AIAA SciTech Forum
9 - 13 January 2017, Grapevine, Texas
AIAA Information Systems-AIAA Infotech @ Aerospace

Validation of Convex Optimization Algorithms and Credible Implementation for Model Predictive Control

Guillaume Davy * and Pierre-Loic Garoche †
Onera - The French Aerospace Lab, Toulouse, FRANCE
Raphael Cohen‡ and Eric Feron §
Georgia Institute of Technology, Atlanta, GA, USA

Advanced real-time embedded algorithms are growing in complexity and length, related to the growth in autonomy, which allows vehicles to plan paths of their own. However, this promise cannot happen without proper attention to the considerably stronger operational constraints that real time, safety-critical applications must meet. This paper discusses the formal verification for optimization algorithms with a particular emphasis on receding-horizon controllers. Following a brief historical overview, a prototype autocoder for embedded convex optimization algorithms is discussed. Options for encoding code properties and proofs, and their applicability and limitations is detailed as well.

I. Introduction

The need for more safety and better performance is currently pushing the introduction of advanced numerical methods into next generations of cyber-physical systems. While most of the algorithms described in this paper have been known for a long time, their online use within embedded systems is relatively new and opens issues that have to be addressed. Among these methods, we are concerned specifically with numerical optimization algorithms. These algorithms solve a constrained optimization problem, defined by an objective function – the cost function – and a set of constraints to be satisfied:

$$\begin{array}{ll} \min & f_0(x) \\ \text{s.t.} & f_i(x) \leq b_i \text{ for } i \in [1, m] \end{array} \tag{1}$$

This problem searches for $x \in \mathbb{R}^n$ the optimization variable, minimizing $f_0 \in R^n \to \mathbb{R}$, the objective function, while satisfying constraints $f_i \in \mathbb{R}^n \to \mathbb{R}$, with associated bound b_i. An element $\mathbb{R}^n$ is feasible when it satisfies all the constraints f_i. An optimal point is defined by the element having the smallest cost value among all feasible points. An optimization algorithm computes an exact or approximated estimate of the value of the optimal cost, together with one or more feasible points achieving this value. A subclass of these problems can be efficiently solved: convex problems. In these cases, the functions f_0 and f_i are required to be convex and the optimal solution is unique. We refer the reader to[1,2] for more details on convex optimization.

Recently this formalism has been used with great success for the guidance of safety-critical application. Such applications include autonomous cars[3] and reusable rockets.[4,5] The latter case has resulted in a spectacular experiments, including landings of SpaceX's Falcon 9[6] and BlueOrigin's New Shepard. Thus, powerful algorithms solving optimization problems do exist and are already used online. Although those algorithms have been embedded on board and running on an actual system, they still lack the level of qualification required by civil aircraft or manned rocket flight. computing a solution in a given time. Let

*Ph.D Student, Onera, Toulouse.
†Research Scientist, Onera, Toulouse.
‡Ph.D Student, School of Aerospace Engineering, 85 Fifth Street NW, Atlanta, GA.
§Professor of Aerospace Engineering, School of Aerospace Engineering, 85 Fifth Street NW, Atlanta, GA.

us consider the specific case of Linear Programming problems, a subclass of convex optimization. Multiple algorithm methods are available to solve them. First, the simplex method, which has been used successfully since the 1940s, despite its exponential worst-case complexity. Then the Ellipsoid algorithm and the very popular interior-point methods. These latter both exhibit polynomial complexity, though only interior point methods are considered to be practical for large desktop applications. These two methods are also applicable to more general settings such as quadratic programming (QP), affine constraints and quadratic cost, second order conic programming (SOCP), quadratic constraints and linear cost, or semidefinite programs (SDP).

When optimization algorithms are used offline, the soundness of their implementation and the feasibility of the computed optimizers is not as critical. Solutions could be validated *a posteriori*.[7,8] This paper is concerned with online real-time use of such algorithms, providing evidence of the *a priori* validity of the computed solution. This paper focuses on the linear programming paradigm and makes the following contributions:

- high level properties defining a sound optimization algorithm are formalized;
- these properties are expressed directly on the code artifact as annotations;
- the evidence supporting the algorithms properties is expressed at the code level
- the approach is demonstrated on the Ellipsoid algorithm and on an instance of an Interior Point method.

We believe this paper addresses a major certification issue that can considerably influence the use of optimization methods in real-time safety-critical applications.

Related work. Several authors including ,[9,10,11] have worked on the certification problem of optimization algorithms for their online uses 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.

Formally verifying high level properties of numerical algorithm implementations was performed on the other context of linear controller stability.[12] This theoretical work was later instantiated concretely, generating code specifications and proving it with respect to the code.[13,14] Some other work develop that approach for weaker properties.[15,16]

Our work follows a similar approach with respect to[17] in which interior-point method algorithms are annotated with convergence proof elements in order to show the soundness of the imperative implementation. This work however, remains theoretical, expressed in Matlab level, without any proof performed on the actual code.

An other solution to online optimization is presented in[18] but it doesn't scale much to big problems as required by MPC and you can't completly generalize to convex optimization yet. Therefore working on interior point method proof is more interesting.

Structure. The paper is structured as follows: Section II presents backgrounds for linear programming and axiomatic semantics using Hoare triples. Section III shows the technique of Credible Autocoding (code generation) into the frame of receding horizong controller. Sections IV and V focus on the two considered approaches: Interior Point method and Ellipsoid method. Section VI presents our early proof results on the code artifact, while Section VII concludes.

II. Preliminaries

In order to support the following analysis, this section introduces the notions and notations used throughout the paper. First, we discuss Linear Programming (LP) problems. Then we introduce axiomatic semantics and Hoare logic.

II.A. Linear Programming

Linear Programming is a class of optimization problems. A linear programming problem is defined by a matrix A of size $m \times n$, and two vectors b, c of respective size m, n.

