



**HAL**  
open science

# Preservation of Lyapunov-Theoretic Proofs: From Real to Floating-Point Numbers

Vivien Maisonneuve

► **To cite this version:**

Vivien Maisonneuve. Preservation of Lyapunov-Theoretic Proofs: From Real to Floating-Point Numbers. [Research Report] Mines ParisTech. 2013. hal-01086727

**HAL Id: hal-01086727**

**<https://minesparis-psl.hal.science/hal-01086727>**

Submitted on 24 Nov 2014

**HAL** is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

# Preservation of Lyapunov-Theoretic Proofs: From Real to Floating-Point Numbers

Vivien Maisonneuve<sup>1</sup>

*CRI, Mathématiques et Systèmes  
MINES ParisTech  
Fontainebleau, France*

---

## Abstract

In [3] Feron presents how Lyapunov-theoretic proofs of stability can be migrated toward computer-readable and verifiable certificates of control software behavior by relying on Floyd's and Hoare's proof system.

However, Feron's proofs are addressed towards exact, real arithmetic and do not accurately represent the behavior of realistic programs run with machine arithmetic. We address the issue of errors resulting from the use of floating-point arithmetic: we present an approach to translate Feron's proof invariants on real arithmetic to similar invariants on floating-point numbers and show how our methodology applies to prove stability, thus allowing to verify whether the stability invariant still holds when the controller is implemented.

We study in details the open-loop system of Feron's paper. We also use the same approach for Feron's closed-loop system, but the constraints are too tight to show stability in this second case: more leeway should be introduced in the proof on real numbers, otherwise the resulting system might be unstable.

---

## 1 Introduction

Provable stability constitutes an essential attribute of control systems, especially when human safety is involved, as in medical or aeronautical domains. Motivated by such applications, there exist many theorems to support system stability and performance under various assumptions and in various settings.

Stability criteria apply to a class of dynamical systems for which a stability proof is needed. Modern systems developments, such as adaptive control technologies, rely on robust stability and performance criteria as the primary

---

<sup>1</sup> Email: [vivien.maisonneuve@cri.mines-paristech.fr](mailto:vivien.maisonneuve@cri.mines-paristech.fr).

justification for their relevance to safety-critical control applications. Lyapunov’s stability theory plays a critical role in that regard.

The low-level software implementation of a control law can be inspected by analysis tools available to support the development of safety-critical computer programs. The simplest program analysis techniques consist of performing several simulations, sometimes including a software or hardware representation of the controlled system in the loop. However, simulations provide information about a large but only finite number of system behaviors. More advanced methods include model checking and abstract interpretation, e.g. using *As-trée* [2]. In these methods, inputs are computer programs and outputs are certificates of proper program behavior along the chosen criterion. Another possibility is to use theorem-proving techniques, supported by computer tools such as *Coq*, *Isabelle* or *PVS* [7,10,9]. These proof assistants can be used to establish properties of programs and more general mathematical constructs. Model checking, abstract interpretation, and theorem-proving tools are all used to verify safety-critical applications.

In [3], Feron investigates how control-system domain knowledge and, in particular, Lyapunov-theoretic proofs of stability and performance, can be migrated toward computer-readable and verifiable certificates of control software behavior by relying on Floyd’s and Hoare’s proof system [11], applied to MATLAB pseudocode (Sections 2 and 3). His article focuses on using this framework to support such properties, namely, bounded-input, bounded-state stability, as they apply to control-system code implementations. But errors resulting from the use of floating-point arithmetic are not addressed. Feron presents both open-loop and closed-loop implementations of his controller, i.e. depending whether the system action on the environment and its feedback are modeled or not.

In this paper, we present an approach to translate Lyapunov-theoretic stability proof invariants on pseudocode with real arithmetic, as provided in Feron’s article, to similar invariants on machine code that take into account rounding errors introduced by floating-point arithmetic. We use them to verify whether stability conditions still hold, in which case system stability with floating-point numbers can be established.

This document is organized as follows. The next section describes the second-order dynamical system example used by Feron in [3], with the corresponding controller. Next, Feron’s analysis of the open-loop controller with real numbers is presented, followed by its rewriting with floating-point arithmetic. Then we discuss the case of the closed-loop system. The document concludes with a discussion of the generality of this successful approach.

## 2 Motivating Example by Feron

We consider the first system described in the article of Feron [3]. It is a dynamical system composed of a single mass and a single spring shown in Figure 1.

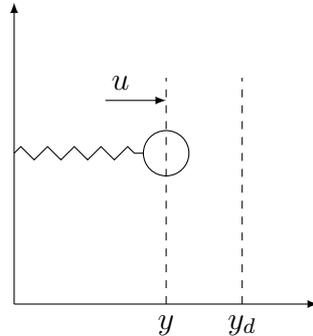


Fig. 1. Mass-spring system

The position input  $y$  of the mass is available for feedback control. The signal  $y_d$  is the reference signal, that is, the desired position to be followed by the mass.

