Credible Autocoding of The Ellipsoid Algorithm Solving Second-Order Cone Programs

Raphael Cohen, Eric Féron, Pierre-Loïc Garoche

Credible Autocoding of The Ellipsoid Algorithm Solving Second-Order Cone Programs

Raphael Cohen, Eric Feron and Pierre-Loïc Garoche

***Abstract*— The efficiency of modern optimization methods, coupled with increasing computational resources, has led to the possibility of real-time optimization algorithms acting in guidance of systems. Unfortunately, those algorithms are still seen as new and obscure and are not considered as a viable option for safety critical roles. This paper deals with the formal verification of convex optimization algorithms. Additionally, we demonstrate how theoretical proofs of real-time convex optimization algorithms can be used to describe functional properties at the code level, thereby making it accessible for the formal methods community. In seeking zero-bug software, we use the Credible Autocoding framework. We focused our attention on the Ellipsoid Algorithm solving second-order cone programs (SOCP). The paper also considers floating-point errors and gives a framework to numerically validate the method.**

I. Introduction

Formal verification of optimization algorithms used online within control systems is the sole focus of this research. Recently, such algorithms have been used online with great success for the guidance of dynamical systems, including, autonomous cars [1] and reusable rockets [2]. Thus, powerful algorithms solving optimization problems are already used online, have been embedded on board, and yet still lack the level of qualification required by civil aircraft or manned rocket flight. Automatic code generation for solving convex optimization problems has already been done [3], but does not include the use of formal methods. Likewise, work within the field of model predictive control already exists where numerical properties of algorithms are being evaluated [4]. Nevertheless, this work is only valid for Quadratic Programming and using fixed-point numbers. As well, no formal verification is performed. On the other hand, some contributions already have been made concerning formal verification of control systems [5], [6], but focuses on formal verification and code generation for linear control system and typical feedback control techniques. Research has also been made toward the verification of numerical optimization algorithms [7], yet it remains theoretical and no proof was performed.

Work already exists about formal verification for convex optimization algorithms [8]. This work remains unfortunately very basic. Only a formalization of annotations at code level are presented and no proof was actually performed with the use of formal methods. As well, only an initial formulation of a numerical analysis is given, without concrete result.

The paper is structured as follows: at first, Section II presents backgrounds for convex optimization and axiomatic semantics. Section III focuses on the axiomatization of an optimization problem and the formal verification of the ellipsoid method. Section IV presents a modification of the original ellipsoid method in order to avoid ill-conditioned ellipsoids. Following this, a floating-point analysis is presented in Section V. Finally, Section VI concludes.

Raphael Cohen is a PhD Student in Aerospace Engineering at the Georgia Institute of Technology , Atlanta, GA, USA. raphael.cohen@gatech.edu

Eric Feron Dutton-Ducoffe Professor of Aerospace Engineering at the Georgia Institute of Technology , Atlanta, GA, USA feron@gatech.edu

Pierre-Loïc Garoche is a Research Scientist at Onera – The French Aerospace Lab, Toulouse, France pierre-loic.garoche@onera.fr

II. Preliminaries

A. Second-Order Cone Programming

Optimization algorithms solve a constrained optimization problem, defined by an objective function, the cost function, and a set of constraints to be satisfied:

$$\begin{aligned} \min \quad & f_o(x) \\ \text{s.t.} \quad & f_i(x) \leq b_i \text{ for } i \in [1, m] \end{aligned}$$

This problem searches for $x \in \mathbb{R}^n$, the optimization variable, minimizing $f_o \in \mathbb{R}^n \to \mathbb{R}$, the objective function, while satisfying constraints $f_i \in \mathbb{R}^n \to \mathbb{R}$, with associated bound b_i. A subclass of these problems can be efficiently solved: convex problems. In these cases, the functions f_o and f_i are required to be convex [9]. Here, we only present a specific subset of convex optimization problems: Second-Order Cone Programs. For $x \in \mathbb{R}^n$, a SOCP in standard form can be written as:

$$\begin{aligned} \min \quad & f^T x \\ \text{s.t.} \quad & \|A_i x + b_i\|_2 \leq c_i^T x + d_i \text{ for } i \in [1\ ,\ m] \end{aligned}$$

With: $f \in \mathbb{R}^n,\ A_i \in \mathbb{R}^{n_i \times n},\ b_i \in \mathbb{R}^{n_i},\ c_i \in \mathbb{R}^n,\ d_i \in \mathbb{R}$.