Definition 1 *Let us consider $A \in \mathbb{R}^{m\times n}$, $b \in \mathbb{R}^m$ and $c \in \mathbb{R}^n$. We write $P(A, b, c)$ as the linear program:*

$$\min_{Ax\geq b} \langle c, x\rangle. \tag{2}$$

Definition 2 *Let us consider the problem $P(A, b, c)$ and let us assume that an optimal point exists and is unique. We have the following definitions:*

- *$E_f = \{x \mid Ax \geq b\}$ the feasible set of P,*
- *$f(x) = c^T x = \langle c, x\rangle$ the cost function and*
- *$x^* = \arg\min_{x\in E_f} f$ the optimal point.*

II.B. Axiomatic semantics and Hoare Logic

Semantics of programs express their behavior. For the same program, different means can be used to specify it: (i) a denotational semantics, expressing the program as a mathematical function, (ii) an operational semantics, expressing it as a sequence of basic computations, or (iii) an axiomatic semantics. In the latter case, the semantics can be defined in an incomplete way, as a set of projective statements, ie. observations. This idea was formalized by[19] and then[20] 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 most uses P and Q are expressed in first order formulas over the variables of the program. Depending on the level of precision of these annotations, the behavior can be fully or partially specified. In our case we are interested in specifying, at code level, algorithm specific properties such as the convergence of the analysis or preservation of feasibility for intermediate iterates.

Software frameworks, such as the Frama-C platform, *cf.*,[21] provide means to annotate a source code with these contracts, and tools to reason about these formal specifications. For the C language, ACSL, *cf.*[22] (ANSI C Specification language) can be used as source comments to specify function contracts, or local annotations such as loop invariants. Statement local annotations act as cuts in proofs and are typically required when analyzing loops.

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 formalization of matrix addition.

```
/*@ logic LMat mat_add(LMat x0, LMat x1);
  @ axiom getM_add: \forall LMat A, B; getM(mat_add(A, B) == getM(A);
  @ axiom getN_add: \forall LMat A, B; getN(mat_add(A, B) == getN(A);
  @ axiom get_add:
  @     \forall LMat A, B; (
  @         \forall integer i, j;
  @             mat_get(mat_add(A, B), i, j) == mat_get(A, i, j)
  @               + mat_get(B, i, j)); */
```

Figure 1: Axiomatization of matrix addition

These definitions are straightforward and fully defined. We also wrote a library for operators used in linear optimization, defining $norm, grad, hess$, lower.... Figure 2 presents the definition a Hessian-induced norm, parametrized by the Hessian at point X.

Lemmas can also be defined to support later analyzers to prove annotations.To preserve proof consistency, the Frama-C tool requires to prove these additional lemmas. For example, splitting a complex theorem into several small lemmas usually helps analyzers in proving the whole goal.

```
/*@ logic real norm(LMat V, LMat X) =
  @     sqrt(mat_get(mat_mul(
  @         mat_mul(transpose(V), inv(hess(X)))
  @     , V), 0, 0)); */
```

Figure 2: Hessian norm in ACSL.

III. Model Predictive Contol and Code Generation

III.A. Model Predctive Contol

Model Predictive Control (also known as receding horizon control) is an optimal control strategy based on numerical optimization. In this technique, a dynamical model of the plant is being used to predict potential future trajectories. As well, a cost function J, that depends on the potential future control input and state, is being considered over the receding prediction horizon N and the objective here is to minimize this cost J. At each time t, a Convex Optimization problem where J has to be minimized, is being solved. From the solution of this problem, an optimal trajectory starting at $x(t)$ is being calculated and the control input sent to the plant correspond to the first input of this optimal trajectory. A time step later, at $t + \Delta t$, the exact same process occurs and is repeated until final time. Fig 3 illustrates this framework.

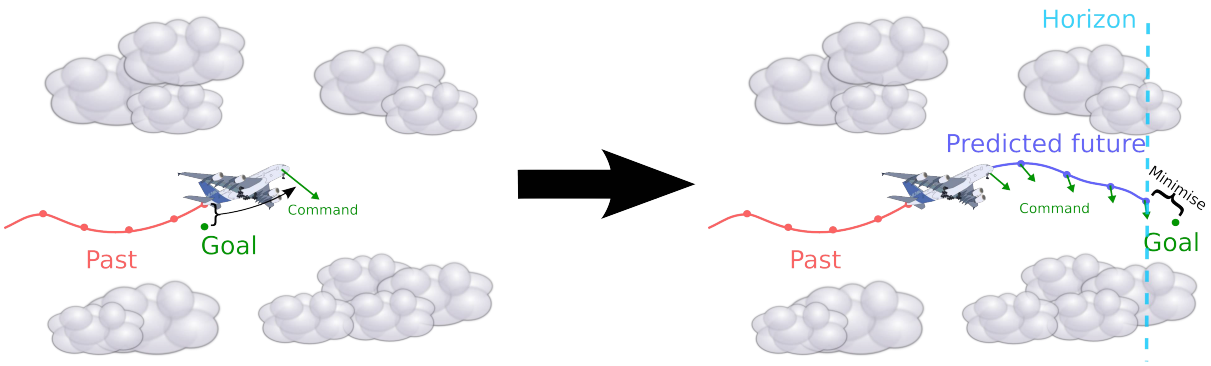


Figure 3: How MPC improve control

In order to simplify the problem, without lost of generality, we consider the MPC that stabilizes the origin of a Linear Time Invariant (LTI) system. Please find in figure 4 a standard formulation of such a MPC. We refer the reader to[1] for more details on MPC.

$$
\begin{aligned}
\text{minimize} \quad & \sum_{k=0}^{N} J(x_k, u_k) \\
\text{subject to} \quad & u_k \in \mathcal{U} \ , \ x_k \in \mathcal{X} \ , \ k = 0, \dots, N \\
& x_{k+1} = Ax_k + Bu_k \ , \ k = 1, \dots, N-1 \\
& x_0 = x(t) \ , \ x_N = 0
\end{aligned}
$$

Figure 4: Standard MPC Formualtion Stabilizing the Origin

Once this formulation has been done, one can see the control below as a function taking $x(t)$ as an input and $J, A, B, N, \mathcal{U}, \mathcal{X}$ as parameters. Please find the block diagram representing the closed loop system in figure 5.