A discrete-time MATLAB implementation of the controller, using real numbers, is provided in [3]. The source code is shown below:

```

1 Ac = [0.4990, -0.0500; 0.0100, 1.0000];
2 Bc = [1; 0];
3 Cc = [564.48, 0];
4 Dc = -1280;
5 xc = zeros(2, 1);
6 receive(y, 2); receive(yd, 3);
7 while (1)
8     yc = max(min(y - yd, 1), -1);
9     skip;
10    u = Cc*xc + Dc*yc;
11    xc = Ac*xc + Bc*yc;
12    send(u, 1);
13    receive(y, 2);
14    receive(yd, 3);
15    skip;
16 end

```

Apart from the mechanical system state observation  $y$  and the desired system output  $y_d$ , variables in this code are:

- $x_c = \begin{pmatrix} x_{c1} \\ x_{c2} \end{pmatrix} \in \mathbb{R}^2$  is the discrete-time controller state;

- $y_c \in [-1, 1]$  is the bounded output tracking error, i.e. the input  $y - y_d$  passed through a saturation function to avoid variable overflow in the controller;
- $u \in \mathbb{R}$  is the mechanical system input, i.e. the action to be performed according to the controller.

Constants  $A_c$ ,  $B_c$ ,  $C_c$  and  $D_c$  are the discrete-time controller state, input, output and feedthrough matrices. The commands `send` and `receive` are basically I/O: they respectively send and receive data given in the commands first argument through a specific channel given by the commands second argument.

### 3 Open-Loop Stability Proof in Real by Feron

The stability proof of this system relies on Lyapunov theory. In simple terms, a system is *Lyapunov stable* if all states  $x_c$  reachable from an initial starting state belonging to a bounded neighborhood  $V$  of an equilibrium point  $x_e$  remain in  $V$ .

Lyapunov theory provides constraints that must be satisfied by such a  $V$ . On linear systems, they are equations that can be solved using linear matrix inequalities [1]. Commonly,  $V$  is an ellipsoid.

In this case, to prove Lyapunov stability, we need to show that at any time,  $x_c$  belongs to the set  $\mathcal{E}_P$  chosen by Feron according to Lyapunov's theory:

$$\mathcal{E}_P = \{x \in \mathbb{R}^2 \mid x^T \cdot P \cdot x \leq 1\}, \quad P = 10^{-3} \begin{pmatrix} 0.6742 & 0.0428 \\ 0.0428 & 2.4651 \end{pmatrix}.$$

using  $\mathcal{E}_P$  as the stability neighborhood  $V$ .

This set is a full ellipse, centered around 0 and slightly slanted:

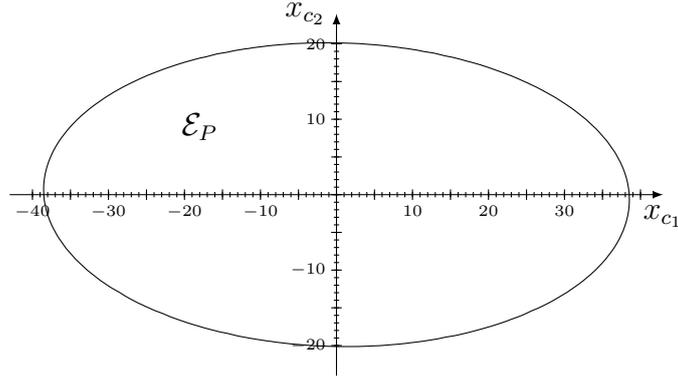
$$x_c \in \mathcal{E}_P \iff 0.6742x_{c1}^2 + 0.0856x_{c1}x_{c2} + 2.4651x_{c2}^2 \leq 1000.$$

$\mathcal{E}_P$  is drawn in Figure 2.

A stability proof of the controller is provided in [3], using Floyd-Hoare program annotation technique [11]: each program instruction comes with an invariant. The program annotated by Feron is reproduced below.

```

5 xc = zeros(2,1);
  %  $x_c \in \mathcal{E}_P$ 
6 receive(y, 2); receive(yd, 3);
  %  $x_c \in \mathcal{E}_P$ 
7 while (1)
  %  $x_c \in \mathcal{E}_P$ 
8   yc = max(min(y - yd, 1), -1);
  %  $x_c \in \mathcal{E}_P, \quad y_c^2 \leq 1$ 
    
```