Because we are focusing on SOCP, this work does not include semidefinite programs (SDP). In control

systems, SDP's are mostly used off-line, checking beforehand system's stability [10]. Thus, SOCP represents a trade off between being general and useful for the control community.

B. Axiomatic Semantic and Hoare Logic

Semantics of programs express their behavior. Here, we specify a program using axiomatic semantics. In this case, the semantics can be defined in an incomplete way, as a set of projective statements, ie. observations. This idea was formalized by [11] and then [12] as a way to specify the expected behavior of a program through pre- and post-condition, or assume-guarantee contracts.

Hoare Logic: A piece of code C is axiomatically described by a pair of formulas (P, Q) such that if P holds before executing C, then Q should be valid after its execution. This pair acts as a contract for the function and (P, C, Q) is called a Hoare triple. In our case we are interested in specifying, at code level, algorithm specific properties such as the convergence or feasibility. Software frameworks, such as the Frama-C platform [13], provide means to annotate a source code with these contracts, and tools to reason about these formal specifications. For the C language, ACSL [14] (ANSI C Specification Language), can be used as source comments to specify function contracts, or local annotations such as loop invariants. In this work, we used the WP Frama-C plugin, and the SMT (Satisfiability modulo theories) solver Alt-Ergo to prove properties at code level.

Linear Algebra-based Specification: ACSL also provides means to enrich underlying logic by defining types, functions, and predicates. In its basic version, ACSL contains no predicate related to linear algebra. It is however possible to introduce such notions, supporting the expression of more advanced properties. Figure 1 presents the definition of new ACSL types and formalization of matrix addition. We also wrote a library for operators used in convex optimization, defining *norm, grad, scalar product, det*, etc. Figure 1 presents as well the definition of the vector two norm.

ACSL

```
/*@
type matrix;
type vector;
logic vector vec_of_3_scalar(double *x)
 reads x[0..2];
logic real vector_select(vector x,
 integer i);
logic integer vector_length(vector x);
logic vector vec_add(vector x, vector y);
axiom vec_add_length:
 \forall vector x, y;
  vector_length(x)==vector_length(y)==>
   vector_length(vec_add(x,y) ==
   vector_length(x);
axiom vec_add_select:
 \forall vector x, y, integer i;
  vector_length(x)==vector_length(y)==>
   0 <= i < vector_length(x) ==>
    vector_select(vec_add(x, y), i) ==
    vector_select(x,i)+vector_select(y,i);
*/
```

Fig. 1. Linear Algebra-based ACSL Specification

III. Automatic Annotated Code Generation

A. The Ellipsoid Method Solving SOCP

Let us now recall the main steps of the algorithm detailed in [15]. In the following, we denote $E_k = \text{Ell}(B_k, c_k)$, the ellipsoid computed by the algorithm at the $k-th$ iteration. Throughout the paper, we denote the Ellipsoid $\text{Ell}(B, c)$ by the set:

$$\text{Ell}(B, c) = \{Bu + c : u^T u \leq 1\} \tag{1}$$

Ellipsoid cut: We start the algorithm with an ellipsoid containing the feasible set X, and therefore the optimal point x^*. We iterate by transforming the current ellipsoid E_k into a smaller volume ellipsoid E_{k+1} that also contains x^*. Given an ellipsoid E_k of center c_k, we find a hyperplane containing c_k that cuts E_k in half, such that one half is known not to contain x^*. Finding such a hyperplane is called the *oracle separation* step, *cf.* [15]. In our SOCP setting, this cutting hyperplane is obtained by taking the gradient of either a violated constraint either the cost function. Then, we define the ellipsoid E_{k+1} by the minimal volume ellipsoid containing the half ellipsoid $\hat{E}_k$ that is known to contain x^* that is computed thanks to the Equations (2), (3) and (4). In addition to that, we know an upper bound, γ, of the ratio of $\text{Vol}(E_{k+1})$ to $\text{Vol}(E_k)$ (see Property (1)). Figure 2 illustrates such ellipsoids cuts.

