparing different choices for tolerances in Subsection 6.3.

We give the full details of the inner GI step in the manner presented in [GI83] before continuing with the rest of the discussions. We keep to the notation as given in [GI83], though we only describe our algorithm only for the particular case where the Hessian of the QP is the identity matrix.

**Algorithm 5.1.** (Inner GI step: Full details) Let $(x, A)$ be an S-pair. Suppose $p \notin A$ is such that $c_p^T x - b_p < 0$. Let $n^+ = e_p$. This step attempts to find a new S-pair $(x', A')$ such that $A' \subset A \cup \{p\}$.

Let $q = |A|$. Let the matrix $N \in \mathbb{R}^{n \times q}$ be the matrix of normal vectors of the constraints in the active set indexed by $A$. In our implementation, the $i$th column of $N$ equals $c_{A(i)}$. The columns of $N$ are linearly independent. Let the economy QR decomposition of $N$ be $QR = N$, where $Q \in \mathbb{R}^{n \times q}$ and $R \in \mathbb{R}^{q \times q}$. Let also $N^*$ be the pseudo-inverse or Moore-Penrose generalized inverse of $N$. In other words, $N^*$ can be written as $N^* = R^{-1}Q^T$.

Let the KKT multipliers $u \in \mathbb{R}^q_{+}$ be such that

$$y - x + \sum_{i \in A} u_i c_{A(i)} = 0. \quad (5.1)$$

The vector $u^+ \in \mathbb{R}^{q+1}_{+}$ is set to be $u^+ = \left( u^*, 0 \right)$ at the start of the inner GI step. If $q = 0$, then $u^+ \in \mathbb{R}^1_{+}$ is first set to be $u^+ = 0$. By the end of the inner GI step, $u^+$ would be the new multipliers corresponding to $(x', A')$.

The inner GI step was step 2 in the dual algorithm in [GI83]:

(a) Determine step length
Compute \( z = n^+ - QQ^T n^+ \) (the step direction in the primal space) and if \( q > 0 \),
\( r = R^{-1} Q^T n^+ \).

(b) Compute step length

(i) Partial step length \( t_1 \): (maximum step in dual space without violating dual feasibility).
If \( r \leq 0 \) or \( q = 0 \), set \( t_1 \leftarrow \infty \);
otherwise, set
\[
  t_1 \leftarrow \min_{j=1,\ldots,q} \left\{ \frac{u_j^+(x)}{r_j} \right\} = \frac{u_j^+(x)}{r_j}.
\]

In step (c) below, element \( k \in K \) corresponds to the \( l \)th element of \( A \).

(ii) Full step length \( t_2 \): (minimum step in primal space such that the \( p \)th constraint becomes feasible)
If \( |z| = 0 \), set \( t_2 \leftarrow \infty \);
otherwise, set \( t_2 \leftarrow - (c^T p x - b_p) / z^T n^+ \).

(iii) Step length, \( t \):
Set \( t \leftarrow \min(t_1, t_2) \).

(c) Determine new S-pair and take step:

(i) No step in primal or dual space:
If \( t = \infty \), STOP: Constraints defined by \( A \cup \{ p \} \) are infeasible.
(ii) Step in dual space:
If \( t_2 = \infty \), then set \( u^+ \leftarrow u^+ + t \left( \gamma_1 \right) \), and drop constraint \( k \); i.e., set \( A \leftarrow A \setminus \{ k \} \), \( q \leftarrow q - 1 \), update \( Q \) and \( R \), and go to step 2(a).
(iii) Step in primal and dual space:
Set \( x \leftarrow x + t z \),
\( u^+ \leftarrow u^+ + t \left( \gamma_1 \right) \).
If \( t = t_2 \) (full step), set \( u \leftarrow u^+ \) and add constraint \( p \); i.e., set \( A \leftarrow A \cup \{ p \} \), \( q \leftarrow q + 1 \), update \( Q \) and \( R \) and the inner GI step ends.
If \( t = t_1 \) (partial step) drop constraint \( k \); i.e., set \( A \leftarrow A \setminus \{ k \} \), \( q \leftarrow q - 1 \), update \( Q \) and \( R \), and go to step (a).

5.1. The lower tolerance. We focus our attention on step (b)(ii), where we have to decide whether \( |z| = 0 \). The columns of \( N \) and the vector \( n^+ \) are always of norm 1 in our implementation in SHDQP, so we decide on a lower tolerance slightly larger than machine epsilon, say \( 10^{-13} \), such that if \( |z| < 10^{-13} \), then \( z \) is considered to be zero.

