Mémoire de Master
Master Parisien de Recherche en Informatique

2 Septembre 2005

Calcul Efficace de l'Ensemble Atteignable des Systèmes Linéaires avec Incertitudes

Colas Le Guernic
ENS & PARIS VII

Sous la direction de Oded Maler, VERIMAG

Remerciements

Je tiens à remercier Antoine Girard et Goran Frehse pour m'avoir hébergé pendant mon séjour à l'Université de Pennsylvannie et à l'Université de Carnegie Mellon, ainsi que les professeurs Pappas et Krogh qui m'ont accueilli au sein de leurs laboratoires respectifs.

Je remercie également Antoine Girard pour son aide et ses remarques concernant ce rapport, ainsi que Thao Dang, Alexandre Donzé et Eugène Asarin avec qui j'ai également eu l'occasion d'interagir.

Je remercie enfin pariculièrement Oded Maler pour avoir encadré mon stage.

Résumé

Nous présentons dans ce rapport un nouvel algorithme pour le calcul d'ensemble atteignable par un système dynamique linéaire avec incertitude :

$$\dot{x} = Ax + Bu$$

où $\|Bu\|$ est borné.

Cet algorithme se base sur une discrétisation du problème, pour laquelle nous proposons une amélioration, et calcule en fait les premiers termes de la suite d'ensembles :

$$\Omega_{n+1} = A\Omega_n \oplus U$$

L'originialité de la méthode vient de la séparation de la transformation linéaire et de la somme de Minkowski. Ceci permet de s'affranchir de l'effet d'emballage puisque seule l'erreur initiale due à la discrétisation du système est propagée par le flot.

Bien que non spécifique à un type de donnée particulier, cette méthode semble particulièrement adaptée aux zonotopes. Les zonotopes forment une classe de polytopes présentant des propriétés intéressantes : ils ont un représentation compacte, sont clos par application linéaire et somme de Minkowski, la somme de Minkowski se fait en temps constant.

Plusieurs algorithmes se basant sur cette méthode sont proposés et comparés à d'autres méthodes existantes. Ils présentent une bonne complexité théorique, tant en temps (entre $\mathcal{O}(kd^3)$ et $\mathcal{O}(kd^2 + d^3)$ pour le plus rapide), qu'en espace (entre $\mathcal{O}(kd^2)$ et $\mathcal{O}(kd + d^2)$), où d est le nombre de variables continues, et k le nombre de pas de temps considérés. En pratique ils sont plus rapide, plus précis, et consomment moins de mémoire que les méthodes auxquelles on les a comparés. Ces dernières ayant l'avantage d'être plus générales que notre méthode qui utilise intensivement les redondances du problème.

Une extension au cas affine par morceaux est enfin proposée et testée sur deux exemples.

Table des matières

Introduction

Dans de nombreux domaines (biologie, chimie, ingénierie...) les objets étudiés présentent des aspects continus (température, vitesse...) et discrets (disparition/ajout d'un réactif, boîte de vitesse, interrupteur...). On appelle ces systèmes présentant deux types de dynamique des systèmes hybrides.

La vérification formelle, qui consiste à montrer que tous les comportements exhibés par un système, sous toutes les perturbations externes possibles satisfont, ou non, certaines propriétés est un des principaux enjeux de leur étude. Un outil de base important pour mener à bien cet objectif est le calcul d'ensembles atteignables, c'est à dire l'ensemble des points pour lesquels il existe une trajectoire du système les visitant.

Malheureusement ce problème est indécidable et on ne peut espérer qu'obtenir des sur-approximations de l'ensemble atteignable sur une durée bornée. Même dans ce cas, le problème reste difficile et on se restreint souvent à la classe des systèmes affines par morceaux. Elle présente un intérêt particulier, elle est suffisamment simple pour pouvoir espérer en faire une étude automatique, et suffisamment expressive pour pouvoir modéliser de façon réaliste de nombreux systèmes.

Plusieurs méthodes ont été proposées pour calculer l'ensemble atteignable par un système affine, mais quand on prend en compte l'incertitude sur les paramètres du système, qui peut être due à une mesure, incertaine par essence, ou une entrée inconnue, elles sont victimes de l'effet d'emballage : la propagation et l'accumulation d'erreurs dues aux approximations nécessaires. La méthode que nous présentons ici permet de s'affranchir de cet effet.

Nous commencerons par présenter plus en détail les systèmes hybrides, et mettrons en évidence l'importance des systèmes affines par morceaux. Nous nous ramènerons ensuite, via une discrétisation du problème, pour laquelle nous proposons une amélioration, à l'étude d'une suite d'ensembles, et présenterons la plupart des méthodes aujourd'hui utilisées pour en calculer les premiers termes. Nous présenterons enfin notre algorithme, et nous montrerons comment l'utiliser pour l'étude des systèmes affines par morceaux.

Chapitre 1

Systèmes hybrides

Dans ce chapitre nous allons aborder de manière intuitive la théorie des systèmes hybrides pour mettre en évidence le contexte dans lequel notre travail se situe. Pour un étude plus complète, voir [6, 10].

1.1 Définitions

Les systèmes hybrides recouvrent tous les systèmes impliquant des phénomènes continus et discrets, que ce soit de façon intrinsèque, ou pour simplifier une dynamique continue, comme par exemple, l'abstraction, souvent faite en biologie, d'une sigmoïde par deux (ou trois) modes de fonctionnement.

Nous avons donc besoin d'une définition générale [4, 10, 22] pour les décrire.

Définition 1 *Un automate hybride est un septuple*

$$\mathcal{H} = (\mathcal{Q}, \mathcal{E}, \mathcal{D}, \mathcal{U}, \mathcal{F}, \mathcal{G}, \mathcal{R})$$

où :

- $\mathcal{Q}$ *est l'ensemble des* états discrets
- $\mathcal{E} \subseteq \mathcal{Q} \times \mathcal{Q}$ *est l'ensemble des* transitions
- $\mathcal{D} = \{D_q \mid q \in \mathcal{Q}\}$ *est la collection des* domaines
  $\forall q \in \mathcal{Q}$, D_q *est un sous ensemble de* $\mathbb{R}^n$
- $\mathcal{U} = \{U_q \mid q \in \mathcal{Q}\}$ *est la collection des* domaines des entrées
  $\forall q \in \mathcal{Q}$, U_q *est un sous ensemble de* $\mathbb{R}^p$
- $\mathcal{F} = \{f_q \mid q \in \mathcal{Q}\}$ *est la collection des* champs de vecteurs
  $\forall q \in \mathcal{Q}$, $f_q : D_q \times U_q \to \mathbb{R}^n$
- $\mathcal{G} = \{G_e \mid e \in \mathcal{E}\}$ *est la collection des* gardes
  $\forall e = (q, q') \in \mathcal{E}$, $G_e \subseteq D_q \times U_q$

– $\mathcal{R} = \{R_e \mid e \in \mathcal{E}\}$ *est la collection des fonctions* resets
$\forall e = (q, q') \in \mathcal{E}$, $R_e : G_e \to 2^{D_{q'}}$

Moins formellement, un automate *hybride* peut être vu comme un automate *discret* auquel on ajoute des variables continues dont l'évolution est donnée par un système d'équations différentielles $\dot{x} = f_q(x, u)$ pour chaque état q, les transitions entre états q et q' sont déclenchées quand les variables continues atteignent une certaine partie de l'espace, la garde $G_{qq'}$, et la transition peut réinitialiser les variables continues par une fonction reset $R_{qq'}$.