$$c_{k+1} = c_k - 1/(n+1)B_k p \tag{2}$$

and

$$B_{k+1} = \frac{n}{\sqrt{n^2-1}} B_k + \left(\frac{n}{n+1} - \frac{n}{\sqrt{n^2-1}}\right)(B_k p)p^T \tag{3}$$

with:

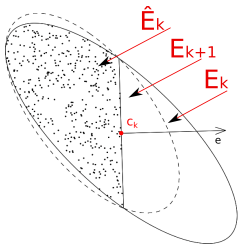

Fig. 2. Ellipsoid Cut

$$p = B_k^T e / \sqrt{e^T B_k B_k^T e}. \quad (4)$$

Property 1: Let $k \geq 0$, by construction:

$$\text{Vol}(E_{k+1}) \leq \exp\{-1/(2 \cdot (n+1))\} \cdot \text{Vol}(E_k)$$

Please find the proof of this property in [16].

Hypotheses: In order to know the number of steps required for the algorithm to return an ϵ-optimal solution, three scalars and a point $x_c \in \mathbb{R}^n$ are needed:

- a radius R such that $X \subset B_R(x_c)$
- a scalar r such that $B_r(x_c) \subset X$
- and V such that $\max_{x \in X} \ f_o(x) - \min_{x \in X} \ f_o(x) \leq V$.

The main result can be stated as:

Theorem 1: Let us assume that X is bounded, not empty and such that R, r and V are known. Then, for all $\epsilon > 0$, the algorithm, using N iterations, will return $\hat{x}$, satisfying:

$$f_o(\hat{x}) \leq f_o(x^*) + \epsilon \text{ and } \hat{x} \in X \ (\epsilon\text{-solution})$$

With $N = 2n(n+1) \log\left(\frac{R}{r}\frac{V}{\epsilon}\right)$, n being the dimension of the optimization problem.

This result, when applied to LP, is historically at the origin of the proof of the polynomial solvability of linear programs. Its proof can be found at [15], [17].

B. Semantics of SOCP

To axiomatize an optimization problem we intend to see it, independently of the method used to solve it, as a pure mathematical object. Our goal is to axiomatize it with enough properties so that when coupling it with the ACSL annotated C code implementation of a solving algorithm, the resulting code would be formally verifiable.

Using ACSL, we define a new type and a high level function, providing the possibility to create objects of the type "*optim*" (Figure 3). When applying a method to solve an actual optimization problem, many concepts are crucial. The work here is to highlight those concepts and write a complete enough library so that when adding the specifications and the semantics of the implementation of a method to this library, the code would be formally verifiable. The concepts of feasibility and optimality are being axiomatized. For instance, please find in Figure 4 the axiomatization of a constraint computation and the feasibility predicate definition.

ACSL

```
/*@ axiomatic OptimSOCP {
type optim;
logic optim socp_of_size_2_6_0(
  matrix A, vector b, matrix C,
  vector d, vector f, int* m)
  reads m[0..5];
logic real constraint(optim OPT,vector x,
  integer i);
logic vector constraints(optim OPT,
  vector x); */
```

Fig. 3. ACSL Optim Type Definition

ACSL

```
/*@
axiom constraint_linear_axiom:
 \forall optim OPT, vector x, integer i;
  getm(OPT)[i] == 0 ==>
   constraint(OPT, x, i) ==
   -scalarProduct(getci(OPT,i),x,
   size_n(OPT))-getdi(OPT,i);
axiom constraint_socp_axiom:
 \forall optim OPT, vector x, integer i;
  getm(OPT)[i] != 0 ==>
   constraint(OPT, x, i) ==
   twoNorm(vector_affine(getAi(OPT,i),
   x,getbi(OPT,i))) -
   scalarProduct(getci(OPT,i),x,
   size_n(OPT))-getdi(OPT,i);
predicate
 isFeasible(optim OPT,vector x) =
 isNegative(constraints(OPT,x)); */
```