But if the lower tolerance was set too high, then the equation (5.1) is affected. Notice that in step (c)(ii), we set \( u^+ \leftarrow u^+ + t \left( \gamma_1 \right) \). Observing a convention \( A(q+1) = p \) and let \( \Delta u^+ = t \left( \gamma_1 \right) \), we have
\[
  \sum_{i=1}^{q+1} \Delta u_i^+ c_{A(i)} = t[N(-r) + c_q]
  = t[Q R (-R^{-1} Q^T c_q) + c_q]
  = t[c_q - QQ^T c_q]
  = t z.
\]

The term in the LHS of (5.1) would incur an error of \( tz \) in the next iteration. As these errors accumulate, the validity of the equation (5.1) will be affected.
5.2. The upper tolerance. After step (c)(iii) is completed, \( R_{q,q} \), the element in the bottom right corner of \( R \in \mathbb{R}^{q \times q} \), is \( |z| \). Considering that \( R \) is upper triangular and the columns of \( R \) have norm 1, having \( |z| \) too small will make the matrix \( R \) ill-conditioned. Notice \( R^{-1} \) is used in the calculation of \( r \). Therefore, we have introduced an upper tolerance \( 10^{-4} \) such that if \( |z| \in (10^{-13}, 10^{-4}) \), then we aggregate the constraints produced.

In our implementation, all the constraints produced by projecting onto the set \( K_l \) would be aggregated to one constraint \( c_{-l} \in \mathbb{R}^n \) for \( 1 \leq l \leq r \). The vectors \( z, Q, R, u \) all have to be recalculated. The new \( R \) matrix will be smaller and is more well conditioned than the previous \( R \). If we still have \( |z| \in (10^{-13}, 10^{-4}) \) at the end of the recalculations, then this can indicate that the the problem is ill-conditioned. We obtain numerical results for two different settings of the two tolerances in Subsection 6.3.

6. Projecting onto the DNN cone

In this section, we show the numerical results for the problem of projecting onto the DNN cone \( D_n \). We had seen earlier in Section 4 that the naive projection strategy for the SIP, where we treat \( \mathbb{R}^{n \times n}_+ \) as just another set instead of collecting all the inequality constraints defining \( \mathbb{R}^{n \times n}_+ \), fares reasonably well under the alternating reflection scheme or the SHDQP (SIP) algorithm with parameter 1. We shall see in this section that the BAP problem involving the DNN cone \( D_n \) appears to be a difficult problem, and we compare the naive projection and constraint collection strategies.

To simplify discussions in this section, we write \( \bar{n} = \frac{1}{2}n(n + 1) \), which is also the dimension of the set \( S^n \), the set of symmetric matrices. We understand \( x \) as either a (not necessarily positive definite) symmetric matrix (i.e., \( x \in S^n \)), or as a (not necessarily nonnegative) vector in \( \mathbb{R}^{\bar{n}} \).

6.1. Implementation notes. We remark on a few things about the dual QP algorithm that are different from the implementation in [GI83]. We used the economy QR factorization instead of the dense QR factorization as suggested in [GI83].

6.2. Analysis of output. We apply the SHDQP algorithms for the BAP for the problem of projecting onto the DNN cone. The starting point is, like Example 4.1 chosen to be in \( S^{100} \) (or \( \mathbb{R}^{5050} \)) by using the randn() function in Matlab to generate each entry. Our experiments are performed by running the file “test_sdnn.m”.

![Figure 6.1](image-url) Figure 6.1. Plots showing the progress of the naive projection strategy for projecting a randomly generated point onto the DNN cone.
Here are a few remarks on the diagrams in Figure 6.1. In the diagram on the left, we see four different plots. For the two straight line plots, the distance of the iterates \( x \) to \( \mathbb{R}_+^n \) and \( S^+_n \) are plotted. While the distance to the sets \( \mathbb{R}_+^n \) and \( S^+_n \) are generally decreasing, our experiments show that the decrease becomes quite slow: the distances of the iterates to \( \mathbb{R}_+^n \) and \( S^+_n \) do not decrease very much afterward. The diagram on the right plots the number of active constraints against the number of iterations.

A possible explanation for the slow convergence of the naive projection method for the BAP for projecting onto the DNN cone, in contrast to the fast convergence that was observed for the tests in Section 4, is as follows. The sets \( S^+_n \) and \( \mathbb{R}_+^{n \times n} \) are highly nonsmooth at the point \( P_{D_n}(x) \). Imagine projecting a point \( y \in \mathbb{R}^n \) onto a polyhedron with \( m \) constraints, where \( m \) is a large number. Any algorithm would need to figure out the active constraints. This suggests that for the BAP involving the DNN cone, the number of supporting hyperplanes that need to be generated would need to be quite large.