Un exemple classique est le thermostat :

Example 1 *Le thermostat peut être modélisé par un automate hybride à trois états {POS, NEG, OFF} (figure 1.1). On veut maintenir une temperature de 20°, quand la température passe au dessus de 22° la climatisation se met en marche, et quand la température passe sous les 18° le chauffage se met en marche.*

Ici, la température, T, est une variable continue du système, et u et ϵ sont des variables d'entrée, u reflète l'influence extérieure, et ϵ l'incertitude sur la mesure de la température ($\epsilon(t)$ est la différence entre la température mesurée à l'instant t et la température réelle).

On pourrait encore compléter le modèle en ajoutant une variable d'entrée pour la température demandée (ici 20°), pour les seuils à partir desquels ont déclanche le chauffage ou la climatisation, pour l'intensité du chauffage ou de la climatisation...

On peut donc distinguer trois types d'entrée, l'influence de l'environnement, l'influence de l'opérateur, et les imperfections du système. Mais dans le cadre de l'analyse des systèmes, elles sont toutes trois sources d'*incertitude*. C'est pourquoi, dans la suite, nous parlerons d'incertitude.

1.2 Atteignabilité

Un problème crucial dans l'étude des systèmes hybrides est de déterminer si les exécutions du système rencontrent, ou non, certaines zones de l'espace, favorables, ou critiques.

Pour cela, on pourrait vouloir exécuter le système grâce à des techniques classiques d'intégration d'équations différentielles, mais à cause des incertitudes, plusieurs exécutions partant du même état initial peuvent produire des trajectoires différentes, on peut aussi par exemple vouloir considérer un ensemble de valeurs initiales pour les variables continues, et non une seule.

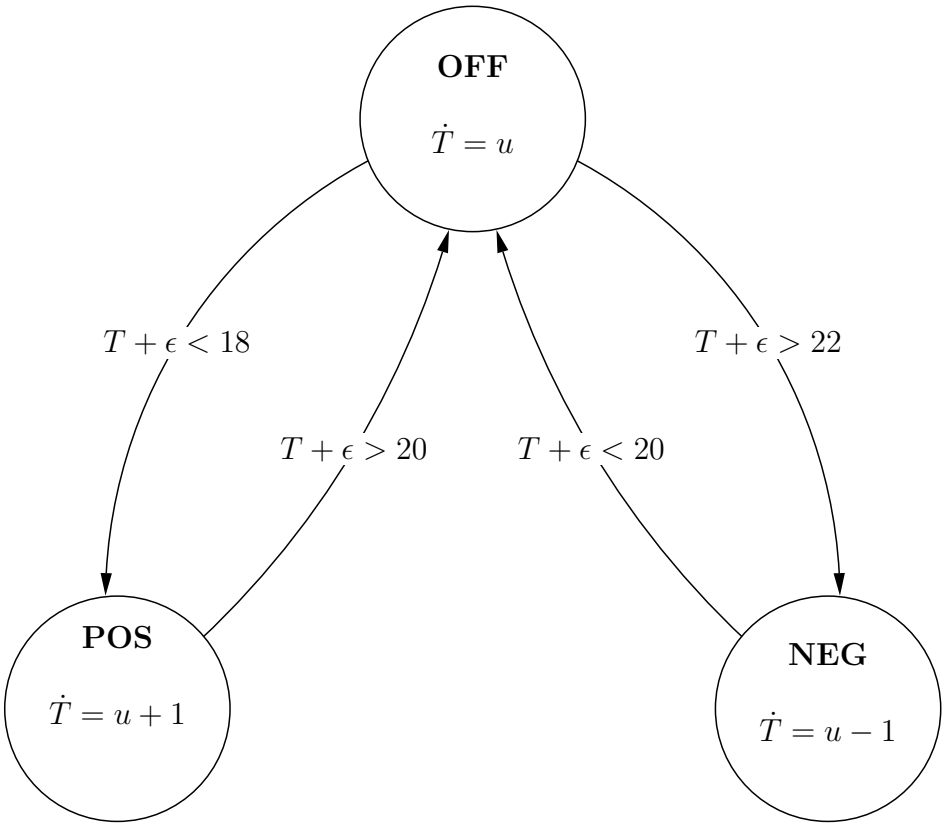


FIG. 1.1 – Automate hybride modélisant un thermostat

Le système peut donc générer un ensemble infini (non dénombrable) de trajectoires, il serait donc illusoire de vouloir faire une vérification systèmatique par simulation. On peut donc essayer de générer un ensemble de trajectoires *représentatives* de la dynamique du système, ou travailler sur des ensembles de points, nous nous intéressons ici à cette dernière approche.

On remplace l'équation différentielle avec entrée, par une inclusion différentielle, au sens de Aubin [2]. Ainsi, à partir de la dynamique non déterministe du système agissant sur un point, on construit une dynamique déterministe agissant sur un ensemble de points capturant tous les comportements possibles du système initial.

1.2.1 Dynamique continue

On se place ici dans un état discret q, la dynamique du système est donnée par :

$$\dot{x} = f(x, u) \tag{1.1}$$

On note $\Phi_t(I)$ l'ensemble atteignable à partir de I en un temps t :

$$\Phi_t(I) = \{y \in \mathbb{R}^n \mid \exists x \text{ solution de (1.1)}, x(0) \in I \wedge x(t) = y\}$$

On peut aussi définir l'ensemble atteignable sur un intervalle de temps $[\underline{t}, \overline{t}]$:

$$\mathcal{R}_{[\underline{t},\, \overline{t}]}(I) = \bigcup_{t \in [\underline{t},\, \overline{t}]} \Phi_t(I)$$

L'idéal serait de pouvoir calculer $\mathcal{R}_{[0,\, \infty[}(I)$, mais c'est un problème indécidable, on va donc calculer une sur-approximation, $\tilde{\mathcal{R}}$, de l'ensemble atteignable sur un intervalle de temps borné. On procède en discrétisant le système.

Une approche pour calculer une telle sur-approximation est de partir de $\tilde{\mathcal{R}}_{[0,\, t]}(I)$ pour calculer $\tilde{\mathcal{R}}_{[0,\, t+\Delta t]}(I)$. Pour cela il suffit de considérer le flot à la frontière de $\tilde{\mathcal{R}}_{[0,\, t]}(I)$ et de gonfler $\tilde{\mathcal{R}}_{[0,\, t]}(I)$ en conséquence [8]. L'intuition est qu'une trajectoire partant d'un point de $\tilde{\mathcal{R}}_{[0,\, t]}(I)$ reste dans $\tilde{\mathcal{R}}_{[0,\, t]}(I)$ pendant Δt ou en sort en passant par sa frontière. Il suffit donc, pour chaque face de la frontière de $\tilde{\mathcal{R}}_{[0,\, t]}(I)$ de considérer la projection maximale du flot sur la normale à cette face et de gonfler cette face en conséquence.

Une autre approche calcule $\tilde{\mathcal{R}}_{[0,\, t]}(I)$ comme l'union des $\tilde{\mathcal{R}}_{[i\Delta t,\, (i+1)\Delta t]}(I)$, chaque ensemble étant calculé à partir du précédent, c'est l'approche que nous utiliserons pour traiter les systèmes linéaires avec incertitude.