The optimization algorithm being at the heart of the control the stability of the whole system depends on its correctness. Checking the soundess of these algorithm is then a crutial work and has to be adressed. For reference about stality of MPC see[23,24]

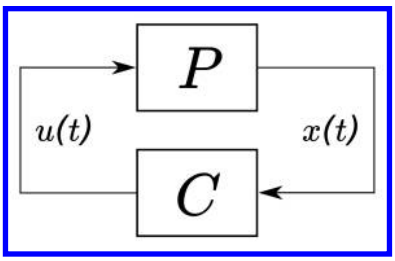


Figure 5: Feedback System

III.B. Credible Implementation for Model Predictive Control

Credible autocoding, is a process by which an implementation of a certain input model in a given programming langage is bein generated along with formally verifiable evidence that the output source code correctly implements the input model. Existing work already provides for the automatic generation of embedded convex optimization code.[25] Given that, proofs of high-level functional properties of convex optimizations 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.

Within our work, an autocoder has been built that generates C code implementations of Convex Optimization solvers along with annotations. Those annotations representing the proof of soundness of these algorithms. Sections IV and V present in details the two solvers used and their annotations.
The autocoder takes as an input a text file containing the MPC formulation, described as in figure 4. Figures 6 and 7 illustrate the framework and the internal aspect of the autocoder.

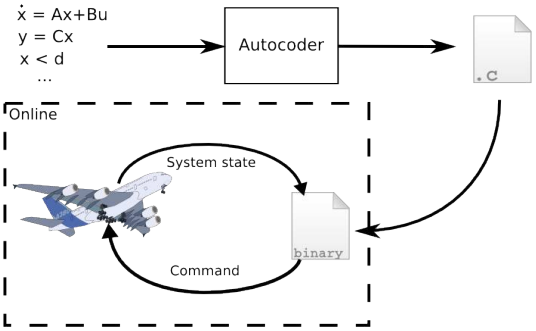


Figure 6: Using an autocoder

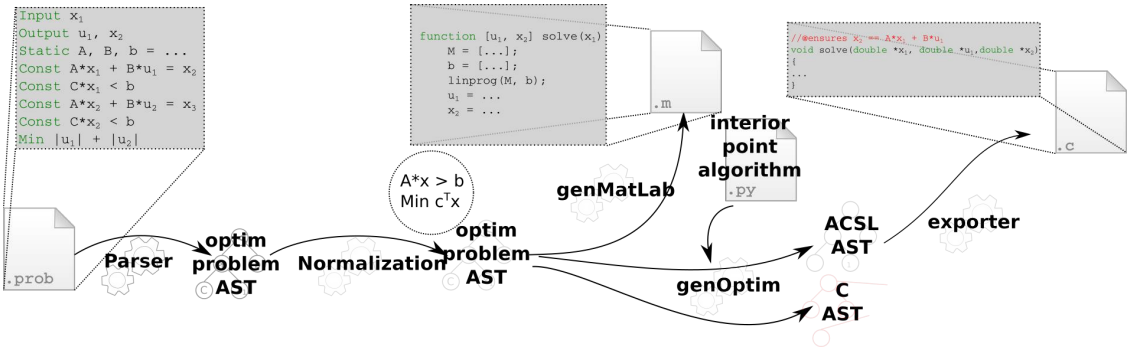


Figure 7: Internal aspect of the autocoder

IV. Interior-Point Method

We focus here on Interior Point methods, that, starting from a feasible solution, solve iteratively a sequence of linear problems, eg. using Newton methods, to reach the optimal solution. This method presented by[26] and improved by[27,28] is one the most efficient methods to solve linear and convex optimization problems. The presented work directly follows the theory presented in.[29] To keep the presentation simple and show the feasibility of our approach, we chose a primal method starting from the analytic center, whereas common methods are primal-dual and start from any point of the domain.
Let us introduce the major concepts before presenting the algorithm.

IV.A. Basic Concepts: Barrier Functions and Central Path

Barrier Function. Finding the optimum of a function is usually achieved by computing its derivative and searching for its zeros. In LP, we know that the optimum value is located on the border of the feasible set, which could be problematic to reach numerically. To solve this issue, we add a barrier function F to the cost function. The rationale of the barrier function $F(x)$ is to diverge when x get closer to the border of the feasible set.

In order to guarantee convergence bounds, the barrier function must be a Lipschitz function. In practice, a self-concordant barrier function[30] satisfies this requirement. The typical choice is a sum of logarithmic functions.

Definition 3 (Log barrier function) *Let F be the barrier function associated to problem P. We have:*

$$F(x) = \sum_{i=1}^{m} -\log(\langle a_i, x\rangle - b_i) \tag{3}$$

where a_i is the i-th row of A.

Definition 4 (Analytic center) *We define the analytic center as the point x_F^* verifying:*

$$x_F^* = \arg\min_{x \in E_f} F(x) \tag{4}$$

Central Path. Using both the barrier function and the initial cost function, one can characterize their linear combination.

Definition 5 *Let us introduce a new function, $\widetilde{f}$ as:*

$$\widetilde{f}(t, x) = tf(x) + F(x) \tag{5}$$

In this function, t is the parameter that weights the relative importance of the initial cost function with respect to the barrier function. When $t = 0$ the original cost function vanishes and minimizing $\widetilde{f}$ leads to the analytic center while when $t \to +\infty$ the cost function is predominant and minimizing $\widetilde{f}$ leads to the optimal point up to ϵ (Figure 8)

Definition 6 (Central path) *The central path is defined as the curve:*

$$x^*(t) = \arg\min_{x \in E_f} \widetilde{f}(t, x) \text{ for } t \in [0, +\infty) \tag{6}$$

Interior-point methods are part of the path-following optimization methods: starting from the analytic center and following the central path to reach the optimal point, as illustrated in Figure 8.

