Credible Autocoding of Convex Optimization Algorithms

Timothy Wang, Romain Jobredeaux, Marc Pantel, Pierre-Loïc Garoche, Eric Féron, Didier Henrion

Credible Autocoding of Convex Optimization Algorithms

Timothy Wang[1], Romain Jobredeaux[1], Marc Pantel[4], Pierre-Loic Garoche[2], Eric Feron[1], and Didier Henrion[3]

[1] Georgia Institute of Technology, Atlanta, Georgia, USA
[2] ONERA – The French Aerospace Lab, Toulouse, FRANCE
[3] CNRS-LAAS – Laboratory for Analysis and Architecture of Systems, Toulouse, FRANCE
[4] ENSEEIHT, Toulouse, FRANCE

Abstract. The efficiency of modern optimization methods, coupled with increasing computational resources, has led to the possibility of real-time optimization algorithms acting in safety critical roles. There is a considerable body of mathematical proofs on on-line optimization programs which can be leveraged to assist in the development and verification of their implementation. In this paper, we demonstrate how theoretical proofs of real-time optimization algorithms can be used to describe functional properties at the level of the code, thereby making it accessible for the formal methods community. The running example used in this paper is a generic semi-definite programming (SDP) solver. Semi-definite programs can encode a wide variety of optimization problems and can be solved in polynomial time at a given accuracy. We describe a top-to-down approach that transforms a high-level analysis of the algorithm into useful code annotations. We formulate some general remarks about how such a task can be incorporated into a convex programming autocoder. We then take a first step towards the automatic verification of the optimization program by identifying key issues to be adressed in future work.

Keywords: Control Theory, Autocoding, Lyapunov proofs, Formal Verification, Optimization, Interior-point Method, PVS, frama-C

1 Introduction

The applications of optimization algorithms are not only limited to large scale, off-line problems on the desktop. They also can perform in a real-time setting as part of safety-critical systems in control, guidance and navigation. For example, modern aircrafts often have redundant control surface actuation, which allows the possibility of reconfiguration and recovery in the case of emergency. The precise re-allocation of the actuation resources can be posed, in the simplest case, as a linear optimization problem that needs to be solved in real-time.

In contrast to off-line desktop optimization applications, real-time embedded optimization code needs to satisfy a higher standard of quality, if it is to be used within a safety-critical system. Some important criteria in judging the quality of an embedded code include the predictability of its behaviors and whether or not its worst case computational time can be bounded. Several authors including Richter [17], Feron and McGovern [10][9] have worked on the certification problem for on-line optimization algorithms used in control, in particular on worst-case execution time issues. In those cases, the authors have chosen to tackle the problem at a high-level of abstraction. For example, McGovern reexamined the proofs of computational bounds on interior point methods for semi-definite programming; however he stopped short of using the proofs to analyze the implementations of interior point methods. In this paper, we extend McGovern's work further by demonstrating the expression of the proofs at the code level for the certification of on-line optimization code. The utility of such demonstration is twofolds. First, we are considering the reality that the verifications of safety-critical systems are almost always done at the source code level. Second, this effort provides an example output that is much closer to being an accessible form for the formal methods community.

The most recent regulatory documents such as DO-178C [20] and, in particular, its addendum DO-333 [21], advocate the use of formal methods in the verification and validation of safety critical software. However, complex computational cores in domain specific software such as control or optimization software make their automatic analysis difficult in the absence of input from domain experts. It is the authors' belief that communication between the communities of formal software analysis and domain-specific communities, such as the optimization community, are key to successfully express the semantics of these complex algorithms in a language compatible with the application of formal methods.

The main contribution of this paper is to present the expression, formalization, and translation of high-level functional properties of a convex optimization algorithm along with their proofs down to the code level for the purpose of formal program verification. Due to the complexity of the proofs, we cannot yet as of this moment, reason about them soundly on the implementation itself. Instead we choose an intermediate level of abstraction of the implementation where floating-point operations are replaced by real number algebra.

The algorithm chosen for this paper is based on a class of optimization methods known collectively as interior point methods. The theoretical foundation behind modern interior point methods can be found in Nemrovskii et. al [13][14]. The key result is the self-concordance of certain barrier functions that guarantees the convergence of a Newton iteration to an ϵ-optimal solution in polynomial time. For more details on polynomial-time interior point methods, readers can refer to [15].

Interior-point algorithms vary in the Newton search direction used, the step length, the initialization process, and whether or not the algorithm can return infeasible answers in the intermediate iterations. Some example search directions are the Alizadeh-Haeberly-Overton (AHO) direction [1], the Monteiro-Zhang (MZ) directions [12], the Nesterov-Todd (NT) direction [16], and the Helmberg-Kojima-Monteiro (HKM) direction [4]. It was later determined in [11] that all of these search directions can be captured by a particular scaling matrix in the linear transformation introduced by Sturm and Zhang in [25]. An accessible introduction to semi-definite programming using interior-point method can be found in the works of Boyd and Vandenberghe [3].

Autocoding is the computerized process of translating the specifications of an algorithm, that is initially expressed in a high-level modeling language such as Simulink, into source code that can be transformed further into an embedded executable binary. An example of an autocoder for optimization programs can be found in the work of Boyd [8]. One of the main ideas behind this paper, is that by combining the efficiency of the autocoding process with the rigorous proofs obtained from a formal analysis of the optimization algorithm, we can create a credible autocoding process [24] that can rapidly generate formally verifiable optimization code.

For this paper, we selected an interior point algorithm with the Monteiro-Zhang (MZ) Newton search direction as the running example. The step length is fixed to be one and the input problem is a generic semi-definite programming problem obtained from system and control. The paper is organized as follows: first we introduce the basics of semi-definite programming and program verification. We then introduce a specific interior point algorithm and recall its properties. After that, we discuss some general principles in how optimization algorithms can be included as part of the credible autocoding framework in generating domain-specific properties and their proofs expressed in the language of the generated optimization source code. We then give an example of a code implementation annotated with the semantics of the optimization algorithm using the Floyd-Hoare method [5]. Finally, we discuss how such autocoding environment can be used as part of the certification process and discuss some future directions of research.

2 Credible Autocoding: General Principles

In this paper, we introduce a credible autocoding framework for convex optimization algorithms. Credible autocoding, analogous to credible compilation from [18], is a process by which the autocoding process generates formally verifiable evidence that the output source code correctly implements the input model. An overall view of a credible autocoding framework is given in figure 1. Existing work already provides for the automatic generation of embedded convex optimization code [8]. Given that proofs of high-level functional properties of interior point algorithms do exist, we want to generate the same proof that is sound for the implementation, and expressed in a formal specification language embedded in the code as comments. One of the key ingredients that made credible autocoding applicable for control systems [23] is that the ellipsoid sets generated by synthesizing quadratic Lyapunov functions are relatively easy to reason about even on the code level. The semantics of interior point algorithms, however, do not rely on simple quadratic invariants. The invariant obtained from the proof of good behavior of interior point algorithms is generated by a logarithmic function.

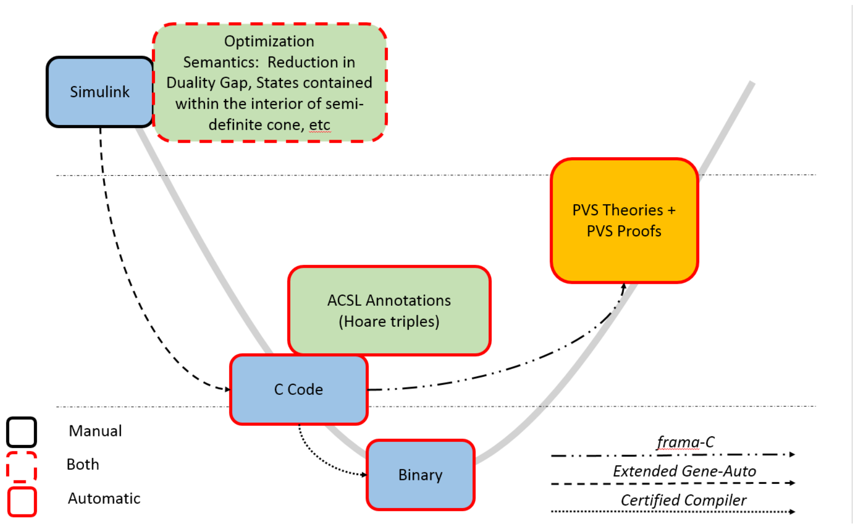
Simulink
Optimization Semantics: Reduction in Duality Gap, States contained within the interior of semi-definite cone, etc
PVS Theories + PVS Proofs
ACSL Annotations (Hoare triples)
C Code
Binary
Manual
Both
Automatic
frama-C
Extended Gene-Auto
Certified Compiler

Fig. 1. Visualization of autocoding and verification process For Optimization Algorithms

This same logarithmic function can also be used in showing the optimization algorithm terminates in within a specified time. This function, is not provided, is perhaps impossible to synthesize from using existing code analysis techniques on the optimization source code. In fact nearly all existing code analyzers only handle linear properties with the notable exception in [19]

3 Semi-Definite Programming and the Interior Point Method

In this section, we give an overview of the Semi-Definite Programming (SDP) problem. The readers who are already familiar with interior point method and convex optimization should skip ahead to the next section. The notations used in this section are as follows: let $A = (a_{i,j})_{1\leq i,j \leq n}, B \in \mathbb{R}^{n\times n}$ be two matrices and $a, b \in \mathbb{R}^n$ be two column vectors. $\mathrm{Tr}\,(A) = \sum_{i=1}^{n} a_{i,i}$ denotes the trace of matrix A. $\langle \cdot, \cdot \rangle$ denotes an inner product, defined in $\mathbb{R}^{n\times n} \times \mathbb{R}^{n\times n}$ as $\langle A, B\rangle := \mathrm{Tr}\left(B^{\mathrm{T}} A\right)$ and in $\mathbb{R}^n \times \mathbb{R}^n$ as $\langle a, b\rangle := a^{\mathrm{T}} b$. The Frobenius norm of A is defined as $\|A\|_F = \sqrt{\langle A, A\rangle}$. The symbol $\mathbb{S}^n$ denotes the space of symmetric matrices of size $n \times n$. The space of $n \times n$ symmetric positive-definite matrices is denoted as $\mathbb{S}^{n+} = \left\{S \in \mathbb{S}^n | \forall x \in \mathbb{R}^n \setminus \{0\}, x^{\mathrm{T}} S x > 0\right\}$. If A and B are symmetric, $A \prec B$ (respectively $A \succ B$) denotes the positive (respectively negative)-definiteness of matrix $B - A$. The symbol I denotes an identity matrix of appropriate dimension.