1.2.2 Dynamique discrète

Pour la dynamique discrète, on étend trivialement le cas du point (pour une trajectoire) à un ensemble en considérant l'intersection de $\mathcal{R}_{[0,\infty[}(I)$ (ou plutot de $\tilde{\mathcal{R}}_{[0,t]}(I)$) avec les gardes, cette intersection servira d'ensemble initial pour l'étude de la dynamique continue dans le nouvel état.

1.3 Systèmes affines par morceaux

Parmis les systèmes hybrides, la classe des systèmes affines par morceaux fait l'objet d'une attention particulière. Elle est en effet suffisamment simple pour pouvoir espérer en faire une étude automatique, et suffisamment expressive pour pouvoir approximer de façon satisfaisante de nombreux systèmes.

Définition 2 *Un système affine par morceaux, est un système modélisé par un automate hybride pour lequel, dans chaque état discret q, la dynamique continue est régie par une équation du type :*

$$\dot{x} = Ax + Bu$$

où A est une matrice $n \times n$ et B une matrice $n \times p$.

1.3.1 Vers un système discret

Avec cette dynamique, un point y est atteignable à partir du point x_0, si il existe un temps t et une instance des variables d'entrée $u : \mathbb{R}^+ \to U$ tels que :

$$y = e^{tA}x_0 + \int_0^t e^{(t-\tau)A}Bu(\tau)\mathrm{d}\tau$$

Si on fait l'hypothèse que pour tout t, $\|Bu(t)\| < \mu$ (cette hypothèse parait raisonable, peu de systèmes acceptent des entrées non bornées) alors :

$$\|y - e^{tA}x_0\| \leq \int_0^t e^{(t-\tau)\|A\|}\mu\mathrm{d}\tau = \frac{e^{t\|A\|} - 1}{\|A\|}\mu$$

Notons $\beta_t = \frac{e^{t\|A\|}-1}{\|A\|}\mu$, d'après l'équation précédente, il est clair que :

$$\Phi_t(I) \subseteq e^{tA}I \oplus \square\beta_t$$

où $e^{tA}I = \{e^{tA}x \mid x \in I\}$, $\oplus$ est la somme de Minkowski (pour deux ensembles A et B, $A \oplus B = \{a + b \mid a \in A \wedge b \in B\}$), et $\square\beta_t$ est la boule de centre 0 et de rayon β_t.

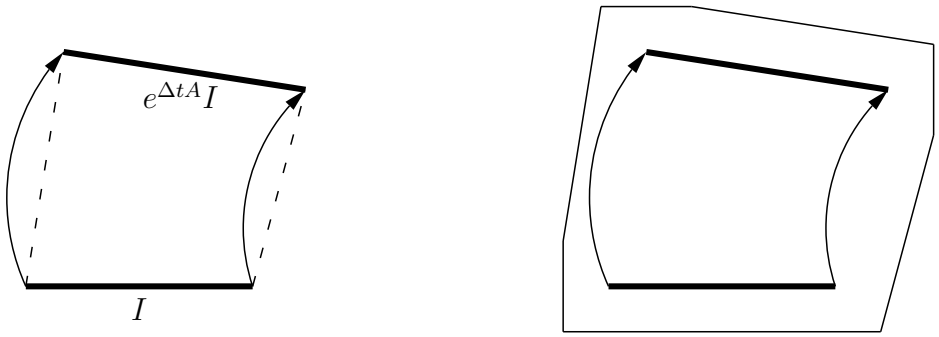


FIG. 1.2 – Principe de la sur-approximation de $\mathcal{R}_{[0,\,\Delta t]}(I)$

On peut donc calculer une sur-approximation de $\Phi_t(I)$, et si P est une sur-approximation de $\mathcal{R}_{[t,\,t+\Delta t]}(I)$, alors $\Phi_{\Delta t}(P)$ est une sur-approximation de $\mathcal{R}_{[t+\Delta t,\,t+2\Delta t]}(I)$.

Pour pouvoir calculer $\tilde{\mathcal{R}}_{[0,\,t]}(I)$, il nous reste à trouver un moyen de sur-approximer $\mathcal{R}_{[0,\,\Delta t]}(I)$ efficacement.

Pour cela, on peut prendre une sur-approximation de l'enveloppe convexe de I et $\Phi_{\Delta t}(I)$, et y rajouter un terme correcteur pour être sûr de ne rater aucun point de $\mathcal{R}_{[0,\,\Delta t]}(I)$.

Ce terme correcteur peut être pris égal à une boule de rayon $(e^{r\|A\|} - 1 - r\|A\|) \sup_{x\in I} \|x\|$ [11].

Ainsi, en trouvant une sur-approximation de $\mathcal{R}_{[0,\,\Delta t]}(I)$, et du flot, nous pouvons calculer une sur-approximation de $\mathcal{R}_{[0,\,T]}(I)$, en calculant (ou en sur-approximant) les $\frac{T}{\Delta t}$ premiers termes de la suite d'ensembles, $\Omega_0 = \tilde{\mathcal{R}}_{[0,\,\Delta t]}(I)$ et

$$\Omega_{i+1} = e^{\Delta t A}\Omega_i \oplus \square\beta_{\Delta t} \tag{1.2}$$

Si les termes de la suite $(\Omega_i)_{i\in\mathbb{N}}$ sont calculés de manière exacte, et si $U = \{u \mid \|Bu\| < \mu\}$, alors quand le pas de temps Δt tend vers 0, la distance entre $\mathcal{R}_{[0,\,T]}(I)$ et sa sur-approximation tend vers 0 [11].

Une première contribution

La première contribution de ce rapport permet l'amélioration de la sur-approximation du flux Φ_t.

Théorème 1 *Soit un système affine $\dot{x} = Ax + Bu$ avec $u : \mathbb{R}^+ \to U$, si*

$U = c + Z\square_1 = \{c + Zv \mid \|v\| \leq 1\}$[1], *alors, pour tout* $k \in \mathbb{N}$ *et pour tout* I *:*

$$\begin{array}{rcl} \Phi_t(I) \subset e^{tA}I & \oplus & \left\{\int_0^t e^{(t-\tau)A}Bc\mathrm{d}\tau\right\} \\ & \oplus & \bigoplus_{i=0}^{k-1} \frac{t^{i+1}}{(i+1)!}A^iBZ\square_1 \\ & \oplus & \square\left(\left(\int_0^t e^{(t-\tau)|A|}\mathrm{d}\tau - \sum_{i=0}^{k-1}\frac{t^{i+1}}{(i+1)!}|A|^i\right)|BZ|\right) \end{array}$$

où pour toute matrice $M = (m_{ij})$, $|M|$ *désigne* $(|m_{i,j}|)$*,et* $\square(M)$ *est l'image de la boule de rayon* 1, $\square_1$*, par la matrice diagonale engendrée par le vecteur :*

$$|M|\begin{pmatrix} 1 \\ \vdots \\ 1 \end{pmatrix}$$

En fait, ce théorème nous dit principalement qu'on peut remplacer $\|A\|$ par $|A|$ dans la méthode précédente.