Approximate Centering Condition (ACC). Since the algorithm is doing numerical computation, the computed points cannot be located exactly on the central path. Thus, we need to introduce an *approximate centering condition*:

Definition 7 (ACC) *Let us define $\lambda(t, x)$ as:*

$$\lambda(t, x) = \widetilde{f}'(t, x)^T F''(x) \widetilde{f}'(t, x) \tag{7}$$

The approximate centering condition becomes:

$$\lambda(t, x) < \beta^2 \tag{8}$$

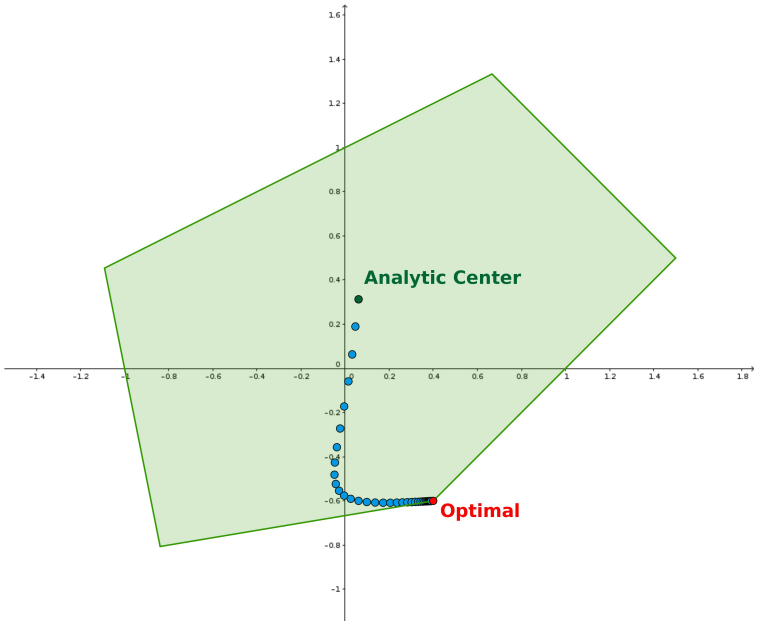


Figure 8: In blue: Points of the central path for different value of t

Notice that $F''(x)$ induces a norm[a] so λ measures how close to 0 the derivative is.

This definition aims to guarantee that the points are close to the central path so that the last point can be close enough to the optimal:

Theorem 1 ***(cf.[29])*** *If* $\lambda(t,x) < \beta^2$ *then* $\langle c,x\rangle - \langle c,x^*\rangle \leq \frac{\Delta}{t}$ *with* $\Delta = 1 + \frac{(\beta+1)\beta}{1-\beta}$

While t increases, computed points converge to the optimal. The approximate centering condition is the invariant that we maintain along the iterations of the algorithm.

IV.B. Algorithm

Since we are targeting embedded systems, we only use static variables, a common practice in embedded systems. We also define macros representing the different values of the problems, simplifying the definition of annotations.

```
/*@ requires acc(0, MatVar(X,2,1));
  @ ensures dot(c, MatVar(X,2,1))-dot(c,sol(A,b,c))<EPSILON;
  @ ensures A * MatVar(X,2,1) > b; */
void pathfollowing() {
    t = 0;
    /*@ loop invariant ensures acc(t, MatVar(X, 2, 1));
        loop invariant A * MatVar(X, 2, 1) > b;
        loop invariant ensures t > LOWER(l); */
    for (unsigned int l = 0;l<NBR;l++)
    {
        compute_pre();
        compute_dt();
        compute_dx();
        t = t + dt;
        for(unsigned int i = 0;i<N;i++)
            X[i] = X[i] + dx[i];
    }
}
```

Figure 9: Shape of the path following algorithm

[a]By construction, thanks to the use of a self-concordant barrier function.

Figure 9 presents the main function of the algorithm, computing the next iterate while following the central path. It relies on:

- `acc`, `dot` and `sol`, ACSL self-defined operators representing, the approximate centering condition, the dot product and the solution of $P(A, b, c)$ respectively.
- several constants representing:
  - `EPSILON`: the desired precision
  - `M` x `N`: the size of the matrix
  - `LOWER`, a function associating a lower bound for t at each iteration .
  - `NBR`: the number of iterations required so that $\text{LOWER}(NBR) > \frac{\Delta}{\epsilon}$
- subfunctions `compute_pre`, `compute_dt`, `compute_dx`

In this program, we need to prove that at the end of the loop we ensure :

$$\text{dot}(c, x) - \text{dot}(c, sol(A, b, c)) < EPSILON.$$

This is proved thanks to a lemma we call `acc_solution` and the main function loop invariants. From the first loop invariant and `acc_solution`, one deduces that $dot(c, x) - dot(c, sol(A, b, c)) < \frac{\Delta}{t}$ and from the second loop invariant, that $t > \frac{\Delta}{\epsilon}$, leading to the expected post-condition.

We now detail the content and contracts of the three functions used by the main one: `compute_dt`, `compute_dx` and `compute_pre`.

FUNCTION `COMPUTE_PRE` computes the gradient $F'(X)$ and the hessian $F''(X)$ of F at point X according to the Equs. 9 and 10.

$$(G)_j(X) = \sum_{i=0}^{m-1} -\frac{a_{i,j}}{\langle a_i, X\rangle - b_i} \tag{9}$$

$$(H)_{j,k}(X) = \sum_{i=0}^{m-1} \frac{a_{i,j}a_{i,k}}{\langle a_i, X\rangle - b_i} \tag{10}$$

FUNCTION `COMPUTE_DT` computes the update of dt according to Eq. 11. Following,[29] we have

$$dt = \frac{5}{36} \cdot \frac{1}{c^T \times H \times c} \tag{11}$$

We can deduce that $dt > 0.125t$. From this, we can lower bound t by a geometric progression: $\text{LOWER}(n) = t_0 \times (1.125)^n$. This allow us to determine the fixed number of iterations required to reach a given level of precision.

```
/*@ requires acc(t, MatVar(X,2,1));
  @ ensures MatVar(H,2,2) == hess(MatVar(X,2,1));
  @ ensures dt > t*0.125;
  @ ensures dt = (5/36)/norm(c,MatVar(X,2,1));
  @ */
void compute_dt();
```

Figure 10: Contract on `compute_dt`

FUNCTION `COMPUTE_DX` updates X using a Newtonian iteration on the derivative of F:

$$dx = H^{-1}(tc + G) \tag{12}$$

Notice that t in this formula is the one already updated $(t + dt)$. We expect from this update that the approximate centering condition remains valid.