Fig. 4. ACSL Feasible Predicate Definition

C. Annotating the Ellipsoid Algorithm

In order to annotate the algorithm and produce a globally formally verifiable code, we adopt a specific technique. Each function will be written in separated files. For each

function, two files are generated: a header and a body file. On the header file (.h), the header of the function and the ACSL contract will be written and on the body file (.c) appears the ACSL annotated C code implementation of the function (updateEllipsoid.c, getGrad.c, initializationEllipsoid.c, ellipsoidMethod.c, etc). Figure 5 displays ACSL contract for the ellipsoid method. To

C Code + ACSL

```
/*@
requires \forall vector x,y;
 isFeasible(OPT(A,b,c), x) ==>
  isFeasible(OPT(A,b,c), y) ==>
   -V <= cost(OPT(A,b,c),x) -
   cost(OPT(A,b,c),y)   <= V;
requires isSubset(
 Feasible(OPT(A,b,c)),
 Ellipsoid( mat_mult_scalar(eye(2),R),
            vec_of_2_scalar(xc)));
requires isSubset(
 Ellipsoid( mat_mult_scalar(eye(2),r),
            vec_of_2_scalar(xc)),
 Feasible(OPT(A,b,c)));
ensures cost(OPT(A,b,c),
 vec_of_2_scalar(x))
 <= cost(OPT(A,b,c),Sol(OPT(A,b,c))+eps;
ensures isFeasible( OPT(A,b,c),
                    vec_of_2_scalar(x));
*/
void ellipsoidMethod();
```

Fig. 5. ellipsoidMethod ACSL Function Contract

keep things simpler, we only presented annotations in the case where we were working with linear programs (only defined by matrix A, vectors b, c) and not with second-order cone programs. As well, we simplified the annotations in order to make it more readable.

Progress: So far, all the matrix-based properties have been proven (addition, multiplication, scalar product, ...). As well, the volume related triples are also proved by the SMT solver Alt-Ergo. The constraint and cutting hyperplane vector computations are proved as well. The loop invariant stating the optimality of c_best is also demonstrated. On the other hand, covering ellipsoids property and the ϵ-optimality of returned solutions is not quite proved by itself for now. These properties being mathematically complex, further work is required in order to have them proved by the SMT solver.

IV. Bounding the Condition Number

Bounding the condition number of B is fundamental and represents the main argument of the algorithm numerical stability. Unfortunately, for the original algorithm, no reasonable bound on $k(B)$ can be found. Therefore, we slightly modified the ellipsoid algorithm to make it able to correct the current ellipsoid E_k in the case where its condition number had become too high (ellipsoid too flat). That way we can control the condition number of B. Also, this correcting step, when it occurs, does not break the convergence of the algorithm and its semantics described in Section III-A.

First let us define the condition number of a matrix:

Definition 1: For a non-singular matrix A of $\mathbb{R}^{n\times n}$, we define the condition number of A as the scalar:

$$k(A) = \|A\| \cdot \left\|A^{-1}\right\| = \sigma_{max}(A)/\sigma_{min}(A) \quad (5)$$

Where σ_i are the singular values of the matrix.

A. Bounding the Singular Values

When updating B_i by the usual formulas of the ellipsoid algorithm, B_i evolves according to

$$B_{i+1} = B_i D_i,$$

where $n-1$ singular values of D_i are $n/\sqrt{n^2-1}$, and one singular value is $n/(n+1)$. It follows that at a single step the largest and the smallest singular values of B_i can change by a factor from $[1/2, 2]$. Let us argue now that one can bound the singular values of the matrix B_i throughout the execution of the program.