Preuve. On peut remplacer le système $\dot{x} = Ax + Bu$ par :

$$\dot{x} = Ax + Bc + BZv$$

avec $v \in \square_1$. On a alors, y appartient a $\Phi_t(I)$ si et seulement si il existe x_0 dans I et $v : \mathbb{R}^+ \to \square_1$ tels que :

$$y = e^{tA}x_0 + \int_0^t e^{(t-\tau)A}Bc\mathrm{d}\tau + \int_0^t e^{(t-\tau)A}BZv(\tau)\mathrm{d}\tau$$

Ainsi :

$$\Phi_t(I) = e^{tA}I \oplus \{\int_0^t e^{(t-\tau)A}Bc\mathrm{d}\tau\} \oplus \{\int_0^t e^{(t-\tau)A}BZv(\tau)\mathrm{d}\tau \mid v : \mathbb{R}^+ \to \square_1\}$$

Nous allons maitenant sur-approximer $\{\int_0^t e^{(t-\tau)A}BZv(\tau)\mathrm{d}\tau \mid v : \mathbb{R}^+ \to \square_1\}$ en utilisant le développement en série entière de l'exponentielle :

$$\begin{array}{rcl} & \left\{\int_0^t e^{(t-\tau)A}BZv(\tau)\mathrm{d}\tau \mid v : \mathbb{R}^+ \to \square_1\right\} \\ = & \left\{\int_0^t \sum_{i=0}^{\infty}\frac{(t-\tau)^i}{i!}A^iBZv(\tau)\mathrm{d}\tau \mid v : \mathbb{R}^+ \to \square_1\right\} \\ = & \left\{\sum_{i=0}^{\infty}A^iBZ\int_0^t\frac{(t-\tau)^i}{i!}v(\tau)\mathrm{d}\tau \mid v : \mathbb{R}^+ \to \square_1\right\} \\ \subset & \bigoplus_{i=0}^{k-1}\left\{A^iBZ\int_0^t\frac{(t-\tau)^i}{i!}v(\tau)\mathrm{d}\tau \mid v : \mathbb{R}^+ \to \square_1\right\} \\ & \oplus\left\{\sum_{i=k}^{\infty}A^iBZ\int_0^t\frac{(t-\tau)^i}{i!}v(\tau)\mathrm{d}\tau \mid v : \mathbb{R}^+ \to \square_1\right\} \end{array}$$

[1] U est l'image du cube unitaire par une application linéaire Z translatée en c, nous verrons plus loin qu'il s'agit d'un zonotope

On a $\|v(\tau)\| \leq 1$ donc :

$$\left\| \int_0^t \frac{(t-\tau)^i}{i!} v(\tau) \mathrm{d}\tau \right\| \leq \frac{t^{i+1}}{(i+1)!}$$

et :

$$\left\{ A^i BZ \int_0^t \frac{(t-\tau)^i}{i!} v(\tau) \mathrm{d}\tau \mid v : \mathbb{R}^+ \to \square_1 \right\} \subset \frac{t^{i+1}}{(i+1)!} A^i BZ \square_1$$

Il nous reste maintenant à sur-approximer $\left\{ \sum_{i=k}^{\infty} A^i BZ \int_0^t \frac{(t-\tau)^i}{i!} v(\tau) \mathrm{d}\tau \right\}$, nous allons sur-approximer cet ensemble par un hyper-rectangle. Nous allons noter $(v)_j$ la j-ième composante de v, et $(M)_{jl}$ le coefficient à la j-ième ligne, l-ième colonne de M.

Soit $v : \mathbb{R}^+ \to \square_1$, on a :

$$\left| \left(\sum_{i=k}^{\infty} A^i BZ \int_0^t \frac{(t-\tau)^i}{i!} v(\tau) \mathrm{d}\tau \right)_j \right| \leq \sum_{i=k}^{\infty} \left| \left(A^i BZ \int_0^t \frac{(t-\tau)^i}{i!} v(\tau) \mathrm{d}\tau \right)_j \right|$$

Majorons $\left| \left(A^i BZ \int_0^t \frac{(t-\tau)^i}{i!} v(\tau) \mathrm{d}\tau \right)_j \right|$

$$\begin{aligned} \left| \left(A^i BZ \int_0^t \frac{(t-\tau)^i}{i!} v(\tau) \mathrm{d}\tau \right)_j \right| &\leq \left| \sum_{l=1}^d \left(A^i BZ \right)_{jl} \left(\int_0^t \frac{(t-\tau)^i}{i!} v(\tau) \mathrm{d}\tau \right)_l \right| \\ &\leq \sum_{l=1}^d \left| \left(A^i BZ \right)_{jl} \right| \left| \left(\int_0^t \frac{(t-\tau)^i}{i!} v(\tau) \mathrm{d}\tau \right)_l \right| \end{aligned}$$

Or, pour toutes matrices M et N, on a :

$$\left(|MN| \right)_{ij} = \left| \sum_{l=1}^d (M)_{il} (N)_{lj} \right| \leq \sum_{l=1}^d (|M|)_{il} (|N|)_{lj} = \left(|M||N| \right)_{ij}$$

de plus, comme on l'a déjà vu, $\left| \left(\int_0^t \frac{(t-\tau)^i}{i!} v(\tau) \mathrm{d}\tau \right)_l \right| \leq \frac{t^{i+1}}{(i+1)!}$ donc :

$$\left| \left(A^i BZ \int_0^t \frac{(t-\tau)^i}{i!} v(\tau) \mathrm{d}\tau \right)_j \right| \leq \sum_{l=1}^d \frac{t^{i+1}}{(i+1)!} \left(|A|^i |BZ| \right)_{jl}$$

On a donc bien :

$$\begin{gathered} \left\{ \sum_{i=k}^{\infty} A^i BZ \int_0^t \frac{(t-\tau)^i}{i!} v(\tau) \mathrm{d}\tau \mid v : \mathbb{R}^+ \to \square_1 \right\} \\ \subset \\ \square \left(\left(\int_0^t e^{(t-\tau)|A|} \mathrm{d}\tau - \sum_{i=0}^{k-1} \frac{t^{i+1}}{(i+1)!} |A|^i \right) |BZ| \right) \end{gathered}$$

□

Ce résultat n'améliore pas la convergence théorique de la méthode, en effet, dans le pire des cas, si A est la matrice identité mutlitpliée par un scalaire, on obtient la même sur-approximation pour le flux. Mais en pratique il permet d'obtenir une sur-approximation plus fine, notemment quand l'incertitude n'est importante que dans quelques directions.

On peut utiliser la même idée pour améliorer la sur-approximation de $\mathcal{R}_{[0,\,\Delta t]}(I)$. Par ailleurs, pour améliorer cette approximation, une autre solution consiste à calculer $\mathcal{R}_{[0,\,\Delta t]}(I)$ à partir de $\mathcal{R}_{[0,\,\frac{\Delta t}{n}]}(I)$ avec un plus petit pas de temp, ceci peut s'avérer utile quand la norme de A est importante, et que la norme sur l'incertitude, μ, est suffisament petite pour nous permettre d'utiliser un pas de temps relativement grand.

1.3.2 Approximation des dynamiques non-linéaires

Un des intérêts des systèmes affines par morceaux, est de pouvoir être utilisés pour l'approximation de dynamiques non-linéaires. L'idée de la méthode [1, 9, 10] est de choisir une partition du domaine de définition du système. Sur chaque élément de la partition, le champ de vecteur initial est approximé par une dynamique affine avec incertitude. L'incertitude permettant de prendre en compte les erreurs dues à l'approximation d'une dynamique non linéaire par une dynamique linéaire.