V. The Ellipsoid Method

The other method that has attracted our attention is the Ellipsoid Method. Despite its relative efficiency with respect to Interior Point methods, the Ellipsoid method benefits from concrete proof elements and could be considered viable option for critical embedded systems where safety is more important than performance. Before recalling the main steps of the algorithm, the needed elements is presented.

V.A. Preliminaries

ELLIPSOIDS IN $\mathbb{R}^n$. An ellipsoid can be characterized by a positive definite matrix. A symmetric matrix P of $\mathbb{R}^{n\times n}$ is positive definite ($\mathcal{S}_n^+$) when

$$x^T P x > 0 \ , \ \forall x \neq 0 \in \mathbb{R}^n$$

Furthermore, this definition implies that all matrices P in $\mathcal{S}_n^+$ are invertible.

Definition 8 (Ellipsoid Sets) *Let $x \in \mathbb{R}^n$ and $P \in \mathbb{R}^{n\times n}$ positive-definite, we denote the Ellipsoid $E(P,x)$ the set :*

$$E(P,x) = \{z \in \mathbb{R}^n : (z-x)^T P^{-1}(z-x) \leq 1\}$$

Definition 9 (Euclidean ball) *Let $n \in \mathbb{N}$ we denote V_n the unit Euclidean ball in $\mathbb{R}^n$. $Vol(V_n)$ denotes its volume.*
In addition, we define $B(x,R)$ as the ball of radius R centered on x.

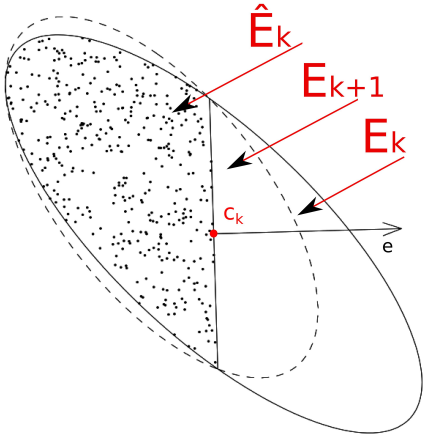
Definition 10 (Volume of ellipsoids) *Let $E(P,x)$ be an ellipsoid set in $\mathbb{R}^n$. We denote by $Vol(E)$ its volume defined as*

$$Vol(E(P,x)) = Vol(E) = \sqrt{det(P)} \cdot Vol(V_n)$$

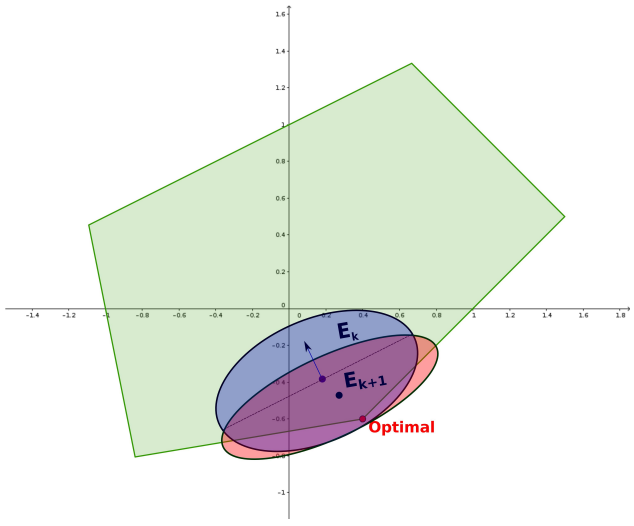
V.B. Algorithm

Let us now recall the main steps of the algorithm detailed in.[31–34]

Remark 1 *In the following, we denote E_k, x_k and P_k, the ellipsoid, the corresponding center and positive-definite matrix, respectively computed by the algorithm at the $k-th$ iteration.*



(a) e representing the vector normal to the chosen cutting hyperplane from oracle separation.



(b) Ellipsoid Cut for an iteration applied to LP

Figure 11: Ellipsoid Cut for an iteration

ALGORITHM. We start the algorithm with an ellipsoid containing the feasible set E_f, 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 x_k, we find a hyperplane containing x_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.*[35] In our LP setting, this step is not computationally expensive. 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^*. In addition to that, we can compute an upper bound γ of the ratio of $\text{Vol}(E_{k+1})$ to $\text{Vol}(E_k)$. The Figures 11 (a) and (b) illustrate such ellipsoids cuts.

ELLIPSOID TRANSFORMATION. From the oracle separation step, a separating hyperplane, e, that cuts E_k in half with the guarantee that x^* is localized in $\hat{E}_k$ has been computed. The following step is the *Ellipsoid transformation*. Using this hyperplane e, one can update the Ellipsoid E_k to its next iterate E_{k+1} according to equations 13 and 14.

$$x_{k+1} = x_k - \frac{1}{n+1} P_k \tilde{e} \tag{13}$$

and

$$P_{k+1} = \frac{n^2}{n^2-1}(P_k - \frac{2}{n+1} P_k \tilde{e}\tilde{e}^T P_k) \tag{14}$$

with: $\tilde{e} = \frac{e}{e^T P_k e}$.

One can now characterize the upper bound of the ratio of $\text{Vol}(E_{k+1})$ to $\text{Vol}(E_k)$.