 Fig. 2. The stability domain  $\mathcal{E}_P$ 

```

9      skip;
      %  $\begin{pmatrix} x_c \\ y_c \end{pmatrix} \in \mathcal{E}_{Q_\mu}$ ,  $Q_\mu = \begin{pmatrix} \mu^P & 0_{2 \times 1} \\ 0_{1 \times 2} & 1-\mu \end{pmatrix}$ ,  $\mu = 0.9991$ 
10     u = Cc*xc + Dc*yc;
      %  $\begin{pmatrix} x_c \\ y_c \end{pmatrix} \in \mathcal{E}_{Q_\mu}$ ,  $u^2 \leq (C_c \ D_c) \cdot Q_\mu^{-1} \cdot (C_c \ D_c)^{-1}$ 
11     xc = Ac*xc + Bc*yc;
      %  $x_c \in \mathcal{E}_{\tilde{P}}$ ,  $\tilde{P} = [(A_c \ B_c) \cdot Q_\mu^{-1} \cdot (A_c \ B_c)^T]^{-1}$ ,  $u^2 \leq (C_c \ D_c) \cdot Q_\mu^{-1} \cdot (C_c \ D_c)^{-1}$ 
12     send(u, 1);
      %  $x_c \in \mathcal{E}_{\tilde{P}}$ 
13     receive(y, 2);
      %  $x_c \in \mathcal{E}_{\tilde{P}}$ 
14     receive(yd, 3);
      %  $x_c \in \mathcal{E}_{\tilde{P}}$ 
15     skip;
      %  $x_c \in \mathcal{E}_P$ 
16 end
    
```

Most of the proof relies on algebraic arguments. For example, the invariant loosening on Line 9:

```

      %  $x_c \in \mathcal{E}_P$ ,  $y_c^2 \leq 1$ 
9     skip;
      %  $\begin{pmatrix} x_c \\ y_c \end{pmatrix} \in \mathcal{E}_{Q_\mu}$ ,  $Q_\mu = \begin{pmatrix} \mu^P & 0_{2 \times 1} \\ 0_{1 \times 2} & 1-\mu \end{pmatrix}$ ,  $\mu = 0.9991$ 
    
```

means

$$x_c \in \mathcal{E}_P \wedge y_c^2 \leq 1 \implies \begin{pmatrix} x_c \\ y_c \end{pmatrix} \in \mathcal{E}_{Q_\mu} \text{ with } Q_\mu = \begin{pmatrix} \mu^P & 0_{2 \times 1} \\ 0_{1 \times 2} & 1-\mu \end{pmatrix} \text{ and } \mu = 0.9991.$$

The correctness of this assertion stems from the fact that, given any value of  $\mu \in [0, 1]$ , the domain  $\mathcal{E}_{Q_\mu}$  is a solid ellipsoid, centered around 0, and whose intersection with the plane  $y_c = 1$  is equal to  $\mathcal{E}_P$ . Consequently, the solid bounded cylinder  $\mathcal{C} = \{ \begin{pmatrix} x_c \\ y_c \end{pmatrix} \mid x_c \in \mathcal{E}_P \wedge y_c^2 \leq 1 \}$  is included within  $\mathcal{E}_{Q_\mu}$

(Figure 3).

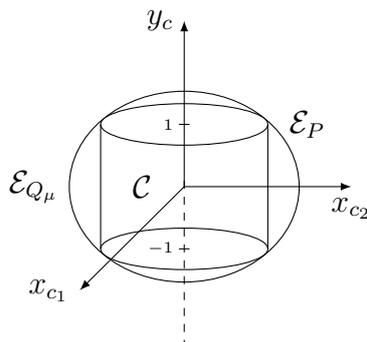


Fig. 3. Inclusion of  $\mathcal{E}_P$  within  $\mathcal{E}_{Q_\mu}$

The following invariants

```

%  $\begin{pmatrix} x_c \\ y_c \end{pmatrix} \in \mathcal{E}_{Q_\mu}$ ,  $Q_\mu = \begin{pmatrix} \mu P & 0_{2 \times 1} \\ 0_{1 \times 2} & 1 - \mu \end{pmatrix}$ ,  $\mu = 0.9991$ 
10 u = Cc*xc + Dc*yc;
%  $\begin{pmatrix} x_c \\ y_c \end{pmatrix} \in \mathcal{E}_{Q_\mu}$ ,  $u^2 \leq (C_c \ D_c) \cdot Q_\mu^{-1} \cdot (C_c \ D_c)^{-1}$ 
11 xc = Ac*xc + Bc*yc;
%  $x_c \in \mathcal{E}_{\tilde{P}}$ ,  $\tilde{P} = [(A_c \ B_c) \cdot Q_\mu^{-1} \cdot (A_c \ B_c)^T]^{-1}$ ,  $u^2 \leq (C_c \ D_c) \cdot Q_\mu^{-1} \cdot (C_c \ D_c)^{-1}$ 

```

also rely on algebraic arguments and theorems.

Other invariants are trivial. Finally, only the very last loosening

```

%  $x_c \in \mathcal{E}_{\tilde{P}}$ 
15 skip;
%  $x_c \in \mathcal{E}_P$ 

```

i.e.  $\mathcal{E}_{\tilde{P}} \subset \mathcal{E}_P$ , that “closes” the loop, is not purely algebraic since its validity relies on the numerical parameters  $A_c$ ,  $B_c$ ,  $C_c$ ,  $D_c$ . This assertion needs to be checked to ensure the correctness of the proof statements. This can be done either numerically, or algebraically for at most two-dimensional systems like this one.

This stability proof has been checked with **Mathematica** [12] and is available online<sup>2</sup>.