For $X, Z \in \mathbb{S}^{n+}$, some basic properties of matrix derivative are $\dfrac{\partial \operatorname{Tr}(XZ)}{\partial X} = Z^{\mathrm{T}} = Z$ and $\dfrac{\partial \det(X)}{\partial X} = \det(X)^{-1}\left(X^{-1}\right)^{\mathrm{T}} = \det(X)^{-1} X^{-1}$.

3.1 SDP Problem

Let $n, m \in \mathbb{N}$, $F_0 \in \mathbb{S}^{n+}$, $F_1, F_2, \ldots, F_m \in \mathbb{S}^n$, and $b = \left[b_1\, b_2 \ldots b_m\right]^{\mathrm{T}} \in \mathbb{R}^m$. Consider a SDP problem of the form in (1). The linear objective function $\langle F_0, Z\rangle$ is to be maximized over the intersection of positive semi-definite cone $\{Z \in \mathbb{S}^n | Z \succeq 0\}$ and a convex region defined by m affine equality constraints.

$$\begin{aligned} \sup_{Z} \quad & \langle F_0, Z\rangle, \\ \text{subject to} \quad & \langle F_i, Z\rangle + b_i = 0, \quad i = 1, \ldots, m \\ & Z \succeq 0. \end{aligned} \tag{1}$$

Note that a SDP problem can be considered as a generalization of a linear programming (LP) problem. To see this, let $Z = \operatorname{Diag}(z)$ where z is the standard LP variable.

We denote the SDP problem in (1) as the *dual* form. Closely related to the dual form, is another SDP problem as shown in (2), called the *primal* form. In the primal formulation, the linear objective function $\langle b, p\rangle$ is minimized over all vectors $p = \left[p_1 \ldots p_m\right]^{\mathrm{T}} \in \mathbb{R}^m$ under the semi-definite constraint $F_0 + \sum_{i=1}^{m} p_i F_i \preceq 0$. Note the introduction in (2) of a variable $X = -F_0 - \sum_{i=1}^{m} p_i F_i$ such that $X \succeq 0$, which is not strictly needed to express the problem, but is used in later developments.

$$\begin{aligned} \inf_{p,X} \quad & \langle b, p\rangle \\ \text{subject to} \quad & F_0 + \sum_{i=1}^{m} p_i F_i + X = 0 \\ & X \succeq 0. \end{aligned} \tag{2}$$

We assume the primal and dual feasible sets defined as

$$\begin{aligned} \mathcal{F}^p &= \left\{ X | X = -F_0 - \sum_{i=1}^{m} p_i F_i \succeq 0, p \in \mathbb{R}^m \right\}, \\ \mathcal{F}^d &= \{Z | \langle F_i, Z\rangle + b_i = 0, Z \succeq 0\} \end{aligned} \tag{3}$$

are not empty. Under this condition, for any primal-dual pair (X, Z) that belongs to the feasible sets in (3), the primal cost $\langle b, p\rangle$ is always greater than or equal to the dual cost $\langle F_0, Z\rangle$. The difference between the primal and dual costs for a feasible pair (X, Z) is called the *duality gap*. The duality gap is a measure of the optimality of a primal-dual pair. The smaller the duality gap, the more optimal the solution pair (X, Z) is. For (2) and (1), the duality gap is the function

$$G(X, Z) = \operatorname{Tr}(XZ)\,. \tag{4}$$

Indeed,

$$\begin{aligned}\operatorname{Tr}(XZ) &= \operatorname{Tr}\left(\left(-F_0 - \sum_{i=1}^{m} p_i F_i\right) Z\right) = -\operatorname{Tr}(F_0 Z) - \sum_{i=1}^{m} p_i \operatorname{Tr}(F_i Z)\\ &= \langle b, p\rangle - \langle F_0, Z\rangle.\end{aligned}$$

Finally, if we assume that both problems are strictly feasible i.e. the sets

$$\begin{aligned}\mathcal{F}^{p'} &= \left\{X | X = -F_0 - \sum_{i=1}^{m} p_i F_i \succ 0, p \in \mathbb{R}^m\right\},\\ \mathcal{F}^{d'} &= \{Z | \langle F_i, Z\rangle + b_i = 0, Z \succ 0\}\end{aligned} \tag{5}$$

are not empty, then there exists an optimal primal-dual pair (X^*, Z^*) such that

$$\operatorname{Tr}(X^* Z^*) = 0. \tag{6}$$

Moreover, the primal and dual optimal costs are guaranteed to be finite. The condition in (6) implies that for strictly feasible problems, the primal and dual costs are equal at their respective optimal points X^* and Z^*. Note that in the strictly feasible problem, the semi-definite constraints become definite constraints.

The canonical way of dealing with constrained optimization is by first adding to the cost function a term that increases significantly if the constraints are not met, and then solve the unconstrained problem by minimizing the new cost function. This technique is commonly referred to as the *relaxation* of the constraints. For example, lets assume that the problems in (2) and (1) are strictly feasible. The positive-definite constraints $X \succ 0$ and $Z \succ 0$, which defines the *interior* of a pair of semi-definite cones, can be relaxed using an indicator function $I(X, Z)$ such that

$$I : (X, Z) \to \begin{cases} 0, & X \succ 0, Z \succ 0\\ +\infty, & \text{otherwise}\end{cases} \tag{7}$$

The intuition behind relaxation using an indicator function is as follows. If the primal-dual pair (X, Z) approaches the boundary of the interior region, then the indicator function $I(X, Z)$ approaches infinity, thus incurring a large penalty on the cost function.

The indicator function in (7) is not useful for optimization because it is not differentiable. Instead, the indicator function can be replaced by a family of smooth, convex functions $B(X, Z)$ that not only approximate the behavior of the indicator function but are also *self-concordant*. We refer to these functions as *barrier* functions. A scalar function $F : \mathbb{R} \to \mathbb{R}$, is said to be self-concordant if it is at least three times differentiable and satisfies the inequality

$$|F'''(x)| \leq 2F''(x)^{\frac{3}{2}}. \tag{8}$$

The concept of self-concordance has been generalized to vector and matrix functions, thus we can also find such functions for the positive-definite variables X and Z. Here we state, without proof, the key property of self-concordant functions.

Property 1. Functions that are self-concordant can be minimized in polynomial time to a given non-zero accuracy using a Newton type iteration [14].

Examples of self-concordant functions include linear functions, quadratic functions, and logarithmic functions. A valid barrier function for the semi-definite constraints from (2) and (1) is

$$B(X, Z) = -\log\det(X) - \log\det(Z). \tag{9}$$

4 Introduction to Program Verification

In this section, we introduce some concepts from program verification that we use later in the paper. The readers who are already familiar with Hoare logic and axiomatic semantics should skip ahead to the next section.

4.1 Axiomatic Semantics

One of the classic paradigms in formal verification of programs is the usage of axiomatic semantics. In axiomatic semantics, the semantics or mathematical meanings of a program is based on the relations between the logic predicates that hold true before and after a piece of code is executed. The program is said to be partially correct if the logic predicates holds throughout the execution of the program. For example, given the simple `while` loop program in figure 2, if we

```
while (x*x>0.5)
      x=0.9*x;
end
```

Fig. 2. A `while` loop Program

assume the value of variable x belongs to the set $[-1, 1]$ before the execution of the `while` loop, then the logic predicate **`x*x<=1`** holds before, during and after the execution of the `while` loop. The predicate that holds before the execution of a block of code is referred to as the pre-condition. The predicate that holds after the execution of a block of code is referred to as the post-condition. Whether a predicate is a pre or post-condition is contextual since its dependent on the block of code that its mentioned in conduction with. A pre-condition for one line of code can be the post-condition for the previous line of the code. A predicate that remains constant i.e. holds throughout the execution of the program is called an *invariant*. For example, the predicate **x*x<=1** is an invariant for the `while` loop. However the predicate **x*x>=0.9** is not an invariant since it only holds during a subset of the total execution steps of the loop.

The invariants can be inserted into the code as comments. We refer to these comments as code specifications or annotations. For example, inserting the predicate **x*x<=1** into the program in figure 2 results in the annotated program in figure 3. The pseudo Matlab specification language used to express the annotations in figure 3 is modelled after ANSI/ISO C Specification Language (ACSL [2]), which is an existing formal specification language for C programs. The pre and post-conditions are denoted respectively using ACSL keywords *requires* and *ensures*. The annotations are captured within comments denoted by the Matlab comment symbol %%. Throughout the rest of the paper, we use this pseudo

```
%% requires x*x<=1;
%% ensures x*x<=1;
while (x*x>0.5)
      x=0.9*x;
end
```

Fig. 3. Axiomatic Semantics for a `while` loop Program

Matlab specification language in the annotations of the example convex optimization program. Other logic keywords from ACSL, such as *exists*, *forall* and *assumes* are also transferred over and they have their usual meanings.

4.2 Hoare Logic

We now introduce a formal system of reasoning about the correctness of programs, that follows the axiomatic semantics paradigm, called Hoare Logic [5]. The main structure within Hoare logic is the Hoare triple. Let P be a pre-condition for the block of code C and let Q be the post-condition for C. We can express the annotated program in 3 as a Hoare triple denoted by $\{P\}\,C\,\{Q\}$, in which both P and Q represent the invariant **x*x<=1** and C is the `while` loop. The Hoare triples is *partially correct* if P hold true for some initial state σ, and Q holds for the new state σ' after the execution of C. For total correctness, we also need prove termination of the execution of C.