Des résultats de convergence sont montrés dans [10]. Nous n'entrerons pas plus dans les détails de la méthode. Il s'agit surtout ici de mettre en évidence l'importance de l'étude des systèmes affines avec incertitudes.

Chapitre 2

Atteignabilité des systèmes affines en temps discret

À partir de maintenant, nous allons nous placer en temps discret et considérer des systèmes linéaires avec incertitudes.

Nous nous intéressons donc à la suite d'ensembles $(\Omega_k)_{k\in\mathbb{N}}$ définie par la relation de récurrence (1.2), pour plus de clareté, nous prendrons les notations suivantes :

$$\Omega_i = A\Omega_{i-1} \oplus U \tag{2.1}$$

2.1 Algorithme usuel

L'objectif ici est de *calculer* les k premiers termes de cette suite, nous devons donc avoir une représentation finie pour les ensembles Ω_i sur laquelle nous pouvons faire des calculs, par exemples des polytopes convexes représentés par une intersection d'un nombre fini de demi-espaces, ou par la liste de leurs sommets. Les algorithmes que nous présentons dans cette section ne sont pas spécifiques à une représentation particulière pour les ensembles.

L'algorithme 1 montre comment calculer de façon exacte[1] les premiers termes de cette suite.

Malheureusement cette algorithme est inutilisable en pratique, il a une complexité désastreuse, la somme de Minkowski fait grandir la taille de la

[1] Il est clair que, puisque nous utilisons une représentation finie (et suffisament simple pour pouvoir faire des calculs), nous ne pouvons calculer les premiers termes de cette suite que si Ω_0 est représentable dans notre structure de donnée. Il est aussi clair que cet algorithme n'est *exact* qu'aux erreurs d'arrondis près (si notre représentation utilise des nombres flottants par exemple).

Algorithme 1 Algorithme usuel : Version Exacte

Entrée : une matrice A, deux ensembles Ω_0 et U, et un entier k.
Sortie : Les k premiers termes de la suite définie par $\Omega_i = A\Omega_{i-1} \oplus U$.

```
1: RS ← Ω_0
2: for i from 1 to k do
3:     Ω_i ← AΩ_{i−1} ⊕ U
4:     RS ← RS ∪ (Ω_i)
5: end for
6: return RS
```

représentation des ensembles Ω_i, et on se retrouve à appliquer une transformation linéaire sur des objets de plus en plus gros[2]. On peut le voir par exemple sur la figure 1.2, la sur-approximation de $\mathcal{R}_{[0,\,\Delta t]}(I)$ est la somme de deux quadrilatères (l'enveloppe convexe de I et $e^{\Delta t A}I$, et un terme correcteur carré) et elle a 7 sommets.

Prenons par exemple le cas où les ensembles Ω_0 et U sont des polygones convexes représentés par leurs N sommets. Comme il est montré en [12], la somme de j polygones convexes ayant au plus N sommets en dimension d a, dans le pire des cas, un nombre de sommets de l'ordre de $\mathcal{O}(j^{d-1}N^{2(d-1)})$. Donc, si N_i est le nombre de sommets de Ω_i, dans le pire des cas :

$$N_i = \mathcal{O}\left(i^{d-1}N^{2(d-1)}\right)$$

La taille de la sortie est donc de l'ordre de $\mathcal{O}\left(dN^{2(d-1)}k^{d-1}\right)$ dans le pire des cas. Cet algorithme *exact* semble donc difficilement utilisable. La solution classiquement proposée est de faire des sur-approximations à chaque étape, pour limiter la taille de la représentation des Ω_i, comme décrit dans l'algorithme 2.

$APPROX$ est un algorithme qui prend en entrée un ensemble et en renvoie une sur-approximation dont la taille de la représentation est plus petite.

À chaque étape, on sur-approxime $A\tilde{\Omega}_{i-1} \oplus U$ pour garder la taille de la représentation de $\tilde{\Omega}_i$ bornée, il existe quelques variantes, nottemment l'approximation peut être effectuée après l'ajout de $\tilde{\Omega}_i$ à RS.

Même avec cette algorithme, l'utilisation des polytopes convexes reste complexe, c'est pourquoi, d'autres classes d'ensembles ont été étudiées.

[2]Nous supposons que nous utilisons une représentation raisonable pour les ensembles, on pourrait représenter Ω_k par $\langle A, \Omega_0, U, k\rangle$ pour éviter cette explosion, mais cela n'aurait pas vraiment de sens. Et à notre connaissance, pour toute classe d'ensembles close par transformation linéaire et somme de Minkowski, la somme de Minkowski augmente la taille de la représentation

Algorithme 2 Algorithme usuel : Version Approchée

Entrée : une matrice A, deux ensembles Ω_0 et U, et un entier k.

Sortie : Une sur-approximation des k premiers termes de la suite définie par $\Omega_i = A\Omega_{i-1} \oplus U$.

1: $RS \leftarrow \Omega_0$
2: **for** i from 1 to k **do**
3: $\quad \tilde{\Omega}_i \leftarrow$ APPROX$(A\tilde{\Omega}_{i-1} \oplus U)$
4: $\quad RS \leftarrow RS \cup (\tilde{\Omega}_i)$
5: **end for**
6: **return** RS

2.2 Différentes structures de données

Quelques classes d'ensembles sont ici présentées très succintement, nous nous intéresserons ensuite plus en détail aux zonotopes, que nous utiliserons par la suite.

2.2.1 Rectangles (Arithmétique d'intervalles)

On ne considère ici que les ensembles de la forme $[a_1;\ b_1] \times \ldots \times [a_d;\ b_d]$, c'est à dire les rectangles dont les arrêtes sont parallèles aux axes du système de coordonnées. Il s'agit en fait de l'ensemble des boules pour la norme infinie.

L'avantage des boules, est que la somme de deux boules est une boule, la collection des boules est close par somme de Minkowski.

$$\begin{aligned}&([a_1;\ b_1] \times \ldots \times [a_d;\ b_d]) \oplus ([a'_1;\ b'_1] \times \ldots \times [a'_d;\ b'_d])\\ &= [a_1 + a'_1;\ b_1 + b'_1] \times \ldots \times [a_d + a'_d;\ b_d + b'_d]\end{aligned}$$

Une boule a une représentation compacte, $2d$ réels, et il est très facile[3], à partir d'un ensemble de points A, de trouver la plus petite boule $BOX(A)$ contenant cet ensemble. Par exemple, si A est un polytope convexe représenté par l'ensemble S de ses sommets :

$$BOX(A) = [\min_{x\in S} \pi_1(x);\ \max_{x\in S} \pi_1(x)] \times \ldots \times [\min_{x\in S} \pi_d(x);\ \max_{x\in S} \pi_d(x)]$$

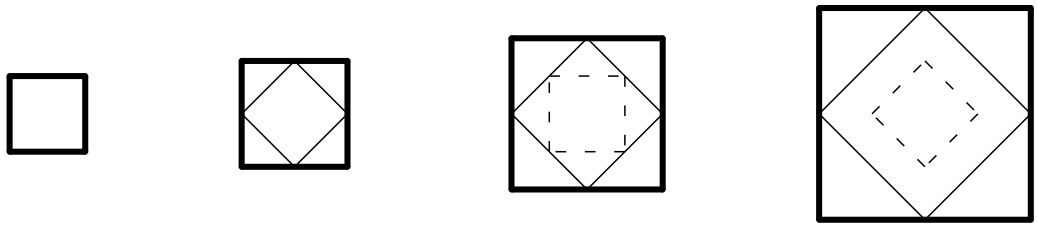
où $\pi_i(x)$ est la i-ième composante de x.