<sup>2</sup> Mathematica source file is available at: [http://www.cri.ensmp.fr/people/maisonneuve/lyafloat/resources/lyafloat\\_stability.nb](http://www.cri.ensmp.fr/people/maisonneuve/lyafloat/resources/lyafloat_stability.nb), and the corresponding PDF file at: [http://www.cri.ensmp.fr/people/maisonneuve/lyafloat/resources/lyafloat\\_stability.pdf](http://www.cri.ensmp.fr/people/maisonneuve/lyafloat/resources/lyafloat_stability.pdf).

## 4 Open-Loop Stability Proof Scheme with Floating-Point Numbers

We would like to check that the stability proof still holds on a controller implemented with floating-point numbers. When using floating-point arithmetic, the values of constants are slightly altered and calculations are likely to produce rounding errors.

In absolute terms, it is impossible to switch from real to floating-point numbers without affecting the behavior of the controller. Thus, the stability proof cannot be preserved in the general case. On the other hand, the program still can be stable if rounding errors are small enough and the final inclusion  $\mathcal{E}_{\tilde{P}} \subset \mathcal{E}_P$  leaves them enough room. We study how proof invariants can be tweaked so that they correspond to a floating-point semantic. Our goal is to derive from the proof scheme for real numbers a proof scheme suited for floating-point arithmetic, whose correctness, although not guaranteed, can be checked as easily.

In this section, we use the IEEE 754 floating-point standard [6] encoded on 64 bits<sup>3</sup> as most of today’s floating-point units do. In this standard, both addition and multiplication are correctly rounded depending on the active rounding mode, which allows to bound the rounding error depending on the values of operands. The case of other numeric representations is discussed in Section 6.

### 4.1 Translation Scheme

The general translation scheme is the following (Figure 4). Each instruction  $\mathbf{instr}_{\mathbb{R}}$  in the controller code with real numbers is turned into an equivalent instruction  $\mathbf{instr}_{\mathbb{F}}$  on floating points, where real constants and operators are replaced by their floating-point counterparts. The post-condition  $\mathbf{post}_{\mathbb{F}}$  on the floating-point side is computed using the same proof argument  $T$  as in the original code, enlarged to include the error term bound  $E$  that may occur in floating-point arithmetics in  $\mathbf{instr}_{\mathbb{F}}$ . The precondition  $\mathbf{pre}_{\mathbb{R}}$  might have been previously replaced by  $\mathbf{pre}_{\mathbb{F}}$  as the postcondition of a preceding instruction.

The error term bound  $E$  depends on the instruction  $\mathbf{instr}_{\mathbb{F}}$  and on the information on program variables given by  $\mathbf{pre}_{\mathbb{F}}$ , hence the notation  $E(\mathbf{pre}_{\mathbb{F}}, \mathbf{instr}_{\mathbb{F}})$ . The floating-point arithmetic error is not necessary bounded even when the variables are, e.g. in a division. In this case, we are not able to compute a suitable  $\mathbf{post}_{\mathbb{F}}$  and the proof translation attempt fails.

Also,  $\mathbf{post}_{\mathbb{F}}$  will be used as a precondition in the next instruction. An effort has to be made so that  $\mathbf{post}_{\mathbb{F}}$  has a similar shape to  $\mathbf{post}_{\mathbb{R}}$ , in order

<sup>3</sup> The procedure that follows would be exactly the same with 32-bit floating-point numbers, only with different numerical results.



```

10 % (x_c) ∈ E_{Q_μ}, Q_μ = ( μ^P  0_{2×1} )
    u = Cc*xc + Dc*yc;

```

### 4.3 Invariant on $u$

Then, we consider the next instruction in the original proof scheme:

```

10 % (x_c) ∈ E_{Q_μ}, Q_μ = ( μ^P  0_{2×1} )
    u = Cc*xc + Dc*yc;
    % (x_c) ∈ E_{Q_μ}, u^2 ≤ (C_c D_c) · Q_μ^{-1} · (C_c D_c)^{-1}

```

First of all, matrices  $C_c$  and  $D_c$  must be replaced by their floating-point counterparts  $C_c^f$  and  $D_c^f$  both in the program instruction Line 10 and the ensuing invariant. This invariant relies only on algebraic arguments and does not depend on the values in the matrices, it still holds considering exact arithmetic operations. But this is not sufficient: indeed, this instruction is a sum of matrix multiplications, i.e. a set of additions and multiplications on floating-point numbers that yield rounding errors.

We can notice that entering this instruction, the values of matrices  $C_c^f$ ,  $D_c^f$  and  $\mathcal{E}_{Q_\mu}$  are known, and the values of  $x_c$  and  $y_c$  are bounded by the precondition

$$\begin{pmatrix} x_c \\ y_c \end{pmatrix} = \begin{pmatrix} x_c \\ y_c \end{pmatrix} \in \mathcal{E}_{Q_\mu},$$

that is

$$0.000673593x_{c_1}^2 + 0.000085523x_{c_1}x_{c_2} + 0.00246288x_{c_2}^2 + 0.9991y_c^2 \leq 1. \quad (1)$$

From (1), we deduce:

$$\begin{cases} |x_{c_1}| \leq 3 \cdot 10^5 \sqrt{\frac{13\,695}{829\,322\,227\,639}} < 38.5515 \\ |x_{c_2}| \leq 10^5 \sqrt{\frac{33\,710}{829\,322\,227\,639}} < 20.1614 \\ |y_c| \leq \frac{100}{\sqrt{9991}} < 1.00046 \end{cases} \quad (2)$$

Here we are able to find algebraic solutions, but this may be impossible with ellipsoids of higher dimension. Still, we would be able to find bounds using numerical methods.