Hoare logic includes a set of axioms and inference rules for reasoning about the correctness of Hoare triples for various program structures of a generic imperative programming language. Example program structures include loops, branches, jumps, etc. In this paper, we only consider `while` loops. For example, a Hoare logic axiom for the `while` loop is

$$\frac{\{P \wedge B\}\,C;\{P\}}{\{P\}\,\text{while } B \text{ do } C \text{ done }\;\{\neg B \wedge P\}}. \tag{10}$$

Informally speaking, the axioms and inference rules should be interpreted as follows: the formula above the horizontal line implies the formula below that line. From (10), note that the predicate P holds before and after the `while` loop. We typically refer to this type of predicate as an *inductive invariant*. Inductive invariants require proofs as they are properties that the producer of the code is claiming to be true. For the `while` loop, according the axiom in (10), we need to show that the predicate P holds in every iteration of the loop. In contrast, axiomatic semantics also allows predicates that are essentially assumptions about the state of the program. This is especially useful in specifying properties about the inputs. For example, the variable **x** in figure 3 is assumed to have an value between -1 and 1. The validity of such property cannot be proven since it is an assumption. This type of invariant is referred to as an *assertion*. In our example, the assertion **x<=1 && x>=-1** is necessary for proving that **x*x<=1** is an inductive invariant of the loop.

For this paper, we also use some basic inferences rules from Hoare logic. They are listed in Table 1. The consequence rule in (11) is useful whenever

$$\frac{\{P_1 \Rightarrow P_2\}\, C\, \{Q_1 \Rightarrow Q_2\}}{\{P_1\}\, C\, \{Q_2\}} \quad (11) \qquad \frac{\{P\}\, C_1\, \{Q\}\,;\{Q\}\, C_2\, \{W\}}{\{P\}\, C_1; C_2\, \{W\}} \quad (12)$$

$$\frac{}{\{P\}\, SKIP\, \{P\}} \quad (13) \qquad \frac{}{\{P[e/x]\}\, x := expr\, \{P\}} \quad (14)$$

$$\frac{}{\{P\}\, x := expr\, \{\exists x_0\, (x = expr\, [x_0/x]) \wedge P\, [x_0/x]\ \}} \quad (15)$$

Table 1. Axiomatic Semantics Inference Rules for a Imperative Language

a stronger pre-condition or weaker post-condition is needed. By stronger, we meant the set defined by the predicate is smaller. By weaker, we mean precisely the opposite. The substitution rules in (14) and (15) are used when the code is an assignment statement. The weakest pre-condition $P[x/expr]$ in (14) means P with all instances of the expression *expr* replaced by x. For example, given a line of code **y=x+1** and a known weakest pre-condition **x+1<=1**, we can deduct that **y<=1** is a correct post-condition using the backward substitution rule. Although usually (14) is used to compute the weakest pre-condition from the known post-condition. Alternatively the forward propagation rule in (15) is used to compute the strongest post-condition. The skip rule in (13) is used when executing the piece of code does not change any variables in the pre-condition P.

4.3 Proof Checking

The utility of having the invariants in the code is that finding the invariants is in general more difficult than checking that given invariants are correct. By expressing and translating the high-level functional properties and their proofs onto the code level in the form of invariants, we can verify the correctness of the optimization program with respect to its high-level functional properties using a proof-checking procedure i.e. by verifying each use of a Hoare logic rule.

5 An Interior Point Algorithm and Its Properties

We now describe an example primal-dual interior point algorithm. We focus on the key property of convergence. We show its usefulness in constructing the inductive invariants to be applied towards documenting the software implementation. The algorithm is displayed in Table 2 and is based on the work in [12].

5.1 Details of the Algorithm

The algorithm in Table 2 is consisted of an initialization routine and a `while` loop. The operator **length** is used to compute the size of the input problem data. The operator $\hat{}^{-1}$ represents an algorithm such as QR decomposition that returns the inverse of the matrix. The operator $\hat{}^{0.5}$ represents an algorithm such as Cholesky decomposition that computes the square root of the input matrix. The operator **lsqr** represents a least-square QR factorization algorithm that is used to solve linear systems of equation of the form $Ax = b$. With the assumption of real algebra, all of these operators return exact solutions.

In the initialization part, the states X, Z and p are initialized to feasible values, and the input problem data are assigned to constants $F_i, i = 1, \ldots, m$. The term feasible here means that X, Z, and p satisfies the equality constraints of the primal and dual problems. We discuss more about the efficiency of the initialization process later on.

The `while` loop is a Newton iteration that computes the zero of the derivative of the potential function

$$\phi(X, Z) = \left(n + \nu\sqrt{n}\right) \log \operatorname{Tr}(XZ) - \log\det(XZ) - n \log n, \tag{16}$$

in which ν is a positive weighting factor. Note that the potential function is a weighted sum of the primal-dual cost gap and the barrier function potential. The weighting factor ν is used in computing the duality gap reduction factor $\sigma \equiv \dfrac{n}{n + \nu\sqrt{n}}$. A larger ν implies a smaller σ, which then implies a shorter convergence time. For our algorithm, since we use a fix-step size of 1, a small enough σ combined with the newton step could result in a pair of X and Z that no longer belong to the interior of the positive-semidefinite cone. In the running example, we have $\nu = 0.4714$. While this choice of ν doubled the number of iterations of the running example compared to the typical choice of $\nu = 1$,

Algorithm 1. MZ Short-Path Primal-Dual Interior Point Algorithm

Input: $F_0 \succ 0,\, F_i \in \mathbb{S}^n, i = 1, \ldots, m,\, b \in \mathbb{R}^m$
ϵ: Optimality desired

```
 1. Initialize:
        Compute $Z$ such that $\langle F_i, Z \rangle = -b_i, i = 1, \ldots, m$;
        Let $X \leftarrow \hat{X}$; // $\hat{X}$ is some positive-definite matrix
        Compute $p$ such that $\sum_i^m p_i F_i = -X_0 - F_0$;
        Let $\mu \leftarrow \frac{\langle X, Z \rangle}{n}$;
        Let $\sigma \leftarrow 0.75$;
        Let $n \leftarrow$ length $F_i$, $m \leftarrow$ length $b_i$;
 2. while $n\mu > \epsilon$ {
 3.         Let $\phi_- \leftarrow \langle X, Z \rangle$;
 4.         Let $T_{inv} \leftarrow Z^{0.5}$;
 5.         Let $T \leftarrow T_{inv}^{-1}$;
 6.         Compute $(\Delta Z, \Delta X, \Delta p)$ that satisfies (17);
 7.         Let $Z \leftarrow Z + \Delta Z$, $X \leftarrow X + \Delta X$, $p \leftarrow p + \Delta p$;
 8.         Let $\phi \leftarrow \langle X, Z \rangle$;
 9.         Let $\mu \leftarrow \frac{\langle X, Z \rangle}{n}$;
10.         if $(\phi - \phi_- > 0)$ {
11.             return;
            }
    }
```

Table 2. Primal-Dual Short Path Interior Point Algorithm

however it is critical in satisfying the inductive invariants of the `while` loop that are introduced later this paper.

Let symbol $T = Z^{-0.5}$ and T_{inv} denotes the inverse of T. The `while` loop solves the set of matrix equations

$$\begin{gathered}\langle F_i, \Delta Z\rangle = 0 \\ \sum_i^m \Delta p_i F_i + \Delta X = 0 \\ \frac{1}{2}\left(T\left(Z\Delta X + \Delta Z X\right)T_{inv} + T_{inv}\left(\Delta X Z + X\Delta Z\right)T\right) = \sigma\mu I - T_{inv} X T_{inv}.\end{gathered} \tag{17}$$

for the Newton-search directions ΔZ, ΔX and Δp. The first two equations in (17) are obtained from a Taylor expansion of the equality constraints from the primal and dual problems. These two constraints formulates the feasibility sets as defined in Eq (3). The last equation in (17) is obtained by setting the Taylor expansion of the derivative of (16) equal to 0, and then applying the symmetrizing transformation

$$H_T : M \to \frac{1}{2}\left(TMT^{-1} + \left(TMT^{-1}\right)^{\mathrm{T}}\right), T = Z^{-0.5} \tag{18}$$

to the result. To see this, note that derivative of (16) is $\left[XZ - \frac{n}{n+\nu\sqrt{n}}\frac{\mathrm{Tr}\,(XZ)}{n}I\right.$ $\left. ZX - \frac{n}{n+\nu\sqrt{n}}\frac{\mathrm{Tr}\,(XZ)}{n}I\right]$.

The transformation in (18) is necessary to guarantee the solution ΔX is symmetric. The parameter σ, as mentioned before, can be interpreted as a duality gap reduction factor. To see this, note that the 3rd equation in (17) is the result of applying Newton iteration to solve the equation $XZ = \sigma\mu I$. With $\sigma \in (0,1)$, the duality gap $\mathrm{Tr}\,(XZ) = n\sigma\mu$ is reduced after every iteration. The choice of T in (18) is taken from [11] and is called the Monteiro-Zhang (MZ) direction. Many of the Newton search directions from the interior-point method literature can be derived from an appropriate choice of T. The M-Z direction also guarantees an unique solution ΔX to (17). The `while` loop then updates the states X, Z, p with the computed search directions and computes the new normalized duality gap. The aforementioned steps are repeated until the duality gap $n\mu$ is less than the desired accuracy ϵ.

5.2 High-level Functional Property of the Algorithm

The key high-level functional property of the interior point algorithm in 2 is an upper bound on the worst case computational time to reach the specified duality gap $\epsilon > 0$. The convergence rate is derived from a constant reduction in the potential function in (16) [9] after each iteration of the `while` loop.

Given the potential function in (16), the following result gives us a tight upper bound on the convergence time of our running example.

Theorem 1. *Let X_-, Z_-, and p_- denote the values of X, Z, and p in the previous iteration. If there exist a constant $\delta > 0$ such that*

$$\phi(X_-, Z_-) - \phi(X, Z) \geq \delta, \tag{19}$$

then Algorithm 2 will take at most $\mathcal{O}\left(\sqrt{n} \log \epsilon^{-1} \operatorname{Tr}\left(X_0 Z_0\right)\right)$ iterations to converge to a duality gap of ϵ,

For safety-critical applications, it is important for the optimization program implementation to have a rigorous guarantee of convergence within a specified time. Assuming that the required precision ϵ and the problem data size n are known a priori, we can guarantee a tight upper bound on the optimization algorithm if the function ϕ satisfies (19). For the running example, this is indeed true. We have the following result.