Property 1 (Reduction ratio) *Let $k \geq 0$ be an iteration of the algorithm. We have*

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

HYPOTHESES. We recall that we assumed the matrix A and the vector b are such that the feasible set E_f is bounded and not empty. Additionally, further information about E_f is needed. Indeed, in order to know the number of steps required for the algorithm to return an ϵ-optimal solution, two scalars are needed:

- a radius R such that E_f is included in $B(\mathbf{0_n}, R)$
- another scalar r such that there exist a center c_1 such that $B(c_1, r)$ is included in E_f.

The main result can be stated as:

Theorem 2 *Let us assume that E_f is bounded, not empty and such that R, r are known. Then, for all $\epsilon > 0$, $c \in \mathbb{R}^n$ there exists a N such that the algorithm, using N iterations, will return $\hat{x}$, such that*

$$f(\hat{x}) \leq f(x^*) + \epsilon \text{ and } \hat{x} \in E_f \text{ } (\epsilon \text{ optimal})$$

V.C. Structure of the Algorithm and Annotations

Figure 12 presents an annotated version of the implementation. Annotations combine ellipsoidal constraints `inEllipsoid` and volume related properties to express the progress of the algorithm along iterates.

VI. Proving the annotation

FramaC supports the analysis of C code with ACSL annotations through multiple analyzers. The tool WP projects both the ACSL specification – the Hoare triples – and the source code in a predicate encoding on which external tools can be applied.

Each function contract, or statement assertion, becomes a goal, a proof objective. This goal can be solved either in an automatic fashion by SMT solvers such as Alt-Ergo, *cf.*,[36] Z3, *cf.*,[37] or thanks to manual proofs supported by proof assistants such as Coq, *cf.*[38]

```
/*@ requires inEllipsoid(Ellipsoid(P_minus,x_minus),sol(A,b,c));
  @ ensures dot(c,MatVar(X,2,1))-dot(c,sol(A,b,c)) <= EPSILON;
  @ ensures A * MatVar(X,2,1) <=  b; */
void ellipsoidMethod() {
  long double P_minus[N*N],x_minus[N];
  long double x_best[N],grad[N];
  int i,Nit;
  initializationEllipsoid(P_minus,x_minus,N,R);
  affectationVector(x_best, x_minus, N);
    /*@ loop invariant Vol(Ellipsoid(P_minus,x_minus)) <= gamma^i * R^N ;
      @ loop invariant inEllipsoid(Ellipsoid(P_minus,x_minus),sol(A,b,c)) ;
      @ loop invariant isMoreOptimal(x_best,x_minus) ; */
  for (i = 0; i < NBR; ++i) {
    updateBest(x_best, x_minus);
    getGrad(x_minus, grad);
    updateEllipsoid(P_minus, x_minus, grad);
  }
  affectationVector(x, x_best, N);
}
```

Figure 12: Shape of the Main Ellipsoid Method Algorithm

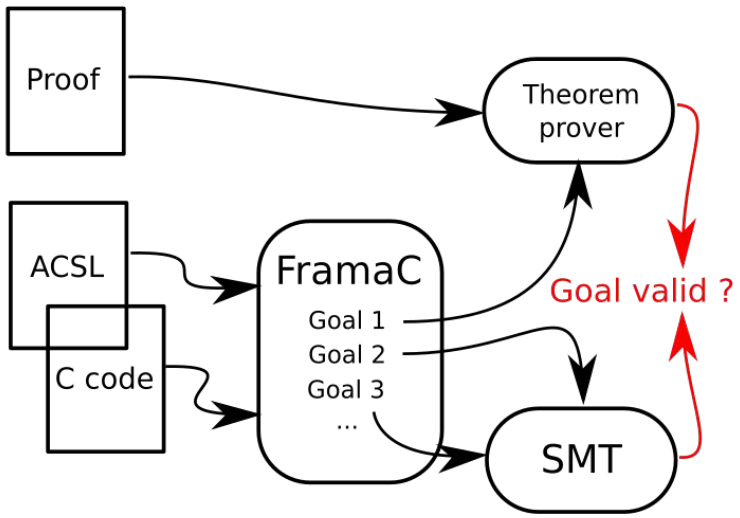


Figure 13: Automatic Verification Framework

VI.A. Low level matrix operation

We used this toolchain to prove that the annotations are valid. The goal is to have as many annotations as possible proven by SMT solvers since a Coq proof requires additional work. One of the difficulties in Coq is to address low level C manipulations such as matrix operations. The memory model axiomatization makes the manipulation of these encodings difficult while the properties are not specifically hard to prove. To avoid this, all matrix operations in C are associated to an ACSL contract, stating the computation performed in linear algebra instead of arrays and pointers. Fig. 14 presents an example of these annotations while Fig. 15, describes the (simplified) generated goal. SMT solvers easily discard this kind of proof objectives.

VI.B. Linear algebra proof

Once these matrix operations have been abstracted by their ACSL axiomatization, all that remains is to prove the textbook proofs at the matrix level.

Most of the theorems can be written as ACSL lemma, enabling their generic proof, independently of a specific instantiation.

In the Interior-point method experiment, proofs performed with the proof assistant Coq are mainly

```
/*@ ensures MatVar(X, 2, 1) == \old(
  @ mat_add(MatVar(X, 2, 1),
  @ MatVar(dx, 2, 1))); */
{
    X[0] = X[0] + dx[0];
    X[1] = X[1] + dx[1];
}
```