A large number of iterations seems to be necessary to solve the BAP for projecting onto the DNN cone, so we now see how the constraint collection strategy behaves. The set \( \mathbb{R}_+^n \) is now treated as \( n \) nonnegativity inequalities instead, and we project only onto \( S^+_n \).

**Figure 6.2.** Plots showing the progress of the constraint collection strategy for projecting a randomly generated point onto the DNN cone.

We conduct experiments for the constraint collection strategy by running the file “test_dnn.m”. In Figure 6.2, the first diagram is a plot of the distance of the iterates
to $S^n_+$ and the largest magnitude of the negative eigenvalue. The next two diagrams show the number of the different types of active constraints against iteration count.

The eigenvalue decompositions arising from projection onto $S^n_+$ can be considered the most expensive step for this problem, and we compare the two runs in Figures 6.1 and 6.2. The two experiments (corresponding to the same starting points) are comparable because 800 iterations for the naive projection strategy correspond to 400 projections onto $S^n_+$, the same number of projections onto $S^n_+$ as 400 iterations in the constraint collection strategy. We see that in the constraint collection strategy, the additional expense of projecting onto $\mathbb{R}^n_-$ in the constraint collection strategy doesn’t seem to bring about accelerated convergence.

In Subsection 6.3, we run experiments involving a smaller value of $n$. Other than testing the tolerances explained in Subsections 5.1 and 5.2 we make an observation about the DNN problem.

6.3. Ill-conditioning in the constraints. We run our experiments for $n = 20$ (corresponding to $\tilde{n} = 210$) for the problem of projecting onto the DNN cone, and make a few observations. The numerical results are summarized in Figure 6.3.

Our numerical experiments were conducted in Matlab, where the machine epsilon is roughly $2 \times 10^{-16}$. We see that in the first two plots, where the lower and upper tolerances defined in Subsections 5.1 and 5.2 were set to be $10^{-13}$ and $10^{-4}$, the absolute value of the most negative eigenvalue and most negative values of the iterates go to zero, indicating the convergence of the iterates to the projection onto the DNN cone. In the last plot, where the tolerances were set to be $10^{-13}$ and $10^{-12}$ instead, we see that the algorithm fails to converge properly. The likely reason, as explained in Section 5, is that the $R$ matrix derived from the QR factorization becomes ill-conditioned.

We now make a remark about the problem of projecting onto the DNN cone. As seen from the values of the absolute values of the most negative eigenvalues and most negative values, we see that the convergence of the iterates to $\mathbb{R}^n_-$ is better than the convergence of the iterates to $S^n_+$. There are $\tilde{n}$ linear inequality constraints in $\mathbb{R}^n_-$, and $n$ eigenvalue positivity constraints in $S^n_+$. A reasonable explanation is that it is more challenging to approximate the normal cones at points on the boundary of $S^n_+$ using polyhedrons than to approximate $\mathbb{R}^n_-$ in a similar manner.

7. Conclusion

We described and showed our implementation of the SHDQP algorithm for the SIP (1.1) and the BAP (1.2). The theory of using supporting hyperplanes and quadratic programming to solve these set intersection problems were first laid in [Pan12]. An implementation SHDQP using these ideas is realized in this paper by feeding the constraints generated by the projection process into what we call the inner GI step described in the dual active set quadratic programming algorithm in [GI83].

The SHDQP algorithm can solve some simple problems at a rate better than linear convergence typically associated with the method of alternating projections, as suggested by the experiments in Section 3.3. Certificates of optimality (the KKT multipliers $u$) or infeasibility (the multiplier $r$) are obtained when the algorithm can terminate under the stopping conditions. It is hoped that the code is easy to implement.

Our design of the SHDQP algorithm for SIP problems takes advantage of how one can feed linear inequality constraints into the dual QP algorithm. Among the problems we tested, the SHDQP algorithm does quite well when compared to classical methods.
like alternating projections and the Douglas-Rachford algorithm, especially when the input parameter is set to 1, where we reflect instead of project as often as possible, as seen in Section 4.

The BAP problem of projecting onto the DNN cone is however challenging for the SHDQP algorithm, possibly because the projection problems involving the DNN cone can be nonsmooth at the projection. In our example in Subsection 6.3, it seems that the source of difficulty is that $S_+$ is not polyhedral. On the other hand, the SIP problem moves the point to project from around, and therefore avoids the nonsmooth points to achieve faster convergence.