Theorem 2. *There exists a constant $\delta > 0$ such that theorem 1 holds.*

The proof of theorem 2 is not shown here for the sake of brevity but it is based on proofs already available in the interior point method literature (see [6] and [16]).

Using theorems 1 and 2, we can conclude that the algorithm in Table 2, at worst, converges to the ϵ-optimal solution linearly. For documenting the `while` loop portion of the implementation, however we need to construct an inductive invariant of the form

$$0 \leq \phi(X, Z) \leq c, \tag{20}$$

in which c is a positive scalar. While the potential function in (16) is useful for the construction of the algorithm in Table 2, but it is not non-negative. To construct an inductive invariant in the form of (20), instead of using (16), consider

$$\phi(X, Z) = \log \operatorname{Tr}\left(XZ\right), \tag{21}$$

which is simply the log of the duality gap function.

Theorem 3. *The function in (21) satisfies theorems 1 and 2.*

An immediate implication of theorem 3 is that $\operatorname{Tr}\left(XZ\right)$ converges to 0 linearly i.e. $\exists \kappa \in (0, 1)$ such that $\operatorname{Tr}\left(XZ\right) \leq \kappa \operatorname{Tr}\left(X_- Z_-\right)$, in which X_- and Z_- are values of X and Z at the previous iteration. Using $\operatorname{Tr}\left(XZ\right)$, we can construct the inductive invariant from (20) and to express the convergence property from theorem 1.

Additionally, there are two other inductive invariants to be documented for the `while` loop. The first one is the positive-definiteness of the states X and Z. We need to show that the initial X and Z belongs to a positive-definite cone. We also need to show that they are guaranteed to remain in that cone throughout the execution of the `while` loop. This inductive property is directly obtained from the constraints on the variables X and Z. It is also important in showing the non-negativeness of the potential function $\operatorname{Tr}\left(XZ\right)$. The second inductive invariant is a constraint on the distance of XZ from the central path defined by μI. This inductive property is expressed by the formula

$$\|XZ - \mu I\|_F \leq 0.3105\mu. \tag{22}$$

The value 0.3105 in (22) is selected to guarantee that inductive invariant in (32) holds. Its method of selection is explained later in this paper.

6 Running Example

The Matlab implementation of the algorithm from Table 2 can be found in figure 4.

6.1 Input Problem

The input data is obtained from a generic optimization problem taken from systems and control. The details of the original problem is skipped here as it has no bearing on the main contribution of this article. We do like to mention that the matrices $F_i, i = 0, \ldots, 3$ are computed from the original problem using the tool Yalmip [7].

6.2 Autocoding of Convex Optimization Algorithms

In this paragraph, we describe some general ideas towards the credible autocoding of convex optimization algorithms. We also want to refer the readers to an existing work by Boyd [8] on the autocoding of convex optimizatino algorithm. first interior point algorithm can be modelled within synchronous language environment. For example, the dynamics of the Newton iteration can be modelled using delays and a feedback loop. The scaling matrix T for the Newton direction or the duality gap reduction parameter σ can be captured by the appropriate choice of gains and sums. For the more complex interior point implementations such as those with heuristics in the predictor, it is far less likely that one can easily construct the model in a synchronous programming language. The variations of the interior point method discussed in this paper is relatively simple with changes in one of the parameters such as the symmetrizing scaling matrix T, the step size α which is defaulted to 1 in the algorithm description, the duality gap reduction parameter σ, etc. If we can have standard templates of optimization models that are parameterized by those values, then we can easily plug in the values and then auto-generate the code. This can be applied to the proof as well, as we can construct proof templates for each major variations of interior point algorithms and then plug-in the appropriate values into the auto-generated proof templates. In the next section, we give an example proof template on the code level in the Hoare-triple style for the Matlab implementation.

6.3 Matlab Implementation

The Matlab implementation has one main difference from the algorithm description. The first is that, in the Matlab implementation, the current values of X, Z, p are assigned to the variables X_-, Z_-, and p_- at the beginning of the `while` loop. Note that the variables X_-, Z_-, and p_- are denoted by **Xm**,

Zm and **pm** respectively in the Matlab code. Because of that, steps 3 to 6 of algorithm in Table 2 are executed with the variables X_-, Z_-, and p_- instead of X, Z, and p. Accordingly, step 7 becomes $Z \leftarrow Z_- + \varDelta Z_-$, $X \leftarrow X_- + \varDelta X_-$, $p \leftarrow p_- + \varDelta p_-$. This difference is the result of the need to have the invariant of the form $\phi(X, Z) \leq \kappa\phi(X_-, Z_-)$ for $\kappa \in (0, 1)$, which is important since it expresses the fast convergence property from theorem 1.

7 Annotated Matlab Implementation

Now we want to express the inductive invariants and their proofs as Hoare triples [22] on the Matlab implementation. For the reason of compactness, we have chosen to annotate a Matlab implementation of the algorithm.

7.1 Preliminaries

The annotations are expressed using a pseudo Matlab specification language that is analogous to the ANSI/ISO C Specification Language (ACSL) for C programs. We use overloaded operators in the annotations such as $>$ to denote $\succ$ as well as its regular meaning with scalars. Variables are referenced using either their mathematical symbols such as ϕ, ϕ_-, X, and X_- or their corresponding Matlab names **phi**, **phim**, **X**, and **Xm** Likewise code and invariants are expressed using the Matlab language i.e. **phim=trace(Xm*Zm)/n** or their equivalent mathematical representation $\phi_- = \dfrac{\mathrm{Tr}\left(X_- Z_-\right)}{n}$. The implementation calls three functions **vecs**, **mats** and **krons**. These functions are used to convert the matrix equations from (17) into matrix vector equations in the form $Ax = b$. This type of transformation is commonly used in algorithms in which solutions to matrix equations are needed. For more details, please refer to the appendix.

7.2 Annotations For the Initialization

The implementation is consisted of two parts. The first part is the initialization routine. This part defines all the constants required for the formulation of the problem, and initializes the states. The constants are **F0, F1,F2,F3,b** which corresponds to the symbols $F_0, F_i, i = 1, 2, 3$ from the algorithm. The states are X, Z, and p corresponds to the variables **X,Z,p** in the Matlab code. The second part of the implementation is a `while` loop, in which its execution generates a trace $(X(k), Z(k), p(k)), k \in \mathbb{N}$ until $\mathrm{Tr}\,(X(k)Z(k)) \leq n\epsilon$. We first discuss the annotations in the initialization portion. The annotations of high-level functional properties, namely the convergence property from theorems 1 and 2 is discussed in the ensuing section on the `while` loop.

In the first part of the initialization process, the constants of the input problem are defined. These constants need to satisfy certain regularity conditions such as symmetry or positive-definiteness. For example, in line 2, after we assign the matrix $\begin{bmatrix} 1.0 & 0 \\ 0.0 & 0.1 \end{bmatrix}$ to **F0**, a correct post-condition is that **F0** is positive-definite.

```
%% Example SDP Code: Primal-Dual Short-Step Algorithm
F0=[1, 0; 0, 0.1];
F1=[-0.750999 0.00499; 0.00499 0.0001];
F2=[0.03992 -0.999101; -0.999101 0.00002];
F3=[0.0016 0.00004; 0.00004 -0.999999];
b=[0.4; -0.2; 0.2];
n=length(F0);
m=length(b);
Alpha1t=[vecs(F1), vecs(F2), vecs(F3)];
Alpha1=Alpha1t';
Z=mats(lsqr(Alpha1,-b),n);
X=[0.3409 0.2407; 0.2407 0.9021];
P=mats(lsqr(Alpha1t,vecs(-X-F0)),n);
p=vecs(P);
epsilon=1e-8;
sigma=0.75;
phi=trace(X*Z);
mu=trace(X*Z)/n;
while (phi>epsilon)
    Xm=X;
    Zm=Z;
    pm=p;
    phim=trace(Xm*Zm);
    mu=trace(Xm*Zm)/n;
    Zh=Zm^(0.5);
    Zhi=Zh^(-1);
    Alpha2=krons(Zhi,Zh'*Xm,n,m);
    Alpha3=krons(Zhi*Zm,Zh',n,m);
    rz=zeros(m,1);
    rp=zeros(m,1);
    rh=sigma*mu*eye(n,n)-Zh*Xm*Zh;
    dZm=lsqr(Alpha1,rz);
    dXm=lsqr(Alpha3, vecs(rh)-Alpha2*dZm);
    dpm=lsqr(Alpha1t,rp-dXm);
    p=pm+dpm;
    X=Xm+mats(dXm,n);
    Z=Zm+mats(dZm,n);
    phi=trace(X*Z);
    mu=trace(X*Z)/n;
    if (phi-phim>0)
         break;
    end
 end
```

Fig. 4. Matlab Implementation

The symmetry of a matrix is expressed using the Matlab **transpose** function. This function has its usual meaning. We do not explicitly define it here in this paper. The function **smat** used in the annotation on line 9 of figure 5 is the inverse of the function **svec**, where **svec** is similar in every aspect to the function **vecs** but with the factor $\sqrt{2}$ replaced by 2. For example the code **smat(b)** returns the matrix

$$\begin{bmatrix} 0.4 & -0.1 \\ -0.1 & 0.2 \end{bmatrix}. \tag{23}$$

The correctness of the Hoare triples at lines 1, 3, 5, 7, 9 from figure 5 need to

```
1  % ensures F0>0;
2  F0=[1, 0; 0, 0.1];
3  % ensures transpose(F1)==F1;
4  F1=[-0.750999 0.00499; 0.00499 0.0001];
5  % ensures transpose(F2)==F2;
6  F2=[0.03992 -0.999101; -0.999101 0.00002];
7  % ensures transpose(F3)==F3;
8  F3=[0.0016 0.00004; 0.00004 -0.999999];
9  % ensures smat(b)>0;
10 b=[0.4; -0.2; 0.2];
```

Fig. 5. Input Problem Data

proven as the validity of each of the post-conditions are needed to ensure that the input optimization problem is well-posed. For example, if one of the F_i is not symmetrical, then the solution ΔZ to the third equation of (17) is not necessarily symmetrical. The next part of the initialization process computes the sizes of the problem. In our example, m denotes the number of equality constraints in the dual formulation and n denotes the dimensions of the semi-definite variables X and Z. In figure 6, the post-conditions are the requirement that the problem

```
% ensures n>=1;
n=length(F0);
% ensures m>=1;
m=length(b);
```

Fig. 6. Input Problem Sizes

sizes should be at least one.