Figure 14: Simple example of matrix operation specification

```
goal set_X_post:
  let a = shift_A2_float64(global(G_X_1018), 0) : addr in
  let a_1 = shift_float64(a, 0) : addr in
  let a_2 = shift_float64(a, 1) : addr in
  let a_3 = shift_A2_float64(global(G_dx_1019), 0) : addr in
  let a_4 = shift_float64(a_3, 0) : addr in
  let a_5 = shift_float64(a_3, 1) : addr in
  forall t : (addr,real) farray.
  let r = t[a_1] : real in
  let r_1 = t[a_2] : real in
  let r_2 = t[a_4] : real in
  let r_3 = t[a_5] : real in
  let a_6 = L_mat_add(L_MatVar(t, a, 2, 1), L_MatVar(t, a_3, 2, 1)) : A_LMat in
  let a_7 = L_MatVar(t[a_1 <- r + r_2][a_2 <- r_1 + r_3], a, 2, 1) : A_LMat in
  is_float64(r) → is_float64(r_1) →
  is_float64(r_2) → is_float64(r_3) →
  (1 = L_getN(a_6)) → (2 = L_getM(a_6)) →
  (1 = L_getN(a_7)) → (2 = L_getM(a_7)) →
  (L_mat_get(a_6, 0, 0) = L_mat_get(a_7, 0, 0)) →
  (L_mat_get(a_6, 1, 0) = L_mat_get(a_7, 1, 0)) →
  (a_6 = a_7)
```

Figure 15: Generated goal for basic matrix operation

restricted to the core results available in Nesterov Book, or direct implications from them. All remaining proofs can be automatically performed by SMT solvers.

Fig. 16, illustrates a Coq subgoal that appears in these proofs. This specific one expresses the following, stating that the computed solution is ϵ-optimal:

Theorem 3 (optimality) $\forall\ x \in \mathbb{R}^n,\ t \in \mathbb{R}$ *if*

$$acc(t, x)\ \textit{and}\ t > LOWER(NBR) \tag{15}$$

then, we have:

$$dot(c, x) - dot(c, sol(A, b, c)) < EPSILON$$

VII. Conclusion

In this article, annotations for numerical algorithms solving linear programming problems were proposed. We focused on a primal Interior Point algorithm and on the Ellipsoid method. The Interior Point algorithm is fully automated annotated while annotations for the Ellipsoid method are still manual. Annotations can be partitioned in two levels:

- Low-level annotation specifying matrix operations. This separates the handling of memory array accesses from math reasoning, easing the automatic proof of simpler annotations by SMT solvers.

```
Goal
  let a := (shift_A2_float64 ((global (G_pathfollowing_X_1018)%Z)) 0%Z) in
  let a_1 := (shift_float64 a 0%Z) in
  let a_2 := (shift_float64 a 1%Z) in
  let a_3 := L_MatCst_5_2
    (real_dec -3 0) (real_dec -1 0)
    (real_dec  5 0) (real_dec 1 0)
    (real_dec -1 0) (real_dec 3 0)
    (real_dec -1 0) (real_dec 1 0)
    (real_dec -2 0) (real_dec 6 0) in
  let a_4 := L_MatCst_5_1
    (real_dec -3 0) (real_dec -2 0)
    (real_dec -5 0) (real_dec -2 0)
    (real_dec -4 0) in
  ...
  L_acc 0 (L_MatVar t_13 a 2 1) →
  L_acc r a_5 →
  L_mat_gt (L_mat_mul a_3 a_5) a_4) →
  (real_dec -1 0) * r_1 + (real_dec 4 0) * r_2
    < (real_dec 1 -3) + (L_sol a_3 a_4 (
     L_MatCst_2_1 (real_dec -1 0) (real_dec 4 0))).
```

Figure 16: Optimality of the result in Coq (most precondition have been removed to fit in the paper).

- Math reasoning annotations specify the high level properties of the algorithms: convergence, preservation of feasibility, ϵ-optimality of the solution, etc. These annotations are proven either by SMT solvers or with Coq. The finalization of these proofs is still a work in progress.

Our perspectives include the application of this approach to more general settings, eg. generic linear programming problems. We plan to couple code generation with annotations and proof synthesis, providing embedded code together with its functional soundness proof, at the code level. This would include the generation of the corresponding Coq proofs.

Our approach is also compatible with more general convex optimization problems. Finally, we intend to address floating point semantics issues while performing the proofs. This would guarantee a completely sound and bug-free implementation.

Acknowledgments

This work was partially supported by projects ANR ASTRID VORACE and CPS SORTIES National Science Foundation (NSF) under grant 1446758. The authors would also like to deeply thank Didier Henrion for his participation to this work.

References

[1] Boyd, S. and Vandenberghe, L., *Convex optimization*, Cambridge University Press, New York, NY, USA, 2004.

[2] Nocedal, J. and Wright, S. J., *Numerical Optimization*, Springer, New York, 2nd ed., 2006.

[3] Jerez, J. L., Goulart, P. J., Richter, S., Constantinides, G. A., Kerrigan, E. C., and Morari, M., "Embedded Online Optimization for Model Predictive Control at Megahertz Rates," *IEEE Trans. Automat. Contr.*, Vol. 59, No. 12, 2014, pp. 3238–3251.

[4] Açikmese, B., Carson, J., and Blackmore, L., "Lossless Convexification of Nonconvex Control Bound and Pointing Constraints of the Soft Landing Optimal Control Problem," *IEEE Trans. Contr. Sys. Techn.*, Vol. 21, No. 6, 2013, pp. 2104–2113.

[5] Blackmore, L., Açikmese, B., and Carson, J., "Lossless convexification of control constraints for a class of nonlinear optimal control problems," *Systems & Control Letters*, Vol. 61, No. 8, 2012, pp. 863–870.

[6] Blackmore, L., *IEEE Control Systems Magazine*, Vol. 36, No. 6, 2016, pp. 24–26.

[7] Roux, P., "Formal Proofs of Rounding Error Bounds," *Journal of Automated Reasoning*, 2015, pp. 1–22.

[8]Roux, P., Voronin, Y.-L., and Sankaranarayanan, S., "Validating Numerical Semidefinite Programming Solvers for Polynomial Invariants," *Static Analysis Symposium, 23rd International Symposium, SAS 2016, Edinburgh, UK, September 8-10, 2016. Proceedings*, edited by Springer, LNCS, 2016, to appear.

[9]Richter, S., Jones, C. N., and Morari, M., "Certification aspects of the fast gradient method for solving the dual of parametric convex programs," *Mathematical Methods of Operations Research*, Vol. 77, No. 3, 2013, pp. 305–321.