The effectiveness of the general strategy of supporting hyperplanes and quadratic programming for typical problems in practice remains to be seen.
Further directions include improving the performance, memory usage, functionality, and other aspects of the algorithm so that we can have an effective algorithm that requires minimal human input.

8. Acknowledgments

I thank Toh Kim Chuan for making some very important suggestions in an earlier draft, and for bringing to my attention the problem of projecting onto the DNN cone. This problem turns out to test the limits of the Best Approximation Problem.

References

[Agm83] S. Agmon, *The relaxation method for linear inequalities*, Canad. J. Math. 4 (1983), 479–489.
[BB96] H.H. Bauschke and J.M. Borwein, *On projection algorithms for solving convex feasibility problems*, SIAM Rev. 38 (1996), 367–426.
[BD86] J.P. Boyle and R.L. Dykstra, *A method for finding projections onto the intersection of convex sets in Hilbert spaces*, Lecture Notes in Statistics (1986), 28–47.
[BR09] E.G. Birgin and M. Raydan, *Dykstra’s algorithm and robust stopping criteria*, Encyclopedia of Optimization (C. A. Floudas and P. M. Pardalos, eds.), Springer, US, 2 ed., 2009, pp. 828–833.
[BZ05] J.M. Borwein and Q.J. Zhu, *Techniques of variational analysis*, Springer, NY, 2005, CMS Books in Mathematics.
[CC11] A. Cegielski and Y. Censor, *Opial-type theorems and the common fixed point problem*, Fixed-Point Algorithms for Inverse Problems in Science and Engineering (H.H. Bauschke, R. Burachik, P.L. Combettes, V. Elser, D.R. Luke, and H. Wolkowicz, eds.), Springer-Verlag, Berlin, 2011, pp. 155–183.
[Deu95] F. Deutsch, *The angle between subspaces of a Hilbert space*, Approximation Theory, Spline Functions and Applications (S. P. Singh, ed.), Kluwer Academic Publ., The Netherlands, 1995, pp. 107–130.
[Deu01a], *Accelerating the convergence of the method of alternating projections via a line search: A brief survey*, Inherently Parallel Algorithms in Feasibility and Optimization and their Applications (D. Butnariu, Y. Censor, and S. Reich, eds.), Elsevier, 2001, pp. 203–217.
[Deu01b], *Best approximation in inner product spaces*, Springer, 2001, CMS Books in Mathematics.
[DR56] J. Douglas and H.H. Rachford, *On the numerical solution of heat conduction problems in two and three space variables*, Trans. Amer. Math. Soc. 82(2) (1956), 421–439.
[Dyk83] R.L. Dykstra, *An algorithm for restricted least-squares regression*, J. Amer. Statist. Assoc. 78 (1983), 837–842.
[ER11] R. Escalante and M. Raydan, *Alternating projection methods*, SIAM, 2011.
[Gas09] L. Di Gaspero, *Quadrprog++*, 2009, http://quadrprog.sourceforge.net/.
[GL83] D. Goldfarb and A. Idnani, *A numerically stable dual method for solving strictly convex quadratic programs*, Math. Programming 27 (1983), 1–33.
[GP98] U.M. García-Palomares, *A superlinearly convergent projection algorithm for solving the convex inequality problem*, Oper. Res. Lett. 22 (1998), 97–103.
[GP01] , *Superlinear rate of convergence and optimal acceleration schemes in the solution of convex inequality problems*, Inherently Parallel Algorithms in Feasibility and Optimization and their Applications (D. Butnariu, Y. Censor, and S. Reich, eds.), Elsevier, 2001, pp. 297–305.
[Han88] S.P. Han, *A successive projection method*, Math. Program. 40 (1988), 1–14.
[Kru06] A.Y. Kruger, *About regularity of collections of sets*, Set-Valued Anal. 14 (2006), 187–206.
[LM79] P.L. Lions and B. Mercier, *Splitting algorithms for the sum of two nonlinear operators*, SIAM J. Numer. Anal. 16(6) (1979), 964–979.
[Pan12] C.H.J. Pang, *Set intersection problems: Supporting hyperplanes and quadratic programming*, arxiv:1212.6843.
[Pie84] G. Pierre, *Decomposition through formalization in a product space*, Math. Programming 28 (1984), 96–115.

E-mail address: matpckj@nus.edu.sg