The next section of the initialization process computes feasible initial conditions for the states Z, X, and p. A feasible Z is computed from solving a system of linear equations formulated using the m affine constraints from the dual problem. The variable X on the other hand can be initialized to any positive-definite matrix. In our running example, the variable is initialized to a value that satisfies the property

$$\|XZ - \mu I\|_F \leq 0.3105\mu. \tag{24}$$

This property is one of the inductive invariants discussed later in this paper.

Remark 1. The property in (24) is a constraint on the set of points belonging to the interior of the positive semi-definite cone in which X can be initialized to. Essentially, it guarantees that X is initialized to within a neighborhood of the central path i.e. not initialized too close to the boundary of the semi-definite cone, which for some optimization problems, means a large enough convergence time to present trouble in a real-time optimization setting. Efficient methods exist in the interior point method literature (see [10]) to guarantee initialization within a certain small neighborhood of the central path. For the sake of brevity, the details are skipped here.

Using this initial X, we can compute an initial p that satisfies the feasible set defined by the matrix equality constraint $F0 + \sum_{i}^{m} p_i F_i + X = 0$ from the primal problem. An important property of the variables X and Z is their positive-

```
1 Alpha1t=[vecs(F1), vecs(F2), vecs(F3)];
2 Alpha1=Alpha1t';
3 % ensures Z>0;
4 Z=mats(lsqr(Alpha1,-b),n);
5 % ensures X>0;
6 X=[0.3409 0.2407; 0.2407 0.9021];
7 P=mats(lsqr(Alpha1t,vecs(-X-F0)),n);
8 p=vecs(P);
```

Fig. 7. Initialization of the Optimization Variables

definiteness, which is expressed by the post-conditions shown in lines 3 and 5 of figure 7.

The last portion of the initialization code is shown in figure 8. The code assigns the desired optimality 1×10^{-8} to the variable **epsilon**, and computes the initial normalized duality gap **mu** and the initial potential **phi**. By using the skip rule from (13), we can propagate forward the invariants $X \succ 0$ and $X \succ 0$ from figure 7 to the end of the code in figure 8. By using the consequent rule from (11), and the fact that $X \succ 0 \wedge Z \succ 0 \implies \operatorname{Tr}(XZ) \geq 0 \implies \exists c > 0, \operatorname{Tr}(XZ) \leq c$, we get the pre-conditions in line 3 of figure 8. Since line 5 is a statement that assigns the expression $\operatorname{Tr}(XZ)$ to the variable ϕ, we can apply the backward substitution rule from (14) here. We get the post-conditions **phi<=c** and **phi>=0**, which are

```
1 epsilon=1e-8;
2 sigma=0.75;
3 % requires exists c>0 && trace(XZ)<=c && trace(XZ)>=0;
4 % ensures phi<=c && phi>=0;
5 phi=trace(X*Z);
```

Fig. 8. Duality Gap and the Initial Potential

displayed in line 4 of figure 8. The positive constant c remains symbolic in our annotated example. In a more realistically annotated code, a numerical value maybe assigned to c. Now we are ready to examine the `while` loop portion of the optimization code.

7.3 Annotations for the `while` loop

The post-condition

$$0 \leq \phi \leq c \tag{25}$$

in line 4 of figure 8 is precisely the inductive invariant from (20) obtained using theorem 3. Invoking the while axiom from Hoare logic, we insert **phi<=c** and **phi>=0** as both a pre and post-conditions for the `while` loop. They are displayed in figure 9.

We also have the pre-conditions $X \succ 0$ and $Z \succ 0$ that is propagated, using the skip rule, from the annotated code in figure 7. We claim that these two are also inductive invariants for the `while` loop, which is to be proven later. They appear in figure 9 as both pre and post-conditions of the `while` loop. For now we assume that $X \succ 0$ and $Z \succ 0$ are true, which implies that **phi>=0** is true. This ends the proof for the inductive invariant **phi>=0**. Now we move on to provide

```
1 % requires phi<=c && c>0 && phi>=0 && X>0 && Z>0;
2 % ensures  phi<=c && c>0 && phi>=0 && X>0 && Z>0;
3 while (phi>epsilon) {
4      .
5      .
6      .
7      .
8 }
```

Fig. 9. Invariants for the Main Loop

a proof of the inductive invariant **phi<=c** in the form of Hoare triples.

We begin the proof by propagating forward the invariant $\mathrm{Tr}(XZ) \leq c$ from line 3 of figure 8 using the skip rule. This results in the pre-condition

trace(X*Z)<=c in line 1 of figure 10. Using the backward substitution rules through lines 2 to 5, we obtain the pre-condition **trace(Xm*Zm)<=c** on line 5 of figure 10. The next line of code assigns the expression $\mathrm{Tr}\,(X_{-}Z_{-})$ to the variable ϕ_{-}. Apply the substitution rule again, we obtain the post-condition **phim<=c** displayed in line 6 of figure 10. Since the variable **phim** is not changed by any line of code after line 5, by applying the skip rule, we obtain the post-condition **phim<= c** in line 11 of 10. Now to show that **phi<=c** is also valid post-condition for the `while` loop, we insert a predicate

$$\phi - \phi_{-} < 0 \tag{26}$$

for the code in line 15 of figure 10. This is displayed in line 13 of figure 10. It is clear that **phim<=c** and **phi-phim<0** implies **phi<=c**. If the predicate in (26) holds true, then by consequent rule from (11), the post-condition **phi<=c** is correct. Note that the condition in (26) is equivalent to the condition in (19). Now it is only necessary to show that **phi - phim** $<$ **0** holds true.

```
1       % requires trace(X*Z)<=c;
2       Xm=X;
3       Zm=Z;
4       pm=p;
5       % requires trace(Xm*Zm)<=c;
6       % ensures phim<=c;
7       phim=trace(Xm*Zm);
8       .
9       .
10      .
11      .
12      % ensures phim<=c;
13      % ensures phi-phim<0;
14      % ensures phi<=c;
15      phi=trace(X*Z);
```

Fig. 10. Loop Body

Reduction in the Duality Gap To prove that the quantity $\mathrm{Tr}\,(XZ)$ is decreasing i.e. **phi-phim<0**, we start with the simple fact that if $\langle X, Z\rangle - \sigma\langle X_{-}, Z_{-}\rangle = 0$, with $\sigma \in (0, 1)$, then $\langle X, Z\rangle - \langle X_{-}, Z_{-}\rangle < 0$.

We also assume for now that $\mathrm{Tr}\,(XZ) - 0.75\,\mathrm{Tr}\,(X_{-}Z_{-}) = 0$ holds as a post-condition for line 20 of figure 11. We also have $\mathrm{Tr}\,(X_{-}Z_{-}) = \phi_{-}$, which holds true because of line 7 of figure 10. This means we have $\mathrm{Tr}\,(XZ) - 0.75\,\mathrm{Tr}\,(X_{-}Z_{-}) = 0 \wedge \mathrm{Tr}\,(X_{-}Z_{-}) = \phi_{-} \implies \mathrm{Tr}\,(XZ) - \phi_{-} < 0$. By the consequent rule, we obtain the pre-condition **trace(X*Z)-phim<0**, which is displayed in line 21 of figure 11. We move forward to the next line of code, which is Line 25 figure 11. It

```
    % require phim<=c;
    % ensures n*mu==trace(Xm*Zm);
    mu=trace(Xm*Zm)/n;
    Zh=Zm^(0.5);
    Zhi=Zh^(-1);
    Alpha2=krons(Zhi,Zh'*Xm,n,m);
    Alpha3=krons(Zhi*Zm,Zh',n,m);
    rz=zeros(m,1);
    rp=zeros(m,1);
    rh=sigma*mu*eye(n,n)-Zh*Xm*Zh;
    dZm=lsqr(Alpha1,rz);
    dXm=lsqr(Alpha3, vecs(rh)-Alpha2*dZm);
    % ensures trace(mats(dXm,n)*mats(dZm,n))==0;
    % ensures trace(Xm*mats(dZm,n))+trace(mats(dXm,n)*Zm)==
        sigma*n*mu-trace(Xm*Zm);
    dpm=lsqr(Alpha1t,rp-dXm);
    % require trace((Xm+mats(dXm,n))*(Zm+mats(dZm,n)))-0.75*
        trace(Xm*Zm)==0;
    p=pm+dpm;
    X=Xm+mats(dXm,n);
    % ensures trace(X*Z)-0.75*trace(Xm*Zm)==0;
    Z=Zm+mats(dZm,n);
    % requires trace(X*Z)-phim<0;
    % requires trace(X*Z) < 0.76*phim
    % ensures phi-phim<0;
    % ensures phi<0.76*phim;
    phi=trace(X*Z);
    mu=trace(X*Z)/n;
```

Fig. 11. Proof of Decreasing Duality-Gap

is the assignment statement and by applying the backward substitution rule on the inserted post-condition **phi-phim<0**, we get exactly the pre-condition **trace(X*Z)-phim<0**. However this does not complete the proof, since we still need to show that the post-condition in line 20 of figure 11 is true. For that we start at the first line of code of figure 11, which is an statement that assigns the quantity $\frac{1}{n}\langle X_-, Z_-\rangle$ to the variable μ. We insert an appropriate post-condition $n\mu = \operatorname{Tr}(X_- Z_-)$. This corresponds to the annotation **n*mu==trace(Xm*Zm)** shown in line 2 of figure 11. This post-condition remains true for the rest of the loop body.