a) Minimum Half Axis: First, we claim that if $\sigma_{min}(B)$ is less than than $r\epsilon/V$ then the algorithm has already found an ϵ-solution (we can thus stop the algorithm and return the current best point found). The scalar ϵ being the wanted precision and the scalars r and V being defined in Section III-A (proof in [16]).

b) Maximum Half Axis: When the largest singular value of B_i is less than $2R\sqrt{n+1}$, we carry out a step as in the basic ellipsoid method. When it is greater, we take some time to "correct" B_i in such a way that E_i^+ is a localizer along with E_i , specifically $E_i \cap X \subset E_i^+ \cap X$, and on the top of it:

- The volume of E_i^+ is at most γ times the volume of E_i ;
- The largest singular value of B_i^+ is at most $2R\sqrt{n+1}$.

For this, let us define $\sigma = \sigma_{max}(B_i) > 2R\sqrt{n+1}$ and let e_o being the corresponding direction. We then consider the matrix G such that:

$$G = \text{diag}\left(\sqrt{n/(n+1)},\ \sqrt{n+1}/\sigma,\ \ldots,\ \sqrt{n+1}/\sigma\right)$$

We conclude then this case by performing the below update on B_i and c_i:

$$B_{i+1} = B_i \cdot G \quad \text{and} \quad c_{i+1} = c_i - (e_o^T c_i) \cdot e_o \tag{6}$$

Please find in figure 6 an illustration of such a correction. The unit ball being the feasible set. Hence, we conclude from this that, throughout the execution of the code we have:

$$\sigma_{min}(B_i) \geq \sigma_{min} = 1/2 \cdot r\epsilon/V$$
$$\sigma_{max}(B_i) \leq \sigma_{max} = 4R\sqrt{n+1}$$

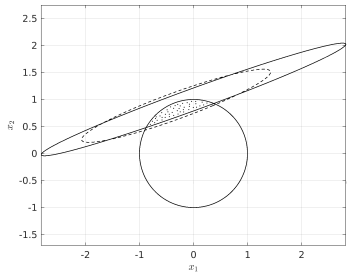

Fig. 6. Correcting the Ellipsoid

B. Corresponding Bounds

By having a lower bound on $\sigma_{min}(B)$ and a upper bound on $\sigma_{max}(B)$, we conclude that throughout the execution of the program:

$$k(B_i) \leq \left(\frac{2}{1/2} \cdot \frac{2R\sqrt{n+1}}{r\epsilon/V}\right) = \frac{8RV\sqrt{n+1}}{r\epsilon} \tag{7}$$

and,

$$\|B_i\|_2 = \sigma_{max}(B) \leq 4R\sqrt{n+1}. \tag{8}$$

At each iteration we know that $x^* \in \text{Ell}(B_i, c_i)$ Which implies that:

$$\|c_i\| \leq R + \|x_c\| + \|B_i\|\,. \tag{9}$$

V. Floating-Point Considerations

Let $\mathbb{F}$ denotes the set of all floating-point numbers and $\mathbb{R}$ the set of reals. We use standard notation for rounding error analysis [18]. We write the relative rounding error unit **u** and the underflow unit **eta**. For IEEE 754 double precision (binary64) we have **u**$= 2^{-53}$ and **eta**$= 2^{-1074}$. We present in this section an analysis targeting the numerical properties of the Ellipsoid Algorithm. Contributions already have been made concerning finite-precision calculations within the ellipsoid method [17]. However, this work only shows that it is possible to compute approximate solutions without giving exact bounds, remains very theoretical and only applied to linear programming. Also, the analysis performed considers abstract finite-precision numbers and floating-points are not mentioned. Thanks to the analysis performed in this section, using the IEEE standard for floating-point arithmetic and knowing exactly how the errors are being propagated, we would be able to check a posteriori the correctness of the analysis using static analyzers [19]. Throughout this section, we work on the modified version of the algorithm presented in Section IV and use the presented bounds.