De plus la fonction BOX à une propriété intéressante :

Proposition 1 *Pour tout ensembles A et B, on a :*

$$BOX(A) \oplus BOX(B) = BOX(A \oplus B)$$

[3]encore une fois, en supposant que A à une représentation raisonnable



En trait épais : $\tilde{\Omega}_i$, l'ensemble calculé
En trait fin : $A\tilde{\Omega}_{i-1}$, l'image par A de l'ensemble calculé avant approximation
En trait pointillé : Ω_i, l'ensemble exacte

FIG. 2.1 – Sur-approximations successives pour la rotation

Preuve. Intuitivement, pour chaque face F, normale aux vecteur e_i du système de coordonnées, de $BOX(A)$ il existe un vecteur u_i de A dans F, de même, on peut trouver un v_i dans B, et $u_i + v_i$ atteind une face de $BOX(A) \oplus BOX(B)$ et une face de $BOX(A \oplus B)$ □

Malheureusement, l'image d'une boule par une application linéaire n'est pas nécessairement une boule, et même si il est facile de trouver la plus petite boule la contenant, ces sur-approximations successives peuvent produire une erreur importante.

Prenons l'exemple de la rotation de 45 degrée dans le plan, sans incertitude ($U = 0$) le rayon de Ω reste constant, pourtant, à cause des sur approximations successives, le rayon de $\tilde{\Omega}_i$ est $\sqrt{2}^i$ fois celui de Ω_0. La figure 2.1 illustre cet exemple.

2.2.2 Rectangles orientés

Pour tenter de résoudre ce problème, on considère un système de coordonnées variable [19], ce qui revient à dire que les rectangles considérés ne sont plus orientés suivant les axes du système de coordonnées initial, mais de façon à ce que l'approximation soit la meilleure possible. L'exemple de la rotation sans incertitude ne pose alors plus de problème.

Par contre, le choix de l'orientation du rectangle approximant n'est pas trivial, d'autant moins que l'ensemble des rectangles orientés n'est clos ni par transformation linéaire, ni par somme de Minkowski.

La méthode qui semble la plus efficace pour choisir l'orientation des rectangles, que nous ne détaillerons pas ici, est due à Lohner [18]. Elle est basée sur la décomposition QR des matrices, en un produit d'une matrice orthogonale Q et d'une matrice triangulaire supérieure R.

2.2.3 Ellipsoïdes

Les rectangles orientés permettent d'obtenir une précision supérieure à celle obtenue avec une arithmétique d'intervalles, mais on perd la cloture par somme de Minkowski, et on est toujours pas clos par transformation linéaire, on peut aussi considérer que les rectangles sont des formes trop simples pour représenter des ensembles engendrés par une dynamique continue. C'est pourquoi l'utilisation d'ellispoïdes, ou de familles d'ellipsoïdes, à été proposée.

Elles ont l'avantage d'avoir une représentation aussi compacte que les rectangles orientés, une matrice $d \times d$ et un vecteur, et contrairement a ceux-ci, l'ensemble des ellipsoïdes est clos par transformation linéaire (mais toujours pas par somme de Minkowski).

Dans [16], Kurzhanski et Vályi décrivent comment effectuer efficacement les différentes opérations nécessaires à l'utilisation des ellipsoïdes pour l'étude des systèmes dynamiques. Malheureusement, pour ces opérations, entrent parfois en jeu des problèmes d'optimisation non-triviaux, ainsi que des approximations inévitables, la classe des ellipsoïdes n'étant pas close par somme de Minkowski.

2.2.4 Zonotopes

Les précédentes classes d'ensembles ne sont pas closes à la fois par transformation linéaire et par somme de Minkowski. Les approximations ne servent plus à limiter la taille de la représentation des ensembles, puisque tous les ensembles d'une même classe ont une représentation de même taille[4], mais à approximer ces opérations. L'ensemble des zonotopes, est le plus petit ensemble clos par transformation linéaire et somme de Minkowski[5], ces opérations ne posent plus de problèmes, mais comme pour les polytopes convexes, il faut une méthode pour controller la taille de leur représentation.

Un zonotope est la somme de Minkowski d'un ensemble de segments, il peut aussi être vu comme l'image par une application affine du cube unitaire en dimension supérieure.

Définition 3 *Un zonotope est représenté par son centre u, et ces générateurs $v_1, \ldots, v_m$.*

$$\langle u;\ v_1, \ldots, v_m \rangle = \{u + \sum_{j=0}^{m} \alpha_j v_j \mid \forall j,\ \alpha_j \in [-1;\ 1]\}$$

[4] $2d$ pour les rectangles, $d(d+1)$ pour les rectangles orientés et les ellipsoïdes

[5] plus précisément, le plus petit ensemble clos par transformation linéaire et somme de Minkowski contenant des ensembles connexes non réduits à un point

*Un zonotope de m générateurs en dimension d est d'*ordre $\frac{m}{d}$.

Avoir une représentation de taille variable (ici $(m+1)d$) permet de choisir entre précision et vitesse d'exécution en faisant varier la taille maximale autorisée.

Propriétés

Étant la somme d'un ensemble fini de segments, les zonotopes sont moins généraux que les polytopes convexes, ils ont en particulier une symmétrie centrale. On peut aussi remarquer qu'en dimension $d \geq 3$, les polytopes convexes à symmétrie centrale ne sont pas tous des zonotopes. Par contre, ils présentent plusieurs avantages, nottemment une représentation compacte par rapport au polytopes convexes, qui sont représentés par l'ensemble de leur sommets ou par l'intersection de demi-espaces :

Proposition 2 *[23] Le nombre de sommets d'un zonotope en dimension d avec m générateurs est borné par :*

$$2\sum_{j=0}^{d-1}\binom{m-1}{j}$$

Le nombre de faces de dimension $d-1$ d'un zonotope en dimension d avec m générateurs est borné par :

$$2\binom{m}{d-1}$$

Ces bornes sont atteintes dans les cas génériques.

Ainsi, on peut encoder un objet ayant $\mathcal{O}(m^{d-1})$ sommets avec md réels, si on le voit comme un polytope convexe, son encodage nécessite $\mathcal{O}(dm^{d-1})$ réels. Outre cette aspect *compact*, un autre intérêt des zonotopes est que la transformation linéaire et la somme de Minkowski sont relativement simples.

Proposition 3 *La transformation linéaire d'un zonotope d'ordre p ($m = dp$ générateurs) se fait en $\mathcal{O}(p\mathcal{M}(d))$ où $\mathcal{M}(d)$ est la complexité de la multiplication de deux matrices $d \times d$:*

$$A\langle u;\ v_1, \ldots, v_m\rangle = \langle Au;\ Av_1, \ldots, Av_m\rangle$$

La somme de Minkowski de deux zonotopes se fait en $\mathcal{O}(d)$:

$$\langle u;\ u_1, \ldots, u_n\rangle \oplus \langle v;\ v_1, \ldots, v_m\rangle = \langle u+v;\ u_1, \ldots, u_n, v_1, \ldots, v_m\rangle$$

Pour la somme de Minkowski, on atteint $\mathcal{O}(d)$ en utilisant des listes doublement chaînées pour la liste des générateurs, la concaténation se fait alors en $\mathcal{O}(1)$, seul l'addition des centres des zonotopes prend du temps. Mais si on sait que le centre d'un des deux zonotopes est l'origine ($u = (0, \ldots, 0)$), alors la somme de Minkowski se fait en $\mathcal{O}(1)$. C'est souvent le cas pour le zonotope U qui modélise l'incertitude du système.