We move on to the next block of the code in figure 11. By examining the lines 4 to 15 of the code, we can determine that this part of the program compute the search directions ΔX_-, ΔZ_- and Δp_- by solving the linear equations listed in 17. We have the post-conditions

$$\langle \Delta X_-, \Delta Z_-\rangle = 0 \tag{27}$$

and

$$\operatorname{Tr}(X_- \Delta Z_-) + \operatorname{Tr}(Z_- \Delta X_-) = \sigma n\mu - \operatorname{Tr}(X_- Z_-) \tag{28}$$

that holds true for the block of code from lines 4 and 15. They are displayed as annotations in lines 13 and 14 of figure 11. To show that the post-condition $\langle \Delta X_-, \Delta Z_-\rangle = 0$ is true, we have the following theorem.

Lemma 1. *If ΔZ_- satisfies the equation $\langle F_i, \Delta Z_-\rangle = 0$, and ΔX_- satisfies the equation $\sum_i^m \Delta p_{-i} F_i + \Delta X_- = 0$, then $\langle \Delta X_-, \Delta Z_-\rangle$ is also zero.*

To see that lemma 1 holds, note that $\langle \Delta X_-, \Delta Z_-\rangle = \langle -\sum_i^m p_{i-} F_i, \Delta Z_-\rangle = -\sum_i^m \Delta p_{i-} \langle F_i, \Delta Z_-\rangle = 0$.

Now we show the post-condition from (28) is also true. We already know lines 11 and 12 figure 11 computes ΔZ_- and ΔX_- that satisfies the equation

$$\begin{aligned}\frac{1}{2}\left(T\left(Z_- \Delta X_- + \Delta Z_- X_-\right) T_{inv} + T_{inv}\left(\Delta X_- Z_- + X_- \Delta Z_-\right) T\right)\\ = \sigma\mu I - T_{inv} X_- T_{inv}.\end{aligned} \tag{29}$$

Taking the trace of both sides of (29), we obtain the following equation

$$\langle X_-, \Delta Z_-\rangle + \langle Z_-, \Delta X_-\rangle = \operatorname{Tr}\left(\sigma\mu I - Z_-^{\frac{1}{2}} X_- Z_-^{\frac{1}{2}}\right) \tag{30}$$

With (30), we can state that $\operatorname{Tr}(X_- \Delta Z_-) + \operatorname{Tr}(Z_- \Delta X_-)$ is equivalent to $\operatorname{Tr}\left(\sigma\mu I - Z_-^{\frac{1}{2}} X_- Z_-^{\frac{1}{2}}\right) = \sigma n\mu - \langle X_-, Z_-\rangle$.

We now move forward to lines 17 to 20 of the annotated code in figure 11. This next block of code updates the states X, Z, p with the computed Newton steps. For this block of code, we start with the pre-condition

$$\langle X_-, Z_-\rangle + \langle X_-, \Delta Z_-\rangle + \langle Z_-, \Delta X_-\rangle + \langle \Delta X_-, \Delta Z_-\rangle = \sigma n\mu, \tag{31}$$

which is obtained from summing the post-conditions in (27) and (28) with the term $\langle X_-, Z_-\rangle$. Note that the post-condition $n\mu = \mathrm{Tr}\,(X_- Z_-)$ from line 2 of figure 11 still holds true at line 17, so we have $\langle X_- + \Delta X_-, Z_- + \Delta Z_-\rangle = \sigma n \mu \implies \langle X_- + \Delta X_-, Z_- + \Delta Z_-\rangle - \sigma\langle X_-, Z_-\rangle = 0$. By the consequent rule and since $\sigma = 0.75$ by line 2 of figure 8, we get the post-condition $\langle X_- + \Delta X_-, Z_- + \Delta Z_-\rangle - 0.75 < X_-, Z_- >= 0$, which is displayed as pre-condition for line 16 of figure 11. Next we apply the backward substitution rule on the post-condition $\langle X, Z\rangle - 0.75 < X_-, Z_- >= 0$ in line 19 of figure 11, we get the desired pre-condition from line 16. We now have finished the analysis of the inductive invariant **phi<=c**.

Before we move on to the next section, note that the post-condition $\langle X, Z\rangle - 0.75 < X_-, Z_- >= 0$ also implies that $\mathrm{Tr}\,(XZ) < 0.76\phi_-$, which is another valid pre-condition for line 25 of figure 11. Using the substitution rule on $\mathrm{Tr}\,(XZ) < 0.76\phi_-$, we get the post-condition $\phi < 0.76\phi_-$, which is displayed in line 24 of figure 11. This invariant, in conjunction with the `while` loop's termination condition $\phi <= \epsilon$, can be used to prove termination of the `while` loop within the bounded time specified in theorem 1.

Positive-Definiteness of *X* and *Z* To show that $X \succ 0$ and $Z \succ 0$ are valid inductive invariants, we use some results from [12]. Note that some of these results are posted throughout this section without proof for the sake of brevity. First, we have a norm bound on the distance of XZ from the central path expressed using the invariant

$$\|XZ - \mu I\|_F \leq 0.3105\mu, \tag{32}$$

As discussed in the previous section, the variable X is initialized to a value such that the condition in (32) is satisfied. Assume that (32) is true, we have the pre and post-condition **norm(X*Z-mu*eye(2,2),'fro')<=0.3105*mu** for the `while` loop. They are displayed respectively on line 1 and 2 of figure 12. We also of course

```
1 % requires norm(X*Z-mu*eye(2,2),'fro')<=0.3105*mu && X>0 && Z
     >0;
2 % ensures norm(X*Z-mu*eye(2,2),'fro')<=0.3105*mu && X>0 && Z
     >0;
3 while (phi>epsilon) {
4       .
5       .
6       .
7       .
8 }
```

Fig. 12. Positive-Definiteness of X and Z as Inductive Invariants

have $X \succ 0$ and $Z \succ 0$ as the other inductive invariants of the `while` loop.

To prove the inductive invariants in figure 12, we examine the body of the `while` loop which is displayed in figure 13. We first insert the post-conditions $X_- \succ 0$ and $Z_- \succ 0$ respectively for lines 2 and 4 of figure 13. Note that by using the backward substitution rule twice, those post-conditions becomes the pre-conditions $X \succ 0$ and $Z \succ 0$, which precisely match with the ones inserted at the beginning of the `while` loop. Additionally, the invariant from (32) is also transformed into the post-condition

$$\|X_- Z_- - \mu I\|_F \leq 0.3105\mu, \tag{33}$$

which is displayed in line 7 of figure 13. The post-condition in (33) is important as it implies several more conditions that are vital to proving the correctness of $Z \succ 0$ and $X \succ 0$.

The next part of the code computes the Newton directions. For the code in line 17 of figure 13, we insert the post-condition

$$\|Z_-^{-0.5} \Delta Z_- Z_-^{-0.5}\|_F \leq 0.7. \tag{34}$$

The post-condition in (34) is generated from the post-condition in (33). The value 0.7 is obtained from an over-approximation of the expression $\dfrac{\sqrt{n\left(1-\sigma\right)^2 + 0.3105^2}}{1-0.3105}$. The proof for this result is skipped here and can be found in [12]. We move on to the next line of code, which computes the Newton search direction ΔX_-. Here we insert the post-condition

$$\|Z_-^{-0.5} \Delta X_- \Delta Z_- Z_-^{0.5}\|_F \leq 0.3105\sigma\mu, \tag{35}$$

which is also generated from the post-condition in (33). Next we insert the post-condition

$$\begin{aligned} Z_-^{-0.5}\left(\Delta Z_- X_- + Z_- \Delta X_-\right) Z_-^{0.5} + Z_-^{0.5}\left(X_- \Delta Z_- + \Delta X_- Z_-\right) Z_-^{-0.5} \\ = 2\left(\sigma\mu I - Z_-^{0.5} X_- Z_-^{0.5}\right). \end{aligned} \tag{36}$$

for line 21 of figure 13, which is displayed in line 20. Note that for the invariants in (34), (35), and (36), we have implicitly assumed that the variable **Zh** is equal to $Z_-^{0.5}$ and the variable **Zhi** is equal to $Z_-^{-0.5}$. This is true because of the assignment statements in lines 9 and 10 figure 13. As discussed before, ΔX_- and ΔZ_- is assumed to exactly satisfy the equation in (29). The correctness of post-condition in (36) is verified by multiplying (29) by 2.

The next line of code, which is line 24 of figure 13, updates the state X with the computed Newton step ΔX_-. Using consequent rule, we can insert the pre-condition

$$\begin{aligned} \frac{1}{2}\|Z_-^{-0.5}\left(\left(Z_- + \Delta Z_-\right)\left(X_- + \Delta X_-\right) - \sigma\mu I\right) Z_-^{0.5} + \\ Z_-^{0.5}\left(\left(X_- + \Delta X_-\right)\left(Z_- + \Delta Z_-\right) - \sigma\mu I\right) Z_-^{-0.5}\|_F \\ \leq 0.3105\sigma\mu, \end{aligned} \tag{37}$$

which is generated from the post-condition in (36), for line 24 of 13. The proof for this is as follows. First note that

$$\begin{aligned}Z_-^{-0.5}\left(\left(Z_-+\mathit{\Delta} Z_-\right)\left(X_-+\mathit{\Delta} X_-\right)-\sigma\mu I\right)Z_-^{0.5}+\\ Z_-^{0.5}\left(\left(X_-+\mathit{\Delta} X_-\right)\left(Z_-+\mathit{\Delta} Z_-\right)-\sigma\mu I\right)Z_-^{-0.5}=\\ Z_-^{-0.5}\left(\mathit{\Delta} Z_-X_-+Z_-\mathit{\Delta} X_-\right)Z_-^{0.5}+Z_-^{0.5}\left(X_-\mathit{\Delta} Z_-+\mathit{\Delta} X_-Z_-\right)Z_-^{-0.5}+\\ 2\left(Z_-^{0.5}X_-Z_-^{0.5}-\sigma\mu I\right)+Z_-^{-0.5}\left(\mathit{\Delta} Z_-\mathit{\Delta} X_-\right)Z_-^{0.5}+Z_-^{0.5}\left(\mathit{\Delta} X_-\mathit{\Delta} Z_-\right)Z_-^{-0.5}.\end{aligned} \tag{38}$$

Second, by using the post-condition in (36), we can simplify (38) further to

$$Z_-^{-0.5}\left(\mathit{\Delta} Z_-\mathit{\Delta} X_-\right)Z_-^{0.5}+Z_-^{0.5}\left(\mathit{\Delta} X_-\mathit{\Delta} Z_-\right)Z_-^{-0.5}. \tag{39}$$

Taking the Frobenius norm of (39) and using the post-condition in (35), we get the pre-condition in (37). This ends the proof to show that (36) implies (37).