A. Problem Formulation

To take into account the uncertainties on the variables due to floating-point rounding, we modify the algorithm to make it more robust. For this, we choose to evaluate those uncertainties and conclude on a coefficient λ that represents by how much we are going to widen the ellipsoid E_k at each iteration (see Figure 7). Within this algorithm, we focus our attention on the update formulas (2), (3) and (4). Let us assume we have $B \in \mathbb{F}^{n\times n}$, $p \in \mathbb{F}^n$, $c \in \mathbb{F}^n$. We want to find $\lambda \geq 1 \in \mathbb{R}$ such that:

$$\text{Ell}\Big(B^+, c^+\Big) \subset \text{Ell}\Big(\lambda \cdot \text{fl}(B^+),\ \text{fl}(c^+)\Big). \tag{10}$$

Beforehand, we state an equivalent condition. Its proof can be found in [16].

Lemma 1: [Equivalent Condition]

$$\text{Ell}\Big(B^+, c^+\Big) \subset \text{Ell}\Big(\lambda \cdot \text{fl}(B^+),\ \text{fl}(c^+)\Big) \iff$$

$$\Big\|\text{fl}(B^+)^{-1}\Big(B^+u + c^+ - \text{fl}(c^+)\Big)\Big\| \leq \lambda\ ,\forall\ u \in B_1(0).$$

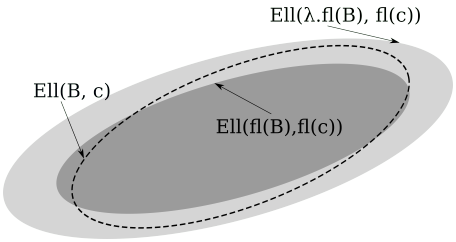

Fig. 7. Ellipsoid Widening

Let us define Δ_B, $\Delta_{B^{-1}}$ and Δ_c representing the floating-point errors, such that:

$$\Delta_B = \text{fl}(B^+) - B^+ \; ; \; \Delta_c = \text{fl}(c^+) - c^+$$
$$\Delta_{B^{-1}} = (\text{fl}(B^+))^{-1} - (B^+)^{-1}$$

and assume that after performing the floating-point analysis we found $\mathcal{E}_B$ and $\mathcal{E}_c$ such that:

$$|(\Delta_B)_{i,j}| \leq \mathcal{E}_B \text{ and } |(\Delta_c)_i| \leq \mathcal{E}_c \quad \forall i,j \in [1,n],$$

We dedicated Section V-B to the computation of $\mathcal{E}_c$ and $\mathcal{E}_B$. Additionally, we need to compute a number $\mathcal{E}_{B^{-1}}$ such that: $|(\Delta_{B^{-1}})_{i,j}| \leq \mathcal{E}_{B^{-1}} \quad \forall i,j \in [1,n]$.
The quantity $(B^+)^{-1}$ is not used explicitly in the algorithm and its floating-point error could not be evaluated by numerically analyzing the computer instructions. Instead, we will use perturbation matrix theory [20] and the theorem 2 below:

Theorem 2: Let A be a non-singular matrix of $\mathbb{R}^{n\times n}$ and ΔA a small perturbation of A. Then, from [20], we know that,

$$\frac{\left\|(A+\Delta A)^{-1} - A^{-1}\right\|}{\|A^{-1}\|} \leq k(A)\frac{\|\Delta A\|}{\|A\|} \qquad (11)$$

This will give us a lower bound on $\mathcal{E}_{B^{-1}}$ given $\mathcal{E}_B$, the norm of B and its condition number. The result is stated in the following lemma (see proof in [16]).

Lemma 2: [Widening - Analytical Sufficient Condition]

$$1 + \frac{n\mathcal{E}_B\sigma_{max} + (\sigma_{min} + n\mathcal{E}_B)\sqrt{n}\mathcal{E}_c}{\sigma^2_{min}} \leq \lambda \implies$$
$$\text{Ell}\Big(B^+, c^+\Big) \subset \text{Ell}\Big(\lambda \cdot \text{fl}(B^+),\ \text{fl}(c^+)\Big).$$

After founding such a λ, we would like to know whether the algorithm is still converging. As well, because the method's proof lies in the fact that the final ellipsoid has a small enough volume, this correction will have an impact of the number of iterations. Lemma 3 addresses those issues. Again, its proof can be found in [16].