Par ailleurs, si on limite l'ordre des zonotopes à $\mathcal{O}(1)$, et si on utilise une multiplication matricielle ayant une bonne complexité théorique [5], on peut atteindre pour la transformation linéaire une complexité de l'ordre de $\mathcal{O}(d^{2,376})$. Mais la taille des matrices considérées en pratique est insuffisante pour que cet algorithme soit vraiment efficace.

Algorithmes

Comme nous l'avons vu, la somme de Minkowski pour les zonotopes se fait en concaténant la liste des générateurs, la taille de la somme de deux zonotopes est donc égale à la somme des tailles de ces zonotopes. Ainsi, dans l'algorithme 1, si on note encore N_i la taille de la représentation de Ω_i, si Ω_0 est d'ordre p et U d'ordre q, alors :

$$N_i = (p + iq)d^2 + d$$

La taille de la sortie est donc $k(p + \frac{k+1}{2}q)d^2 + kd$. Si p et q sont en $\mathcal{O}(1)$, l'algorithme 1 pour les zonotopes a une complexité en temps en $\mathcal{O}(k^2\mathcal{M}(d))$ et en espace en $\mathcal{O}(k^2d^2)$.

k est un paramètre crucial, c'est le nombre de pas considérés. Pour pouvoir prendre un pas de temps petit, et donc éviter une approximation initiale trop grossière, il faut pouvoir travailler avec un k grand, c'est pourquoi une complexité en k^2 n'est pas satisfaisante.

Nous allons donc devoir effectuer des approximations pour limiter l'ordre, p, des zonotopes manipulés. Un des intérêts des zonotopes, est aussi ce paramètre p, qui permet de choisir la précision de la sortie : si on prend p supérieur à l'ordre maximal atteint par l'algorithme 1, alors on calcule avec l'algorithme 2 les Ω_i de façon exacte, alors que si on prend $p = 1$ on se rapproche des méthodes se basant sur des approximations rectangulaires.

Approximer un zonotope d'ordre q par un zonotope d'ordre p de façon optimale est un problème difficile. C'est pourquoi les méthodes d'approximation utilisées fonctionnent principalement sur le principe suivant : On extrait un certain nombre de générateurs du zonotope Z pour construire deux zonotopes, Z_1 et Z_2 tels que

$$Z = Z_1 \oplus Z_2$$

Et on sur-approxime Z_2 par un rectangle R, en projetant ses générateurs sur les axes du systèmes de coordonnées. On sur-approxime alors $Z_1 \oplus Z_2$ par $Z_1 \oplus R$.

Le problème est de choisir quels générateurs vont être sur-approximés par un rectangle.

Flush-When-Full

On peut choisir de sur-approximer par un rectangle le zonotope engendré par les plus petits générateurs du zonotope initial, l'idée étant que l'erreur d'approximation augmente avec la taille des générateurs. Le problème, selon Kühn [14], est qu'alors la taille des plus petits générateurs va grandir. Ainsi, au bout de quelques pas de temps, le zonotope $\tilde{\Omega}_i$ n'aura plus de *petits* générateurs, et on devra faire des approximations de plus en plus grossières.

Il faut donc maintenir un certain nombre de *petits* générateurs, quitte à faire parfois des approximations plus grossières que nécessaire.

Dans [14, 15], Kühn propose un algorithme pour atteindre ce but. Les générateurs sont regroupés par blocs de d. L'ordre dans lequel les générateurs d'un zonotope sont listés n'a pas d'importance, mais ici l'ordre des blocs est important, ils sont initialement triés par norme croissante, et l'algorithme utilisé va maintenir cet ordre.

Un zonotope d'ordre p est représenté par :

$$\langle u;\, V_1, \ldots, V_p \rangle$$

où $V_j = v_{dj+1} \ldots v_{dj+d}$.

L'idée est alors à chaque étape de choisir les plus petits blocs pour faire l'approximation, et quand il n'y a plus de *petits* blocs, on approxime plusieurs blocs par un seul pour faire apparaître des blocs nuls.

Plus formellement, il propose de choisir le plus grand ℓ tel que :

$$\|[U, AV_1, \ldots, AV_{\ell-1}]\| > \|AV_\ell\|$$

(où $[U, AV_1, \ldots, AV_{\ell-1}]$ est une matrice par blocs) et de remplacer les blocs $V_1, \ldots, V_{\ell-1}$ par $0, \ldots, 0$ et le bloc V_ℓ par le plus petit rectangle approximant $[U, AV_1, \ldots, AV_\ell]$.

Example 2 *La table 2.1 montre les 12 premières itérations de l'algorithme FWF pour la récurence $\Omega_i = \Omega_{i-1} \oplus I$. On ne fait ici aucune sur approximation, puisque la suite ainsi définie ne fait que sommer des rectangles, mais cette exemple permet d'avoir l'intuition de ce qui se passerait sur un problème plus compliqué, on ne fait des sur-approximations* importante *qu'aux étapes* 4 *et* 9*, et on aura toujours des* petits *blocs à sur-approximer, comme à l'étape* 7*, toutes les autres étapes sont faites sans sur-approximation.*

n	Z	n	Z	n	Z
0	$[0,0,0]$	4	$[0,0,4I]$	8	$[I,3I,4I]$
1	$[0,0,I]$	5	$[0,I,4I]$	9	$[0,0,9I]$
2	$[0,I,I]$	6	$[I,I,4I]$	10	$[0,I,9I]$
3	$[I,I,I]$	7	$[0,3I,4I]$	11	$[I,I,9I]$

TAB. 2.1 – FWF pour $\Omega_i = \Omega_{i-1} \oplus I$

Plus petits générateurs

Dans [11], Antoine Girard, contrairement à ce que préconise Kühn[6], propose de prendre les $(q-p+1)d$ plus petits générateurs pour l'ordre :

$$u < v \Leftrightarrow \|u\|_1 - \|u\|_\infty < \|v\|_1 - \|v\|_\infty$$

L'idée est que plus $\|u\|_1 - \|u\|_\infty$ est petit, plus u se trouve *proche* d'un axe du système de coordonnées, et donc l'approximation sera moins grossière que pour un vecteur plus *éloigné*.

Cette méthode semble plus précise en pratique que celle de Kühn, mais elle nécessite un tri des vecteurs à chaque étape. Une étude plus approfondie reste à faire pour déterminer son efficacité.

2.2.5 Récapitulatif

La table 2.2 récapitule les différentes propriétés de clôtures pour les classes d'ensembles que nous venons de présenter. On a ajouté aussi dans ce tableau la clôture par union, intersection, et intersection avec un demi-espace, qui sont des propriétés importantes quand on voudra passer d'un système affine à un système affine par morceaux.

Seul les polytopes convexes et les zonotopes peuvent être utilisés pour un algorithme *exact*.

2.3 L'effet d'emballage

Toutes les méthodes que nous venons de voir nécessitent, à chaque étape, de sur-approximer $A\tilde{\Omega}_{i-1} \oplus U$ pour obtenir $\tilde{\Omega}_i$. Ces sur-approximations successives s'accumulent et sont propagées par la transformation linéaire A. C'est ce qu'on appelle l'effet d'emballage.