Lines 24 and 28 updates the states X and Z. We now apply the substitution rule to the weakest pre-condition in (37), and obtain the post-condition

$$\frac{1}{2}\|Z_-^{-0.5}\left(ZX-\sigma\mu I\right)Z_-^{0.5}+Z_-^{0.5}\left(XZ-\sigma\mu I\right)Z_-^{-0.5}\|_F\leq 0.3105\sigma\mu, \tag{40}$$

which is displayed in line 26 of figure 13.

For the code in line 28 of figure 13, we insert a pre-condition

$$Z_-+\mathit{\Delta} Z_-\succ 0, \tag{41}$$

which is generated from the post-condition in (34). To see that (41) is a correct pre-condition, we use the post-condition from (34), which implies that $\|Z_-^{-0.5}\mathit{\Delta} Z_-Z_-^{-0.5}\|_F<1$, and

$$\|Z_-^{-0.5}\mathit{\Delta} Z_-Z_-^{-0.5}\|_F<1 \implies I+Z_-^{-0.5}\mathit{\Delta} Z_-Z_-^{-0.5}\succ 0. \tag{42}$$

Finally, we also have the fact that

$$I+Z_-^{-0.5}\mathit{\Delta} Z_-Z_-^{-0.5}=Z_-^{-0.5}\left(Z_-+\mathit{\Delta} Z_-\right)Z_-^{-0.5}, \tag{43}$$

which combined with (42) implies $Z_-+\mathit{\Delta} Z_-\succ 0$. By backward substitution, we can obtain the weakest pre-condition in (41) from the post-condition $Z\succ 0$. This completes the proof for $Z\succ 0$ being an inductive invariant. Next we show that $X\succ 0$ is also a valid post-condition for the loop body. For line 30 of figure 13, we introduce the invariant

$$\begin{aligned}\|Z^{0.5}XZ^{0.5}-\sigma\mu I\|_F\leq\frac{1}{2}\|Z_-^{-0.5}\left(ZX-\sigma\mu I\right)Z_-^{0.5}+\\ Z_-^{0.5}\left(XZ-\sigma\mu I\right)Z_-^{-0.5}\|_F,\end{aligned} \tag{44}$$

that holds true for any $Z\succ 0$, $Z_-\succ 0$, and symmetric X.

```
% ensures Xm>0;
Xm=X
% ensures Zm>0;
Zm=Z;
pm=p;
phim=trace(Xm*Zm);
% ensures norm(Xm*Zm-mu*eye(2,2),'fro')<=0.3105*mu;
mu=trace(Xm*Zm)/n;
Zh=Zm^(0.5);
Zhi=Zh^(-1);
Alpha2=krons(Zhi,Zh'*Xm,n,m);
Alpha3=krons(Zhi*Zm,Zh',n,m);
rz=zeros(m,1);
rp=zeros(m,1);
rh=sigma*mu*eye(n,n)-Zh*Xm*Zh;
% ensures norm(Zhi*mats(dZm,n)*Zhi,'fro')<=0.7;
dZm=lsqr(Alpha1,rz);
% ensures norm(Zhi*mats(dZm,n)*mats(dXm,n)*Zh,'fro')
    <=0.3105*sigma*mu;
dXm=lsqr(Alpha3, vecs(rh)-Alpha2*dZm);
% ensures Zhi*(dZm*Xm+Zm*dXm)*Zh+Zh'*(Xm*dZm+dXm*Zm)*Zhi
    '==2*(sigma*mu*I-Zh*Xm*Zh);
dpm=lsqr(Alpha1t,rp-dXm);
p=pm+dpm;
% requires 0.5*norm(Zhi*((Zm+mats(dZm,n)*(Xm+mats(dXm,n)-
    sigma*mu*I)*Zh+Zh'*((Xm+mats(dXm,n)*(Zm+mats(dZm,n)-
    sigma*mu*I)*Zhi','fro')<=0.3105*sigma*mu;
X=Xm+mats(dXm,n);
% requires Zm+mats(dZm,n)>0;
% ensures 0.5*norm(Zhi*(X*Z-sigma*mu*I)*Zh+Zh'*(Z*X-sigma
    *mu*I)*Zhi','fro')<=0.3105*sigma*mu;
% ensures Z>0;
Z=Zm+mats(dZm,n);
% requires norm(Z^0.5*X*Z^0.5-sigma*mu*I,'fro')<=0.5*norm
    (Zhi*(X*Z-sigma*mu*I)*Zh+Zh'*(Z*X-sigma*mu*I)*Zhi','
    fro');
phi=trace(X*Z);
% require Z>0;
% requires trace(X*Z)/n=sigma*mu;
% ensures norm(Z^(0.5)*X*Z^(0.5)-mu*I,'fro')<=0.3105*mu;
% ensures Z^0.5*X*Z^0.5>0;
% ensures X>0;
mu=trace(X*Z)/n;
```

Fig. 13. Annotated Loop Body: $X \succ 0$ and $Z \succ 0$

Finally we move to examine the code in line 36, we see that the variable **mu** is updated with the expression $\frac{\mathrm{Tr}\,(XZ)}{n}$. From the previous section, we know $\langle X, Z\rangle - \sigma\langle X_-, Z_-\rangle = 0$ is an invariant. This combined with the fact that the condition $n\mu = \mathrm{Tr}\,(X_- Z_-)$ still holds true before the execution of line 36 implies that

$$\langle X, Z\rangle = n\sigma\mu \tag{45}$$

is a valid pre-condition for line 36 of 13. Using the invariants from (40), (44), the pre-condition in (45), and apply the backward substitution rule, we deduct a necessary post-condition of

$$\|Z^{0.5}XZ^{0.5} - \mu I\|_F \leq 0.3105\mu \tag{46}$$

for lines 36 of figure 13. The post-condition in (46) implies both $\|XZ - \mu I\|_F \leq 0.3105\mu$ and $Z^{0.5}XZ^{0.5} \succ 0$. The former concludes the proof for the inductive invariant $\|XZ - \mu I\|_F \leq 0.3105\mu$. The latter in conjunction with $Z \succ 0$, which is already proven, implies that $X \succ 0$. This concludes the proof for the inductive invariant $X \succ 0$. These post-conditions are displayed in lines 34 and 35 of figure 13.

8 Future Work

In this paper, we introduce an approach to communicate high-level functional properties of convex optimization algorithms and their proofs down to the code level. Now we want to discuss several possible directions of interest that one can explore in the future. On the more theoretical front, we can look at the possibility that there might exist linear approximations to the potential function used in the construction of the invariant. Having linear approximations would possibly allow us to construct efficient automatic decision procedures to verify the annotations on the code level. On the more practical front, we also need to demonstrate the expression of the interior point semantics on an implementation-level language like C rather than the high-level computational language used in this paper. Related to that is the construction of a prototype tool that is capable of autocoding a variety of convex optimization programs along with their proofs down to the code level. There is also a need to explore the verification of those proof annotations on the code level. It is clear that none of the Hoare triple annotations shown in the previous section, even expressed in a more realistic annotation language, can be handled by existing verification tools. Finally, we also need to be able to reason about the invariants introduced in this paper in the presence of the numerical errors due to floating-point computations.

9 Conclusions

This paper proposes the transformation of high-level functional properties of interior point method algorithms down to implementation level for certification purpose. The approach is taken from a previous work done for control systems. We give an example of a primal-dual interior point algorithms and its convergence property. We show that the high-level proofs can be used as annotations for the verification of an online optimization program.

10 Acknowledgements

The authors would like to acknowledge support from the Vérification de l'Optimisation Rapide Appliquée à la Commande Embarquée (VORACE) project, the NSF Grant CNS - 1135955 "CPS: Medium: Collaborative Research: Credible Autocoding and Verification of Embedded Software (CrAVES)" and Army Research Office's MURI Award W911NF-11-1-0046

References

1. F. Alizadeh, J.-P. A. Haeberly, and M. L. Overton. Primal-dual interior-point methods for semidefinite programming: Convergence rates, stability and numerical results. *SIAM Journal on Optimization*, 5:13–51, 1994.
2. P. Baudin, J.-C. Filliâtre, C. Marché, B. Monate, Y. Moy, and V. Prevosto. *ACSL: ANSI/ISO C Specification Language*, 2008. http://frama-c.cea.fr/acsl.html.
3. S. Boyd and L. Vandenberghe. *Convex Optimization*. Cambridge University Press, New York, NY, USA, 2004.
4. C. Helmberg, F. Rendl, R. J. Vanderbei, and H. Wolkowicz. An interior-point method for semidefinite programming. *SIAM Journal on Optimization*, 6:342–361, 1996.
5. C. A. R. Hoare. An axiomatic basis for computer programming. *Commun. ACM*, 12:576–580, October 1969.
6. M. Kojima, S. Shindoh, and S. Hara. Interior-Point Methods for the Monotone Semidefinite Linear Complementarity Problem in Symmetric Matrices. *SIAM Journal on Optimization*, 7(1):86–125, Feb 1997.
7. J. Löfberg. Yalmip : A toolbox for modeling and optimization in MATLAB, 2004.
8. J. Mattingley and S. Boyd. Cvxgen: a code generator for embedded convex optimization. *Optimization and Engineering*, 13(1):1–27, 2012.
9. L. McGovern and E. Feron. Requirements and hard computational bounds for real-time optimization in safety-critical control systems. In *Decision and Control, 1998. Proceedings of the 37th IEEE Conference on*, volume 3, pages 3366–3371 vol.3, 1998.
10. L. K. McGovern. *Computational Analysis of Real-Time Convex Optimization for Control Systems*. PhD thesis, Massachussetts Institute of Technology, Boston, USA, May 2000.
11. R. D. Monteiro and Y. Zhang. A unified analysis for a class of long-step primal-dual path-following interior-point algorithms for semidefinite programming. *Math. Programming*, 81:281–299, 1998.
12. R. D. C. Monteiro. Primal–dual path-following algorithms for semidefinite programming. *SIAM J. on Optimization*, 7(3):663–678, March 1997.
13. Y. Nesterov and A. Nemirovskii. A general approach to the design of optimal methods for smooth convex functions minimization. *Ekonomika i Matem. Metody*, 24:509–517, 1988.