Lemma 3: [Convergent Widening Coefficient] Let $n \in \mathbb{N}, n \geq 2$.
The algorithm implementing the widened ellipsoids, with coefficient λ will converge if:

$$\lambda < \exp\{1/(n(n+1))\} \qquad (12)$$

In that case, if N denotes the original number of iteration needed, the algorithm implementing the widened ellipsoids will require:

$$N_\lambda = N/\big(1 - n(n+1)\log(\lambda)\big) \text{ iterations} \qquad (13)$$

B. Floating-Point Rounding of Elementary Transformations

In this section, we express the floating-point errors taking place when performing the update formulas (2) and (3). For this, we present first the error analysis for basic operations appearing in the algorithm.
Rounding of a Real. Let $z \in \mathbb{R}$

$$\tilde{z} = \text{fl}(z) = z + \delta + \eta \qquad \text{with } |\delta| < \mathbf{u} \text{ and } |\eta| < \mathbf{eta}/2$$

Product and Addition of Floating-Points. Let $a, b \in \mathbb{F}$.

$$\begin{aligned} \text{fl}(a \times b) &= (a \times b)(1+\epsilon_2) + \eta_2 \\ \text{fl}(a+b) &= (a+b)(1+\epsilon_1) \end{aligned}$$

with: $|\epsilon_1| < \mathbf{u}$, $|\epsilon_2| < \mathbf{u}$, $|\eta_2| < \mathbf{eta}$ and $\epsilon_2\eta_2 = 0$
Reals-Floats Product. Let $z \in \mathbb{R}$ and $a \in \mathbb{F}$,

$$|\text{fl}\big(\text{fl}(z)\cdot a\big) - z\cdot a| \leq |z||a|\cdot\mathbf{u} + |a|\cdot 2\mathbf{u}(1+\mathbf{u})$$

Following the same technique, we know evaluate the uncertainties taking place on the computations of c^+ and B^+. $\langle .,. \rangle$ denotes the scalar product.

Errors on c^+ and B^+: From Equations (2) and (3), we can see that for each component of c and B, the program performs floating-point operations of the form:

$$\text{fl}\big(c + \text{fl}\big(\text{fl}(z)\cdot\text{fl}\langle a,b\rangle\big) \qquad \text{and}$$
$$\text{fl}\big(\text{fl}\big(\text{fl}(z_1)\cdot d\big) + \text{fl}\big)\text{fl}(z_2)\cdot\text{fl}\big(\text{fl}\langle a,b\rangle\cdot c\big)\big)\big).$$

With: $a, b \in \mathbb{F}^n$, $c, d \in \mathbb{F}$ and $z_1, z_2 \in \mathbb{R}$. Hence, by propagating the errors through the elementary operations, we found:

$$\mathcal{E}_c \leq \mathbf{u} \cdot \big(\big(16n^2 + 16n + 3\big) \cdot \|B\| + \|c\| \big) \tag{14}$$

$$\mathcal{E}_B \leq \mathbf{u}\, \|B\| \cdot \big(\big(\frac{n^2}{1 - n\mathbf{u}} + 2 \big) |\beta| + n + 2|\alpha| + 1 \big) \tag{15}$$

VI. Conclusion

We present an axiomatization of SOCP using the specification language ACSL. In addition to that, annotations for numerical algorithms solving those problems were proposed. We focused our attention on the ellipsoid method. We presented how the process of code generation and code verification for a given optimization problem can be automated. We show how to propagate the errors due to floating-point calculations through the operations performed by the program. As it was said earlier, the proof is not finalized yet and further work is therefore required to finalize the ACSL proof of the algorithm. As well, we only presented the correctness of the optimization process for a given and fixed problem. However, when using this technique for control and hence online, we repeat this process for different initializations. Therefore, one future work includes extracting proof of convergence concerning the online utilization of those algorithms, and finally annotating the code accordingly. This would guarantee a completely sound and bug-free implementation.

