HAL open science

Des réels aux flottants : préservation automatique de preuves de stabilité de Lyapunov

Olivier Hermant, Vivien Maisonneuve

▶ To cite this version:

Olivier Hermant, Vivien Maisonneuve. Des réels aux flottants : préservation automatique de preuves de stabilité de Lyapunov. AFADL, Pascale Le Gall; Frederic Dadeau, Jun 2015, Bordeaux, France. pp.51-56. hal-01138327

HAL Id: hal-01138327

https://minesparis-psl.hal.science/hal-01138327

Submitted on 1 Apr 2015

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.

Distributed under a Creative Commons Attribution - ShareAlike 4.0 International License

Des réels aux flottants : préservation automatique de preuves de stabilité de Lyapunov

Olivier Hermant et Vivien Maisonneuve

Abstract

Les programmes contrôle-commande doivent assurer la stabilité d'un système dynamique autour d'un point d'équilibre. De tels programmes sont généralement issus d'une modélisation du système, en partie intégrée au programme, qui calcule alors à chaque cycle la commande adéquate à envoyer au système.

Les concepteurs de tels systèmes s'appuient sur la théorie de Lyapunov pour déterminer les paramètres internes nécessaires à la stabilité. Il s'agit donc d'un cas rare, où l'on dispose *en amont* d'une preuve que le système respecte la propriété voulue.

Nous nous intéressons au problème du transfert de ces preuves de la modélisation vers l'implémentation, et plus particulièrement au problème du passage des nombres réels, utilisés pour la modélisation, aux nombres flottants, qui sont utilisés dans l'implantation du programme sur microcontrôleur.

Pour cela, nous introduisons l'outil LyaFloat, qui permet de vérifier à quelles conditions de précision une preuve de stabilité, donnée en tant qu'annotation dans une logique de Hoare, est préservée ou non lors du passage aux flottants.

1 Introduction

La stabilité est un attribut essentiel des systèmes de contrôle, en particulier lorsqu'ils commandent des systèmes physiques dits critiques, dont un dysfonctionnement peut engendrer des pertes humaines ou matérielles importantes. De nombreuses techniques complémentaires doivent donc être mises en œuvre pour s'en assurer.

Dans le cadre des programmes de type contrôle-commande qui sont l'objet d'étude de l'automatique, la théorie de Lyapunov joue un rôle central. Elle permet, après modélisation de l'environnement physique dans lequel évolue le système contrôlé, de déterminer les valeurs des paramètres constants qui permettront aux variables d'état du système de rester contenues dans une *enveloppe bornée*, autrement dit, d'assurer la *stabilité* et ce, en boucle fermée ou en boucle ouverte, c'est à dire en considérant le modèle de l'environnement ou non.

Ainsi, les concepteurs de tels contrôleurs doivent fournir tout le travail de modélisation et de preuve nécessaire. Mais ce travail s'arrête le plus souvent à un modèle du programme, généralement du code de haut niveau, de type MATLAB ou Simulink. Tout ce travail est perdu ensuite lors des étapes successives menant au code exécuté sur microcontrôleur, et doit parfois être retrouvé [2, 6] a posteriori.

Pour pallier ce problème, Eric Féron [3] propose d'utiliser une logique de Hoare [5], qui permet de propager les invariants quadratiques de Lyapunov vers du code plus bas niveau (en l'occurrence, du code `C`). Cependant, bien que les étapes (par exemple multiplication par une matrice) soient de ce fait plus atomiques, le raisonnement continue de se faire sur des nombres réels, alors que les variables et les constantes du code exécuté sont à précision finie et que les opérations arithmétiques introduisent des erreurs d'arrondi.

C'est ce problème que nous proposons d'étudier. Étant donné un programme et une preuve de sa stabilité par la théorie de Lyapunov, le programme LyaFloat que nous introduisons permet de vérifier automatiquement, si cela est possible, que cette preuve est toujours valide lorsque l'on considère non plus les nombres réels, mais les flottants. De plus, il est possible de faire varier dans LyaFloat la précision des nombres considérés, permettant ainsi de déterminer à quelles conditions de précision sur les nombres la preuve reste valide.

À la suite de Féron [3], nous considérerons donc donnés un programme *numérique*, comportant uniquement des constantes, des opérations arithmétiques ou de saturation $(\max, \min)$, ainsi que des assignations de variables. Ce programme est accompagné, pour chaque instruction, de deux invariants *quadratiques*, la pré- et la postcondition, donnés en logique de Hoare, qui sont supposés valides pour des nombres réels et des opérations arithmétiques exactes.