In floating-point arithmetic, rounding errors created by addition and multiplication operators can be bounded when the operands are known or bounded by (2), provided that overflow, underflow, and denormalized numbers do not occur [4,5].

Here, we need to compute

$$C_c^f x_c + D_c^f y_c = C_{c(0,0)}^f x_{c_1} + D_c^f y_c$$

where all values in the right-hand term are known or bounded. Thus, a constant can be computed that bounds the absolute rounding error created when computing  $u$ . We used Rangelab [8], a static analysis tool to automatically validate the accuracy of floating-point or fixed-point computations, to compute an upper bound  $e$  for the error term<sup>4</sup>:

$$e = 5.90 \cdot 10^{-12}$$

This way, starting from the algebraic result obtained on real numbers

$$|u| \leq \sqrt{(C_c \ D_c) \cdot Q_\mu^{-1} \cdot (C_c \ D_c)^T}$$

we can ensure that with floating-point numbers, the inequality holds:

$$|u| \leq U_f = \sqrt{(C_c^f \ D_c^f) \cdot Q_\mu^{-1} \cdot (C_c^f \ D_c^f)^T} + e$$

which leads to the invariants:

$$\begin{aligned} & \% \begin{pmatrix} x_c \\ y_c \end{pmatrix} \in \mathcal{E}_{Q_\mu}, \quad Q_\mu = \begin{pmatrix} \mu P & 0_{2 \times 1} \\ 0_{1 \times 2} & 1 - \mu \end{pmatrix}, \quad \mu = 0.9991 \\ 10 \quad & \mathbf{u} = \mathbf{Ccf} * \mathbf{xc} + \mathbf{Dcf} * \mathbf{yc}; \\ & \% \begin{pmatrix} x_c \\ y_c \end{pmatrix} \in \mathcal{E}_{Q_\mu}, \quad u^2 \leq U_f^2 \end{aligned}$$

#### 4.4 Invariant on $x_c$

The next instruction, considering constant changes, is:

$$\begin{aligned} 11 \quad & \mathbf{xc} = \mathbf{Acf} * \mathbf{xc} + \mathbf{Bcf} * \mathbf{yc}; \\ & \% x_c \in \mathcal{E}_{\tilde{P}_f}, \quad \tilde{P}_f = [(A_c^f \ B_c^f) \cdot Q_\mu^{-1} \cdot (A_c^f \ B_c^f)^T]^{-1}, \quad u^2 \leq (C_c \ D_c) \cdot Q_\mu^{-1} \cdot (C_c \ D_c)^{-1} \end{aligned}$$

where  $\tilde{P}_f$  is defined the same way  $\tilde{P}$  is, using floating-point terms  $A_c^f$ ,  $B_c^f$  instead of the real-valued counterparts  $A_c$ ,  $B_c$ . Again, this invariant holds independently of matrices values.

Here, we compute

$$\begin{cases} A_{c(0,0)}^f x_{c1} + A_{c(0,1)}^f x_{c2} + y_c \\ A_{c(1,0)}^f x_{c1} + x_{c2} \end{cases}$$

Using the same method as above, absolute rounding errors introduced by floating-point operations can be bounded by constants

$$e_1 = 7.42 \cdot 10^{-15}, \quad e_2 = 3.62 \cdot 10^{-15}$$

<sup>4</sup> Rangelab source file is available at: [http://www.cri.enscm.fr/people/maisonneuve/lyafloat/resources/lyafloat\\_fp.m](http://www.cri.enscm.fr/people/maisonneuve/lyafloat/resources/lyafloat_fp.m).

These constants must be taken into account in the postcondition. Also, invariant on  $u$  must be replaced as previously. Then the postcondition can be replaced by

$$\% x_c \in \mathcal{E}_{\tilde{P}_f}^f, \quad u^2 \leq U_f^2$$

where  $\mathcal{E}_{\tilde{P}_f}^f$  is an ellipse that includes  $\mathcal{E}_{\tilde{P}_f}$  plus the rounding error terms (see Figure 5). Replacing the ellipse in the postcondition by another ellipse has the advantage of introducing little change in the stability proof sketch (instead of using a different domain, which would involve using different theorems), which can greatly facilitate tweaking the rest of the proof in longer codes. Formally,  $\mathcal{E}_{\tilde{P}_f}^f$  must satisfy:

$$\forall x_c \in \mathcal{E}_{\tilde{P}_f}, \forall x'_c \in \mathbb{R}^2, |x'_{c_1} - x_{c_1}| \leq e_1 \wedge |x'_{c_2} - x_{c_2}| \leq e_2 \implies x'_c \in \mathcal{E}_{\tilde{P}_f}^f \quad (3)$$