[10]McGovern, L. K., *Computational Analysis of Real-Time Convex Optimization for Control Systems*, Ph.D. thesis, Massachussetts Institute of Technology, Boston, USA, May 2000.

[11]McGovern, L. K. and Feron, E., "Requirements and hard computational bounds for real-time optimization in safety-critical control systems," *Decision and Control, 1998. Proceedings of the 37th IEEE Conference on*, Vol. 3, 1998, pp. 3366–3371 vol.3.

[12]Feron, E., "From Control Systems to Control Software," *Control Systems, IEEE*, Vol. 30, No. 6, December 2010, pp. 50 –71.

[13]Herencia-Zapana, H., Jobredeaux, R., Owre, S., Garoche, P.-L., Feron, E., Perez, G., and Ascariz, P., "PVS Linear Algebra Libraries for Verification of Control Software Algorithms in C/ACSL," *NASA Formal Methods - Forth International Symposium, NFM 2012, Norfolk, VA USA, April 3-5, 2012. Proceedings*, edited by A. Goodloe and S. Person, Vol. 7226 of *Lecture Notes in Computer Science*, Springer, 2012, pp. 147–161.

[14]Wang, T., Jobredeaux, R., Herencia-Zapana, H., Garoche, P.-L., Dieumegard, A., Feron, E., and Pantel, M., *From Design to Implementation: An Automated, Credible Autocoding Chain for Control Systems*, Vol. 460 of *Lecture Notes in Control and Information Sciences*, Springer Berlin Heidelberg, 2016, pp. 137–180.

[15]Araiza-Illan, D., Eder, K., and Richards, A., "Verification of control systems implemented in simulink with assertion checks and theorem proving: A case study," 2015, pp. 2670–2675.

[16]Pajic, M., Park, J., Lee, I., Pappas, G., and Sokolsky, O., "Automatic verification of linear controller software," 2015, pp. 217–226.

[17]Wang, T., Jobredeaux, R., Pantel, M., Garoche, P.-L., Feron, E., and Henrion, D., "Credible Autocoding of Convex Optimization Algorithms," *Optimization and Engineering*, 2016, to appear.

[18]Bemporad, A., Borrelli, F., Morari, M., et al., "Model predictive control based on linear programming˜ the explicit solution," *IEEE Transactions on Automatic Control*, Vol. 47, No. 12, 2002, pp. 1974–1985.

[19]Floyd, R. W., "Assigning Meanings to Programs," *Proceedings of Symposium on Applied Mathematics*, Vol. 19, 1967, pp. 19–32.

[20]Hoare, C. A. R., "An axiomatic basis for computer programming," *Commun. ACM*, Vol. 12, October 1969, pp. 576–580.

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

[22]Baudin, P., Filliâtre, J.-C., Marché, C., Monate, B., Moy, Y., and Prevosto, V., "ACSL: ANSI/ISO C Specification Language. Version 1.11." `http://frama-c.com/download/acsl.pdf`.

[23]Mayne, D. Q., Rawlings, J. B., Rao, C. V., and Scokaert, P. O., "Constrained model predictive control: Stability and optimality," *Automatica*, Vol. 36, No. 6, 2000, pp. 789–814.

[24]Rawlings, J. B. and Mayne, D. Q., *Model predictive control: Theory and design*, Nob Hill Pub., 2009.

[25]Grant, M. and Boyd, S., "CVX: Matlab Software for Disciplined Convex Programming, version 2.1," `http://cvxr.com/cvx`, March 2014.

[26]Karmarkar, N., "A new polynomial-time algorithm for linear programming," *Combinatorica*, Vol. 4, No. 4, December 1984, pp. 373–395.

[27]Nesterov, Y. and Nemirovski, A., "A general approach to the design of optimal methods for smooth convex functions minimization," *Ekonomika i Matem. Metody*, Vol. 24, 1988, pp. 509–517.

[28]Nesterov, Y. and Nemirovski, A., *Interior-point Polynomial Algorithms in Convex Programming*, Vol. 13 of *Studies in Applied Mathematics*, Society for Industrial and Applied Mathematics, 1994.

[29]Nesterov, Y., *Introductory lectures on convex optimization : a basic course*, Applied optimization, Kluwer Academic Publ., Boston, Dordrecht, London, 2004.

[30]Nesterov, Y. and Nemirovski, A., *Self-Concordant functions and polynomial time methods in convex programming*, Materialy po matematicheskomu obespecheniiu EVM, USSR Academy of Sciences, Central Economic & Mathematic Institute, 1989.

[31]Bland, R. G., Goldfarb, D., and Todd, M. J., "The ellipsoid method: A survey," *Operations research*, Vol. 29, No. 6, 1981, pp. 1039–1091.

[32]Grötschel, M., Lovász, L., and Schrijver, A., "The ellipsoid method and its consequences in combinatorial optimization," *Combinatorica*, Vol. 1, No. 2, 1981, pp. 169–197.

[33]Khachiyan, L. G., "Polynomial algorithms in linear programming," *USSR Computational Mathematics and Mathematical Physics*, Vol. 20, No. 1, 1980, pp. 53–72.

[34]Boyd, S. P. and Barratt, C. H., *Linear controller design: limits of performance*, Prentice Hall Englewood Cliffs, NJ, 1991.

[35]Ben-Tal, A. and Nemirovski, A., "Lecture notes, optimizattion i-ii, convex analysis, non-linear programming theory, non-linear programming algorithms.," 2004.

[36]Conchon, S., Contejean, E., and Iguernelala, M., "Canonized Rewriting and Ground AC Completion Modulo Shostak Theories : Design and Implementation," *Logical Methods in Computer Science*, Vol. 8, No. 3, 2012.

[37]de Moura, L. and Bjørner, N., "Z3: An Efficient SMT Solver," *TACAS*, 2008, pp. 337–340.

[38]The Coq development team, *The Coq proof assistant reference manual*, LogiCal Project, 2012, Version 8.4.