Ces invariants quadratiques proviennent de la structure particulière du code des programmes contrôle-commande et sont donnés par la théorie de Lyapunov. L'outil LyaFloat considère l'invariant d'entrée, et le propage en prenant en compte les constantes altérées et les erreurs d'arrondi des nombres flottants. En fin de programme, la stabilité du système en nombres flottants est vérifiée. Elle se traduit par l'inclusion de l'ellipsoïde (domaine résultant d'un invariant quadratique borné) final dans l'ellipsoïde initial.

D'autres travaux se sont récemment intéressés au problème de la stabilité des programmes *concrets*. Citons en particulier l'approche d'Astrée [2] ainsi que celle de Pierre Roux [6]. Toutes deux s'intéressent au seul problème en boucle ouverte, et s'efforcent de reconstruire des invariants corrects en suivant une approche par interprétation abstraite (sur des domaines ellipsoïdaux). Dans le cas de [6], un ensemble de condition général est même donné. Notre travail adopte une démarche légèrement différente, comme il a déjà été souligné: nous proposons de réutiliser les connaissances spécifiques et les preuves de haut niveau développées par les concepteurs, ce qui nous permet entre autre de traiter les deux cas boucle ouverte et boucle fermée.

Après un survol du domaine des programmes contrôle-commande provenant de l'automatique, nous présentons la structure de la traduction que nous proposons, validée en Coq. L'outil concret LyaFloat est ensuite introduit, ainsi que son application au cas d'étude présenté par Féron [3]. Nous discutons ensuite du travail encore à réaliser. Ces travaux ont fait l'objet d'une partie de la thèse de doctorat de Vivien Maisonneuve [4].

2 Structure d'un programme contrôle-commande

Un programme contrôle-commande reçoit des entrées, provenant du système physique et d'un opérateur et, en fonction de celles-ci et de son état interne, produit des sorties, destinées à des actuateurs, qui eux-mêmes influent sur le système, comme décrit par la Fig. 1.

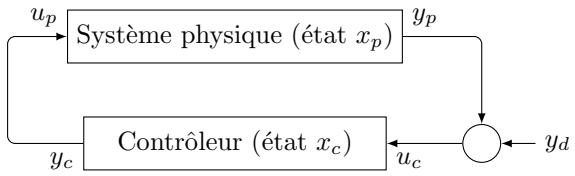

Figure 1: Schéma d'un système contrôle-commande

Le système physique est tout d'abord modélisé de façon continue, avec des équations différentielles. Le modèle est ensuite linéarisé autour du point où l'on recherche la stabilité, afin de permet la conception du contrôleur. Le contrôleur final est, quant à lui, discret, le code lira donc ses entrées et produit ses sorties en boucle à une fréquence donnée. Il est d'autre part linéaire (avec éventuelle saturation des entrées/sorties), les opérations mathématiques en jeu sont donc des opérations matricielles, comme dans le code MATLAB de la Fig. 2, issu de [3].

```
Ac = [0.4990, -0.0500; 0.0100, 1.0000];
Bc = [1; 0];
Cc = [564.48, 0];
Dc = -1280;
xc = zeros(2, 1);
receive(y, 2); receive(yd, 3);
while (1)
    yc = max(min(y - yd, 1), -1);
    skip;
    u = Cc*xc + Dc*yc;
    xc = Ac*xc + Bc*yc;
    send(u, 1);
    receive(y, 2);
    receive(yd, 3);
    skip;
end
```

Figure 2: Code MATLAB d'un contrôleur masse-ressort [3]

Les étapes de modélisation du problème, de linéarisation, de conception du contrôleur ainsi que de passage du continu au discret sont l'objet d'étude de l'automatique, qui fournit ensuite des garanties sur le système obtenu, sous la forme de preuves de stabilité, sur le système en boucle fermé (représenté par l'intégralité du schéma de la Fig. 1) ainsi que sur le système en boucle ouvert

(lorsque l'on considère le contrôleur de la Fig. 1 seul).

Ceci se traduit sur le code de la Fig. 2 en des invariants elliptiques qui sont déterminés par la théorie de Lyapunov et propagés par le code, et que l'on peut présenter en une logique de Hoare.

3 Structure et validité de la traduction

Ainsi, comme indiqué par Féron [3], chaque instruction se retrouve décorée d'une paire pré-/postcondition, la postcondition de l'instruction i étant à la fois:

- le résultat de la transformation de la précondition par i, et éventuellement de l'application de théorèmes,
- et la précondition de l'instruction suivante.

La précondition initiale est que le vecteur des variables d'état du contrôleur se trouve dans une certaine enveloppe: $x_c \in \mathcal{E}_P \Leftrightarrow x_c^{\mathrm{T}} \cdot P \cdot x_c \leq 1$, avec P une matrice définie positive. La postcondition de fin de corps de boucle, est $x_c \in \mathcal{E}_R \Leftrightarrow x_c^{\mathrm{T}} \cdot R \cdot x_c \leq 1$, avec R définie positive. La stabilité est alors une conséquence de $\mathcal{E}_R \subseteq \mathcal{E}_P$.