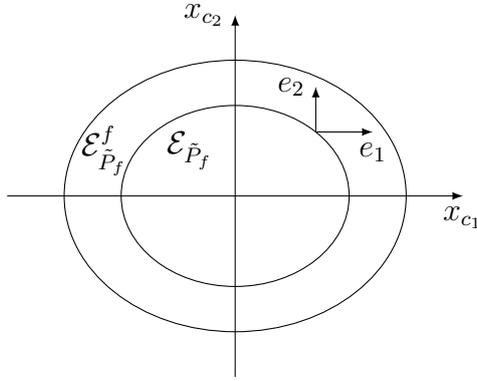


Fig. 5. Relation between  $\mathcal{E}_{\tilde{P}_f}$  and  $\mathcal{E}_{\tilde{P}_f}^f$

At the end of the proof scheme, the system is stable with floating-point numbers if and only if the inclusion

$$\mathcal{E}_{\tilde{P}_f}^f \subset \mathcal{E}_P$$

holds. To succeed,  $\mathcal{E}_{\tilde{P}_f}^f$  should be as narrow as possible with respect to Equation (3). This is not a clear criterion, as several shapes are possible for  $\mathcal{E}_{\tilde{P}_f}^f$  with no clear winner. We propose to define  $\mathcal{E}_{\tilde{P}_f}^f$  as the smallest homothety of  $\mathcal{E}_{\tilde{P}_f}$  centered around 0 that satisfies (3). It can be computed rather easily, for any number of dimension; we give details for two dimensions.

Let  $a, b, c$  be the coefficients of  $\mathcal{E}_{\tilde{P}_f}$ :

$$\mathcal{E}_{\tilde{P}_f} = \{(x_{c_1}, x_{c_2}) \mid ax_{c_1}^2 + bx_{c_2}^2 + cx_{c_1}x_{c_2} \leq 1\}.$$

$a$ ,  $b$  and  $c$  are known, positive values. Then there exists  $k \geq 0$  s.t.

$$\mathcal{E}_{\tilde{P}_f}^f = \{(x_{c_1}, x_{c_2}) \mid ax_{c_1}^2 + bx_{c_2}^2 + cx_{c_1}x_{c_2} \leq k\}.$$

As  $\mathcal{E}_{\tilde{P}_f}^f$  is wider than  $\mathcal{E}_{\tilde{P}_f}$ ,  $k \geq 1$ . We need a condition on  $k$  that guarantees (3).

We consider a point  $(x_{c_1}, x_{c_2})$  located on the border of  $\mathcal{E}_{\tilde{P}_f}$ :

$$ax_{c_1}^2 + bx_{c_2}^2 + cx_{c_1}x_{c_2} = 1. \quad (4)$$

By construction, for any values  $\epsilon_1, \epsilon_2 \in \mathbb{R}$  s.t.  $|\epsilon_1| \leq e_1 \wedge |\epsilon_2| \leq e_2$ , the relation

$$(x_{c_1} + \epsilon_1, x_{c_2} + \epsilon_2) \in \mathcal{E}_{\tilde{P}_f}^f$$

must hold, that is to say:

$$a(x_{c_1} + \epsilon_1)^2 + b(x_{c_2} + \epsilon_2)^2 + c(x_{c_1} + \epsilon_1)(x_{c_2} + \epsilon_2) \leq k.$$

It develops into

$$(ax_{c_1}^2 + bx_{c_2}^2 + cx_{c_1}x_{c_2}) + (2a\epsilon_1 + c\epsilon_2)x_{c_1} + (2b\epsilon_2 + c\epsilon_1)x_{c_2} + (a\epsilon_1^2 + b\epsilon_2^2 + c\epsilon_1\epsilon_2) \leq k,$$

that is

$$1 + (2a\epsilon_1 + c\epsilon_2)x_{c_1} + (2b\epsilon_2 + c\epsilon_1)x_{c_2} + (a\epsilon_1^2 + b\epsilon_2^2 + c\epsilon_1\epsilon_2) \leq k.$$

due to (4).

Greatest values for the left-hand term are reached with  $\epsilon_1 = |e_1| \wedge \epsilon_2 = |e_2|$ , depending on the signs of  $x_{c_1}$  and  $x_{c_2}$ . As the ellipse  $\mathcal{E}_{\tilde{P}_f}$  is symmetric about the origin point  $(0, 0)$ , we can set  $\epsilon_1 = e_1$ , which lets only two cases to study. Finally, we numerically verify that greatest values of the term are reached when  $\epsilon_2 = e_2$ . This is the only case we detail here.

We can write:

$$1 + \alpha x_{c_1} + \beta x_{c_2} + \gamma \leq k$$

with values  $\alpha = (2ae_1 + ce_2)$ ,  $\beta = (2be_2 + ce_1)$  and  $\gamma = (ae_1^2 + be_2^2 + ce_1e_2)$ .