Acknowledgments

This work was partially supported by project NSF CPS SORTIES under grant 1446758. The authors would also like to deeply thank Pierre Roux and Arkadi Nemirovski for their help and participation in this work.

References

[1] Paolo Falcone, Francesco Borrelli, Jahan Asgari, Hongtei Eric Tseng, and Davor Hrovat. Predictive active steering control for autonomous vehicle systems. *IEEE Transactions on control systems technology*, 15(3):566–580, 2007.

[2] Lars Blackmore, Behçet Açikmese, and John M. Carson. Lossless convexification of control constraints for a class of nonlinear optimal control problems. *Systems & Control Letters*, 61(8):863–870, 2012.

[3] Jacob Mattingley and Stephen Boyd. Automatic code generation for real-time convex optimization. *Convex optimization in signal processing and communications*, pages 1–41, 2009.

[4] Panagiotis Patrinos, Alberto Guiggiani, and Alberto Bemporad. A dual gradient-projection algorithm for model predictive control in fixed-point arithmetic. *Automatica*, 55:226–235, 2015.

[5] Eric Feron. From control systems to control software. *Control Systems, IEEE*, 30(6):50 –71, December 2010.

[6] Timothy Wang, Romain Jobredeaux, Heber Herencia-Zapana, Pierre-Loïc Garoche, Arnaud Dieumegard, Eric Feron, and Marc Pantel. *From Design to Implementation: An Automated, Credible Autocoding Chain for Control Systems*, volume 460 of *LNCIS*, pages 137–180. Springer, 2016.

[7] Timothy Wang, Romain Jobredeaux, Marc Pantel, Pierre-Loic Garoche, Eric Feron, and Didier Henrion. Credible autocoding of convex optimization algorithms. *Optimization and Engineering*, 17(4):781–812, 2016.

[8] Raphael Cohen, Guillaume Davy, Eric Feron, and Pierre-Loïc Garoche. Formal verification for embedded implementation of convex optimization algorithms. *IFAC-PapersOnLine*, 50(1):5867–5874, 2017.

[9] Stephen Boyd and Lieven Vandenberghe. *Convex optimization*. Cambridge University Press, 2004.

[10] Stephen Boyd, Laurent El Ghaoui, Eric Feron, and Venkataramanan Balakrishnan. *Linear matrix inequalities in system and control theory*. SIAM, 1994.

[11] Robert W. Floyd. Assigning meanings to programs. *Proceedings of Symposium on Applied Mathematics*, 19:19–32, 1967.

[12] Charles. A. R. Hoare. An axiomatic basis for computer programming. *Commun. ACM*, 12:576–580, October 1969.

[13] P. Cuoq, F. Kirchner, N. Kosmatov, V. Prevosto, J. Signoles, and B. Yakobowski. Frama-c: a software analysis perspective. SEFM'12, pages 233–247. Springer, 2012.

[14] P. Baudin, J.-C. Filliâtre, C. Marché, B. Monate, Y. Moy, and V. Prevosto. ACSL: ANSI/ISO C Specification Language. version 1.11., 2016.

[15] Arkadi Nemirovski. Introduction to Linear Optimization, Lecture notes, Georgia Institute of Technology . URL: `http://www2.isye.gatech.edu/~nemirovs/OPTI_LectureNotes.pdf`.

[16] `http://www.prism.gatech.edu/~rcohen30/cdc_cohen_proof.pdf`

[17] Leonid G Khachiyan. Polynomial algorithms in linear programming. *USSR Computational Mathematics and Mathematical Physics*, 20(1):53–72, 1980.

[18] Siegfried M Rump. Error estimation of floating-point summation and dot product. *BIT Numerical Mathematics*, 52(1):201–220, 2012.

[19] Sylvie Putot, Eric Goubault, and Matthieu Martel. Static analysis-based validation of floating-point computations. *Lecture notes in computer science*, pages 306–314, 2004.

[20] Laurent El Ghaoui. Inversion error, condition number, and approximate inverses of uncertain matrices. *Linear algebra and its applications*, 343:171–193, 2002.