C'est ce que nous cherchons à vérifier avec une traduction en deux étapes, comme illustré dans la Fig. 3, où est schématiquement représentée l'instruction i, les invariant d'entrée d et de sortie d'. Ce dernier est le résultat de l'application du théorème p à l'instruction i en supposant les variables dans le domaine d .

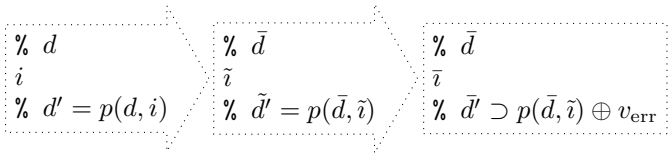

Figure 3: Schéma de la Traduction

Cette traduction de l'instruction i est effectuée en deux étapes, qui ont un impact sur les pré- et postconditions: tout d'abord, nous générons $\tilde{\imath}$, où les constantes réelles sont remplacées par leur représentation flottante, mais où les opérations arithmétiques sont toujours exactes.

La précondition d a potentiellement changé, en particulier dans le cas où l'instruction i n'est pas la première instruction de la boucle. C'est pourquoi l'on condière un invariant d'entrée modifié, $\overline{d}$, qui est propagé avec le même argument p, appliqué maintenant à l'instruction $\tilde{\imath}$.

La seconde étape de la traduction remplace les opérateurs arithmétiques exacts de $\tilde{\imath}$ par leurs représentations en flottant pour obtenir $\bar{\imath}$. Malheureusement, les résultats de la théorie de Lyapunov ne s'appliquent qu'aux des opérations exactes (par exemple, l'addition flottante n'est pas associative). Il est ainsi impossible d'appliquer directement p à l'instruction $\bar{\imath}$ résultante. Aussi, nous conservons l'invariant de l'étape précédente $p(\bar{d}, \tilde{\imath})$, que nous élargissons en rajoutant l'erreur maximale obtenue par des opérations arithmétiques v_{err}, qui dépend de $\tilde{\imath}$ et de $\bar{d}$. Cette erreur est fonction des bornes sur les opérandes (connues grâce à $\bar{d}$).

Comme nous chercherons, en pratique, à conserver une structure d'ellipsoïde pour les invariants, de manière à rester dans le cadre de la théorie de Lyapunov, nous considérons en réalité une surapproximation $\bar{d}'$ de cet invariant.

Cette approche théorique a été formalisée en Coq, dans le cas des opérateurs binaires. L'objet primitif de cette formalisation sont les domaines, des sous-ensembles de réels (cas abstrait) ou de flottants (cas concret), avec des conditions de cohérence issues de l'interprétation abstraite entre les deux. De cette manière, nous restons le plus générique possible. Les théorèmes sont des transformateurs de domaine qui sont supposés valides sur n'importe quel domaine, y compris non ellipsoïdal. En effet il suffit, dans ce dernier cas, de renvoyer l'ensemble $\mathbb{R}$ entier, ce qui équivaut à une perte totale d'information. Pour plus de détails, voir [4], Chap. 5.

4 LyaFloat

LyaFloat est une implantation concrète de l'architecture présentée Fig. 3. Il s'agit d'un programme Python, qui s'appuie sur les deux librairies SymPy, pour manipuler les calculs mathématiques symboliques, et Mpmath, pour l'arithmétique flottante en précision arbitraire. Le calcul de l'erreur maximale v_{err}, en fonction des bornes sur les valeurs des variables, est défini par la norme IEEE 754.

Comme mentionné Sec. 3, nous choisissons de surapproximer la postcondition obtenue, afin de conserver la forme ellipsoïdale de l'invariant. Comme il n'y a pas de critère unique de minimalité pour une ellipsoïde devant contenir un certain domaine, un choix heuristique est fait, voir [4] pour plus de détails.

En boucle ouverte, la ligne 11 de l'exemple de la Fig. 2 se traduit par la Fig. 4. La précondition de cette ligne est l'invariant suivant, où `Zc` étant le vecteur à trois dimensions composé de `xc` et de `yc` (cf. [3, 4]):

$$\texttt{Zc} \in \mathcal{E}_{Q_\mu}, Q_\mu = \left(\begin{smallmatrix} \mu P & 0_{2\times 1} \\ 0_{1\times 2} & 1-\mu \end{smallmatrix}\right), u^2 \leq (C_c\ D_c) \cdot Q_\mu^{-1} \cdot (C_c\ D_c)^{-1}$$

On vérifie finalement l'inclusion de l'ellipsoïde élargie dans $\mathcal{E}_P$, ce qui est possible dans le cas de flottants sur 64 bits. Il a aussi été possible de vérifier la stabilité d'exemples tirés de la littérature, notamment de [6].