We known that  $x_{c_1}$  and  $x_{c_2}$  are bounded, thus so is the term  $\alpha x_{c_1} + \beta x_{c_2}$ : we can compute a minimum bound  $\delta$  s.t.  $\alpha x_{c_1} + \beta x_{c_2} \leq \delta$ . So it is sufficient that  $k$  satisfies:

$$k \geq 1 + \gamma + \delta$$

Consequently, we define  $\mathcal{E}_{\tilde{P}_f}^f$  as the smallest homothety of  $\mathcal{E}_{\tilde{P}_f}$  that satisfy (3), obtained with  $k = 1 + \gamma + \delta$ . The instruction becomes:

$$\% \begin{pmatrix} x_c \\ y_c \end{pmatrix} \in \mathcal{E}_{Q_\mu}, \quad u^2 \leq U_f^2$$

```

11   xc = Ac*xc + Bc*yc;
      %  $x_c \in \mathcal{E}_{\tilde{P}_f}^f, \quad u^2 \leq U_f^2$ 

```

In our case, starting from the ellipse

$$\mathcal{E}_{\tilde{P}_f} = \{(x_{c_1}, x_{c_2}) \mid 0.00269007x_{c_1}^2 + 0.000341414x_{c_1}x_{c_2} + 0.00247323x_{c_2}^2 \leq 1\}$$

we get the following values:

$$\alpha = 1.03246 \cdot 10^{-17}, \quad \beta = 1.84829 \cdot 10^{-17}, \quad \gamma = 7.17582 \cdot 10^{-32}.$$

Using **Mathematica**, we found  $\delta = 5.35754 \cdot 10^{-16} \gg \gamma$  and finally

$$k = 1 + 5.35754 \cdot 10^{-16}$$

that gives  $\mathcal{E}_{\tilde{P}_f}^f$ .

#### 4.5 End of Proof Scheme

Then, what remains of the stability proof scheme becomes:

```

      %  $x_c \in \mathcal{E}_{\tilde{P}}^f, \quad u^2 \leq U_f^2$ 
12   send(u, 1);
      %  $x_c \in \mathcal{E}_{\tilde{P}}^f$ 
13   receive(y, 2);
      %  $x_c \in \mathcal{E}_{\tilde{P}}^f$ 
14   receive(yd, 3);
      %  $x_c \in \mathcal{E}_{\tilde{P}}^f$ 
15   skip;
      %  $x_c \in \mathcal{E}_P$ 
16 end

```

As previously, the final assertion  $\mathcal{E}_{\tilde{P}_f}^f \subset \mathcal{E}_P$  must be checked. Two cases are possible:

- either  $\mathcal{E}_{\tilde{P}_f}^f \subset \mathcal{E}_P$ , then we proved that the program is Lyapunov stable on a floating-point architecture;
- or  $\mathcal{E}_{\tilde{P}_f}^f \not\subset \mathcal{E}_P$ : as  $\mathcal{E}_{\tilde{P}_f}^f$  was obtained through overapproximations, we cannot conclude about the program behavior.

In this open-loop case, we are able to check that  $\mathcal{E}_{\tilde{P}_f}^f \subset \mathcal{E}_P$  (**Mathematica** script is available online: see footnote 2). Thus, the stability of the open-loop system with a 64-bit IEEE 754 compliant implementation is formally proven to hold using our proof translation scheme.

## 5 Closed-Loop Stability Proof

We now show how the proof of state boundedness of the closed-loop system specifications can be migrated to the level of the controller code and executable model of the system. To be more precise, we exploit the invariance of the ellipsoid  $\mathcal{E}_P$  to develop a proof of proper behavior, that is, stability and variable boundedness, for the computer program that implements the controller as it interacts with the physical system. Unlike the developments related to open-loop controller, this proof necessarily involves the presence of the physical system. In [3], Feron chooses to represent the physical system and the computer program by two concurrent programs, as shown below.

Controller dynamics:	Physical system dynamics:
<pre> 1c Ac = [0.4990, -0.0500;         0.0100, 1.0000]; 2c Bc = [1; 0]; 3c Cc = [564.48, 0]; 4c Dc = -1280; 5c xc = zeros(2, 1); 6c receive(y, 2); receive(yd,         3); 7c while (1) 8c     yc = max(min(y - yd, 1),         -1); 9c     u = Cc*xc + Dc*yc; 10c    xc = Ac*xc + Bc*yc; 11c    send(u, 1); 12c    receive(y, 2); 13c    receive(yd, 3); 14c end </pre>	<pre> 1p Ap = [1.0000, 0.0100;         -0.0100, 1.0000]; 2p Bp = [0.00005; 0.01]; 3p Cp = [1, 0]; 4p while (1) 5p     yp = Cp * xp; 6p     send(yp, 2); 7p     receive(up, 1); 8p     xp = Ap * xp + Bp * up; 9p end </pre>

In this scheme, the computer program representation of the physical system is to remain unchanged, since it only exists for modeling purposes and does not correspond to any actual program, whereas the controller code is allowed to evolve to reflect the various stages of its implementation.