14. Y. Nesterov and A. Nemirovskii. *Self-Concordant functions and polynomial time methods in convex programming.* Materialy po matematicheskomu obespecheniiu EVM. USSR Academy of Sciences, Central Economic & Mathematic Institute, 1989.
15. Y. Nesterov and A. Nemirovskii. *Interior-point Polynomial Algorithms in Convex Programming.* Studies in Applied Mathematics. Society for Industrial and Applied Mathematics, 1994.
16. Y. Nesterov and M. J. Todd. Primal-dual interior-point methods for self-scaled cones. *SIAM Journal on Optimization*, 8:324–364, 1995.
17. S. Richter, C. N. Jones, and M. Morari. Certification aspects of the fast gradient method for solving the dual of parametric convex programs. *Mathematical Methods of Operations Research*, 77(3):305–321, 2013.
18. M. Rinard. Credible compilation. Technical report, In Proceedings of CC 2001: International Conference on Compiler Construction, 1999.
19. P. Roux, R. Jobredeaux, P.-L. Garoche, and E. Feron. A generic ellipsoid abstract domain for linear time invariant systems. In *HSCC*, pages 105–114, 2012.
20. S. C. RTCA. DO-178C, software considerations in airborne systems and equipment certification, 2011.
21. S. C. RTCA. DO-333 formal methods supplement to DO-178C and DO-278A. Technical report, Dec 2011.
22. A. M. Turing. Checking a large routine. In *Report of a Conference on High Speed Automatic Calculating Machines*, pages 67–69, 1949.
23. T. Wang, R. Jobredeaux, and E. Feron. A graphical environment to express the semantics of control systems, 2011. arXiv:1108.4048.
24. T. Wang, R. Jobredeaux, H. Herencia-Zapana, P.-L. Garoche, A. Dieumegard, E. Feron, and M. Pantel. From design to implementation: an automated, credible autocoding chain for control systems. *CoRR*, abs/1307.2641, 2013.
25. S. Zhang. Quadratic maximization and semidefinite relaxation. *Mathematical Programming*, 87(3):453–465, 2000.

11 Appendix

11.1 Vectorization Functions

The function **vecs** is similar to the standard vectorization function but specialized for symmetric matrices. It is defined as, for $1 \leq i < j \leq n$ and $M \in \mathbb{S}^n$,

$$\operatorname{vecs} M = \left[M_{11}, \ldots, \sqrt{2}M_{ij}, \ldots, M_{nn}\right]^{\mathrm{T}}. \tag{47}$$

The factor $\sqrt{2}$ ensures the function **vecs** preserves the distance defined by the respective inner products of $\mathbb{S}^n$ and $\mathbb{R}^{\frac{n(n+1)}{2}}$. The function **mats** is the inverse of **vecs**. The function **krons**, denoted by the symbol $\otimes_{sym}$, is similar to the standard Kronecker product but specialized for symmetric matrix equations. It has the property

$$(Q_1 \otimes_{sym} Q_2) \operatorname{vecs}(M) = \operatorname{vecs}\left(\frac{1}{2}\left(Q_1 M Q_2^{\mathrm{T}} + Q_2 M Q_1^{\mathrm{T}}\right)\right). \tag{48}$$

Let $Q_1 = TZ$ and $Q_2 = T_{inv}$ and $M = \Delta X$, we get

$$(TZ \otimes_{sym} T_{inv}) \operatorname{vecs}(\Delta X) = \operatorname{vecs}\left(\frac{1}{2}(TZ\Delta X T_{inv} + T_{inv}\Delta X Z T)\right). \tag{49}$$

Additionally, let $Q_1 = T$, $Q_2 = XT_{inv}$, and $M = \Delta Z$, we get

$$(T \otimes_{sym} XT_{inv}) \operatorname{vecs}(\Delta Z) = \operatorname{vecs}\left(\frac{1}{2}(T\Delta Z X T_{inv} + T_{inv} X \Delta Z T)\right). \tag{50}$$

Combining (49) and (50), we get exactly the left hand side of the third equation in (17). Given a ΔZ, we can compute ΔX by solving $Ax = b$ for x where

$$\begin{gathered} A = (TZ \otimes_{sym} T_{inv}) \\ \Delta X = \operatorname{mats}(x) \\ b = \operatorname{vecs}(\sigma\mu I - T_{inv} X T_{inv}) - (T \otimes_{sym} XT_{inv}) \operatorname{vecs}(\Delta Z). \end{gathered} \tag{51}$$

11.2 Annotated Code

```
%% Example SDP Code: Primal-Dual Short-Step Algorithm
% ensures F0>0;
F0=[1, 0; 0, 0.1];
% ensures transpose(F1)==F1;
F1=[-0.750999 0.00499; 0.00499 0.0001];
% ensures transpose(F2)==F2;
F2=[0.03992 -0.999101; -0.999101 0.00002];
% ensures transpose(F3)==F3;
F3=[0.0016 0.00004; 0.00004 -0.999999];
% ensures smat(b)>0;
b=[0.4; -0.2; 0.2];
% ensures n>=1;
n=length(F0);
% ensures m>=1;
m=length(b);
Alpha1t=[vecs(F1), vecs(F2), vecs(F3)];
Alpha1=Alpha1t';
% ensures Z>0;
Z=mats(lsqr(Alpha1,-b),n);
% ensures X>0;
% ensures norm(X*Z-mu*eye(2,2),'fro')<=0.3105*mu;
X=[0.3409 0.2407; 0.2407 0.9021];
P=mats(lsqr(Alpha1t,vecs(-X-F0)),n);
p=vecs(P);
epsilon=1e-8;
sigma=0.75;
% require exists c>0 && trace(X*Z)<=c;
% ensures exists c>0 && phi<=c;
phi=trace(X*Z);
% ensures n*mu=trace(X*Z);
```

```
mu=trace(X*Z)/n;
%% requires phi<=c && norm(X*Z-mu*eye(2,2),'fro')<=0.3105*mu
    && X>0 && Z>0;
%% ensures phi<=c && norm(X*Z-mu*eye(2,2),'fro')<=0.3105*mu
    && X>0 && Z>0;
while (phi>epsilon)
    % requires trace(X*Z)<=c;
    % requires n*mu==trace(X*Z);
    % requires X>0;
    % ensures Xm>0;
    Xm=X;
    % requires Z>0;
    % ensures Zm>0;
    % ensures trace(Xm*Zm)<=c;
    % ensures n*mu==trace(Xm*Zm);
    Zm=Z;
    pm=p;
    % ensures phim<=c;
    phim=trace(Xm*Zm);
    % ensures norm(Xm*Zm-mu*eye(2,2),'fro')<=0.3105*mu;
    mu=trace(Xm*Zm)/n;
    Zh=Zm^(0.5);
    Zhi=Zh^(-1);
    Alpha2=krons(Zhi,Zh'*Xm,n,m);
    Alpha3=krons(Zhi*Zm,Zh',n,m);
    rz=zeros(m,1);
    rp=zeros(m,1);
    rh=sigma*mu*eye(n,n)-Zh*Xm*Zh;
    % ensures norm(Zhi*mats(dZm,n)*Zhi,'fro')<=0.7;
    dZm=lsqr(Alpha1,rz);
    % ensures norm(Zhi*mats(dZm,n)*mats(dXm,n)*Zh,'fro')
        <=0.3105*sigma*mu;
    dXm=lsqr(Alpha3, vecs(rh)-Alpha2*dZm);
    % ensures trace(mats(dXm,n)*mats(dZm,n))==0;
    % ensures trace(Xm*mats(dZm,n))+trace(mats(dXm,n)*Zm)
        ==0.75*n*mu-trace(Xm*Zm);
    % ensures Zhi*(dZm*Xm+Zm*dXm)*Zh+Zh'*(Xm*dZm+dXm*Zm)*Zhi
        '==2*(sigma*mu*I-Zh*Xm*Zh);
    dpm=lsqr(Alpha1t,rp-dXm);
    % require trace((Xm+mats(dXm,n))*(Zm+mats(dZm,n)))-0.75*
        trace(Xm*Zm)=0;
    p=pm+dpm;
    % requires 0.5*norm(Zhi*((Zm+mats(dZm,n)*(Xm+mats(dXm,n)-
        sigma*mu*I)*Zh+Zh'*((Xm+mats(dXm,n)*(Zm+mats(dZm,n)-
        sigma*mu*I)*Zhi','fro')<=0.3105*sigma*mu;
    X=Xm+mats(dXm,n);
    % ensures  trace(X*Z)-0.75*trace(Xm*Zm)==0;
    % requires Zm+mats(dZm,n)>0;
    % ensures 0.5*norm(Zhi*(X*Z-sigma*mu*I)*Zh+Zh'*(Z*X-sigma
        *mu*I)*Zhi','fro')<=0.3105*sigma*mu;
    % ensures Z>0;
    Z=Zm+mats(dZm,n);
    % requires trace(X*Z)-phim<0;
    % requires trace(X*Z)<0.76*phim;
    % requires phim<=c;
    % ensures phi-phim<0;
    % ensures phi<0.76*phim;
    % ensures phi<=c;
    phi=trace(X*Z);
    % requires Z>0;
    % requires norm(Z^0.5*X*Z^0.5-sigma*mu*I,'fro')<=0.5*norm
        (Zhi*(X*Z-sigma*mu*I)*Zh+Zh'*(Z*X-sigma*mu*I)*Zhi','
        fro');
    % ensures norm(Z^(0.5)*X*Z^(0.5)-mu*I,'fro')<=0.3105*mu;
    % ensures Z^0.5*X*Z^0.5>0;
    % ensures X>0;
    mu=trace(X*Z)/n;
    if (phi-phim>0)
        break;
    end
end
```