Comme il est d'autre part possible de régler la précision des nombres flottants (grâce à la primitive `setfloatify`), on s'aperçoit que le système de la Fig. 2 reste stable jusqu'à une précision de 17 bits pour les nombres flottants, ce qui permettrait par exemple d'opter pour un microcontrôleur manipulant des flottants moins précis.

Dans le cas où nous choisissons une précision inférieure, nous ne sommes plus en mesure de conclure: soit le système est réellement instable, soit trop d'imprécision a été introduit, notamment dans le calcul de l'ellipsoïde englobante $\bar{d}' \subseteq p(\bar{d}, \tilde{\imath})$ ou dans le calcul de l'erreur arithmétique maximale v_{err}.

Il est intéressant de noter que nous avons aussi été en mesure de traiter également le cas de la boucle fermée, ce qui va au delà des approches par interprétation abstraite [3, 6]. Il s'avère impossible de conclure à la stabilité dans le cas de flottants sur 64 bits, il nous faut pour ce faire passer à une précision d'au moins 117 bits, par exemple des flottants en quadruple précision. Notons que le modèle du système physique continue à avoir des constantes et opérations réelles, car il ne correspond à aucun code concret. Ainsi les étapes de traduction ne se font que sur le contrôleur.

```
from lyafloat import *
# Parameters
setfloatify(constants=True, operators=True, precision=53)

# Definition of $\EllP$
P = Rational("1e-3") * Matrix(rationals(["0.6742 0.0428", "0.0428 2.4651"]))
EP = Ellipsoid(P)

# Definition of $\EllQmu$
mu = Rational("0.9991")
Qmu = mu * P
Qmu = Qmu.col_insert(2, zeros(2, 1)).row_insert(2, zeros(1, 3))
Qmu[2,2] = 1 - mu
EQmu = Ellipsoid(Qmu)

# Symbols
xc1, xc2, yc = symbols("xc1 xc2 yc")
Xc = Matrix([[xc1], [xc2]])
Yc = Matrix([[yc]])
Zc = Matrix([[xc1], [xc2], [yc]])

# Constant matrices
Ac = Matrix(constants(["0.4990 -0.0500", "0.0100 1.0000"]))
Bc = Matrix(constants(["1", "0"]))
Cc = Matrix(constants(["564.48 0"]))
Dc = Matrix(constants(["-1280"]))

# Definition and verification of $\EllR$
AcBc = Ac.col_insert(Ac.cols, Bc)
R = (AcBc * Qmu.inv() * AcBc.T).inv()
ER = Ellipsoid(R)
print("ER included in EP :", ER <= EP)

# Computation and verification of $\widefp\EllR$
i = Instruction({Xc: Ac * Xc + Bc * Yc},
       pre=[Zc in EQmu], post=[Xc in ER])
ERbar = i.post()[Xc]
print("ERbar =", ERbar)
print("ERbar included in EP :", ERbar <= EP)
```

Figure 4: Vérification de stablité en boucle ouverte avec LyaFloat

5 Conclusion

De nombreuses améliorations peuvent encore être apportées à notre outil. Tout d'abord, pour atteindre un plus grand degré de confiance dans la preuve de stabilité, nous pourrions soit développer et prouver correct notre vérificateur en Coq, soit demander à LyaFloat de générer un script de preuve Coq démontrant la stabilité en flottants, par exemple en utilisant la librairie Flocq[1]. C'est cette seconde approche que nous privilégierons en premier lieu.

Des cas d'études à plus grande échelle sont aussi envisagés, en collaboration avec le Centre automatique et systèmes à MINES ParisTech. Afin d'en faire un outil facilement utilisable par les automaticiens, il faudra aussi développer un front-end moins bas niveau.

References

[1] Sylvie Boldo and Guillaume Melquiond. Flocq, 2010.

[2] Patrick Cousot, Radhia Cousot, Jérôme Feret, Antoine Miné, and Xavier Rival. The astrée static analyzer, 2013.

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

[4] Vivien Maisonneuve. *Static Analysis of Control-Command Systems – Floating-Point and Integer Invariants.* PhD thesis, MINES ParisTech, 2015.

[5] Doron Peled. *Software reliability methods.* Springer, 2001.

[6] Pierre Roux. *Analyse statique de système contrôle commande: synthèse d'invariants non linéaires.* PhD thesis, Université de Toulouse, 2013.