Establishing proofs of stability of the closed-loop system at the code level is necessarily tied to understanding the joint behavior of the controller and the plant. The entire state space therefore consists of the direct sum of state spaces of the controller and the physical system. The approach described in the previous sections is used to document the corresponding system of two processes. One interesting aspect of these processes is their concurrency, which can complicate the structure of the state transitions. However, a close inspection of the programs reveals that the transition structure of the processes

does not need to rely on the extensions of Hoare’s logic to concurrent programs: one program at a time is running, through the blocking nature of the `receive` primitive.

Feron’s stability proof with real numbers is much longer than for the open-loop system. We do not detail it, the interested reader is referred to [3] for full information. To be noticed, the resulting comments are not much more complex than those available from the study of the controller alone. On the good side, as already mentioned, the Hoare formalism is not significantly affected by the concurrent structure of the closed-loop system.

A floating-point representation of the closed-loop system consists of keeping the right column of the listing above in its original settings, while replacing the left column with the corresponding floating-point implementation, as we did in Section 4. Using similar techniques to the study of the controller alone, proof invariants can be tweaked to take into account constant changes and rounding errors resulting from the use of floating-point arithmetic. Unfortunately, using these invariants it cannot be shown that the stability condition holds at the end of the loop body. In this case, we are unable to prove the system stability on a floating-point architecture: either the system is not stable with the floating-point based controller, or the proof parameters ( $\mathcal{E}_P$ ,  $\mu$ , ...) must be chosen more carefully by the controller designer.

## 6 Conclusion

The general idea is to replace some of the invariants in the original proof scheme by wider ones that include rounding errors, with the hope that the stability condition is strong enough and still holds. This approach is made possible by the fact that rounding errors introduced by the operations used in the code are bounded on bounded inputs and bounded controller state variables.

In this document, we study the case of a floating-point representation of real numbers. They are not available on all architectures, especially on microcontrollers that are commonly used to implement control systems. We quickly discuss the alternative real-number representations.

- We can deal with fixed-point arithmetic the same way we do with floating-point, as long as we stand far enough from extremal values that can lead to overflows.
- Another way to represent real numbers is to use two integers, a numerator and a denominator. Considering that the input values are exact, the elementary operations do not introduce rounding errors but can easily lead to

overflows, e.g. when computing

$$\frac{p_1}{q_1} + \frac{p_2}{q_2} = \frac{p_1q_2 + p_2q_1}{q_1q_2}.$$

A strategy must be used to prevent overflows by introducing approximations: in this case, the question is to quantify the errors introduced by these approximations.

In our example, we exclusively used additions and multiplications: divisions are not involved in linear control. Still, programs with divisions can also be analyzed, if the numerator can be shown to be far enough from zero: it is a supplementary constraint, but it is reasonable to assume that it should be respected on a realistic control system that uses divisions. Differentiable, periodic functions such as (sin) can be computed with an abacus and an interpolation function, thus with bounded error. In the same way, functions not periodic, but restricted to finite domains, can also be approximated. Other functions, such as (tan) or  $\sqrt{\quad}$ , should raise more issues.

## References

- [1] Boyd, S., L. El Ghaoui, E. Feron and V. Balakrishnan, “Linear Matrix Inequalities in System and Control Theory,” Society for Industrial and Applied Mathematics, 1994.  
URL <http://epubs.siam.org/doi/abs/10.1137/1.9781611970777>
- [2] Cousot, P., R. Cousot, J. Feret, A. Miné and X. Rival, *The astrée static analyzer* (2013).  
URL <http://www.astree.ens.fr/>
- [3] Feron, E., *From control systems to control software*, IEEE Control Systems Magazine **30** (2010), pp. 50–71.  
URL <http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=5643477>
- [4] Goldberg, D., *What every computer scientist should know about floating-point arithmetic*, ACM Computing Surveys **23** (1991), pp. 5–48.  
URL <http://portal.acm.org/citation.cfm?doid=103162.103163>
- [5] Higham, N. J., “Accuracy and Stability of Numerical Algorithms,” SIAM, 2002.
- [6] IEEE Computer Society, *IEEE standard for floating-point arithmetic*, Technical report (2008).  
URL <http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=4610935>
- [7] INRIA, *The coq proof assistant* (2013).  
URL <http://coq.inria.fr/>
- [8] Martel, M., *Rangelab* (2011).  
URL <http://perso.univ-perp.fr/mmartel/rangelab.html>
- [9] Owre, S., *PVS specification and verification system* (2013).  
URL <http://pvs.csl.sri.com/>
- [10] Paulson, L., T. Nipkow and M. Wenzel, *Isabelle* (2013).  
URL <http://isabelle.in.tum.de/>
- [11] Peled, D., “Software reliability methods,” Springer, 2001.
- [12] Wolfram Research, *Mathematica* (2013).  
URL <http://www.wolfram.com/mathematica/>