[6] Quand Kühn met en garde contre le choix systématique des plus petits vecteurs pour la sur-approximation, *petit* signifie petit pour une certaine norme, il n'est donc pas clair que l'ordre choisi par Girard provoque les même problèmes, car $x \mapsto \|x\|_1 - \|x\|_\infty$ n'est pas une norme.

	A	$\oplus$	$\cup$	$\cap$	$\cap H$
rectangles		$\surd$		$\surd$	
rectangles orientés					
ellipsoïdes	$\surd$				
zonotopes	$\surd$	$\surd$			
polytopes convexes	$\surd$	$\surd$		$\surd$	$\surd$

TAB. 2.2 – Clôture des différentes classes d'ensembles par les opérateurs de transformation linéaire, somme de Minkowski, union, intersection, intersection avec un demi-espace.

On a vu en 2.2.1 que cet effet peut engendrer, sur des problèmes simple, une erreur exponentielle en k. Dans [14], Kühn évalue la performance de FWF pour un problème légèrement différent :

Théorème 2 *[14]*
Soit $(\Omega_i)_{i\in\mathbb{N}}$ la suite définie par :

$$\begin{cases} \Omega_0 &= 0 \\ \Omega_i &= A_i\Omega_{i-1} \oplus U_i \end{cases}$$

Soit $(\tilde{\Omega}_i)_{i\in\mathbb{N}}$ l'approximation de $(\Omega_i)_{i\in\mathbb{N}}$ calculée par l'algorithme FWF avec des zonotopes d'ordre p.

Si $\|U_i\| < \delta$ et si $\|A_{i+j}A_{i+j-1}\dots A_i\| \leq M$ pour tout i, j et où M et δ sont deux constantes. Alors il existe des constantes c_1 et c_2 (ne dépendant que de p) tels que :

$$dist(\tilde{\Omega}_i, \Omega_i) \leq \delta c_1 c_2^{i^{1/p}}$$

L'algorithme FWF est jugé performant, puisqu'il produit un effet d'emballage sous-exponentiel, mais celui ci reste tout de même important.

Chapitre 3

Suppression de l'effet d'emballage

Les principales contributions de ce rapport sont dans ce chapitre. Rappelons la relation de récurence qui nous intéresse :

$$\Omega_i = A\Omega_{i-1} \oplus U$$

3.1 Séparation de la somme et de l'application linéaire

Nous avons vu en 2.3 que les erreurs d'approximations, propagées et accumulées par le flot, pouvaient engendrer une erreur importante au bout de quelques pas de temps.

Ces approximations sont nécessaires quand la famille d'ensemble considerée n'est pas close par application affine ou somme de Minkowski (comme pour les rectangles), ou quand la somme de Minkowski rend l'ensemble consideré trop compliqué pour pouvoir lui appliquer la transformation linéaire A.

Nous allons montrer ici comment séparer la somme et la transformation linéaire, pour ainsi supprimer l'effet d'emballage.

Remarquons tout d'abord qu'une représentation non récursive de la suite défninie par 2.1 est :

$$\Omega_i = A^i\Omega_0 \oplus \bigoplus_{j=0}^{i-1} A^jU$$

Ainsi pour calculer Ω_{i+1} l'algorithme 1 calculerait :

$$A\Omega_i \oplus U$$

où Ω_i est l'ensemble :

$$(A^i\Omega_0 \oplus A^{i-1}U \oplus \ldots \oplus AU \oplus U)$$

dont la taille de la représentation peut être très grande.

Or, on peut remarquer que :

$$A(A^i\Omega_0 \oplus A^{i-1}U \oplus \ldots \oplus AU \oplus U) \oplus U$$
$$=$$
$$A(A^i\Omega_0) \oplus (A(A^{i-1}U) \oplus (A^{i-1}U \oplus \ldots \oplus AU \oplus U))$$

Ainsi, si pour Ω_i on a calculé $A^i\Omega_0$, $A^{i-1}U$, et $\bigoplus_{j=0}^{i-1} A^jU$, on calcule Ω_{i+1} avec deux transformations linéaires sur des objets de la taille des entrées et deux sommes de Minkowski sur des objets dont la taille est croissante.

Plus précisement, on calcule les trois suites définies par :

$$\begin{aligned} X_{i+1} &= AX_i & X_0 &= \Omega_0 \\ V_{i+1} &= AV_i & V_0 &= U \\ S_{i+1} &= V_i \oplus S_i & S_0 &= \{0\} \end{aligned}$$

On a alors $\Omega_i = X_i \oplus S_i$.

En effet, pour tout i, on a :

$$\begin{aligned} X_i &= A^iX_0 = A^i\Omega_0 \\ V_i &= A^iV_0 = A^iU \\ S_i &= \bigoplus_{j=0}^{i-1} V_j = \bigoplus_{j=0}^{i-1} A^jU \end{aligned}$$

d'où $\Omega_i = X_i \oplus S_i$.

On en déduit l'algorithme 3.

La seule opération se faisant ici sur des objets de taille croissante est la somme de Minkowski, or on a vu en 2.2.4 que pour les zonotopes, la complexité de la somme de Minkowski ne dépend pas de la taille des opérandes. Il est donc intéressant d'utiliser cet algorithme avec des zonotopes. Cela permet aussi de reduire la complexité en espace, en effet, pour stocker Z_1 et $Z_1 \oplus Z_2$ il suffit de stocker Z_1 et Z_2 (puisque la somme de Minkowski $Z_1 \oplus Z_2$ n'est que la concaténation des vecteurs de Z_1 et Z_2).

Ainsi, pour stocker les $\bigoplus_{j=0}^{i-1} A^jU$ pour i entre 0 et k, il suffit de stocker les A^iU pour i entre 0 et k.

On a donc le théorème suivant :

Théorème 3 *L'algorithme 3 est correct et a une complexité en temps en* $\mathcal{O}(k\mathcal{M}(d))$ *et en espace en* $\mathcal{O}(kd^2)$*, pour des zonotopes d'entrée d'ordre* $\mathcal{O}(1)$

Algorithme 3 Exacte

Entrée : une matrice A, deux ensemble Ω_0 et U, et un entier k.
Sortie : les k premiers termes de la suite définie par $\Omega_{i+1} = A\Omega_i \oplus U$.

1: $X_0 \leftarrow \Omega_0$
2: $V_0 \leftarrow U$
3: $S_0 \leftarrow \{0\}$
4: $RS \leftarrow \Omega_0$
5: **for** i from 1 to k **do**
6: $\quad X_i \leftarrow AX_{i-1}$ ▷ $X_i = A^i\Omega_0$
7: $\quad S_i \leftarrow S_{i-1} \oplus V_{i-1}$ ▷ $S_i = \bigoplus_{j=0}^{i-1} A^jU$
8: $\quad RS \leftarrow RS \cup (X_i \oplus S_i)$ ▷ $RS = \bigcup_{j=0}^{i} \Omega_j$
9: $\quad V_i \leftarrow AV_{i-1}$ ▷ $V_i = A^iU$
10: **end for**
11: **return** RS

($\mathcal{O}(d)$ générateurs). Où d est la dimension de l'espace, et $\mathcal{M}(d)$ la complexité de la multiplication de deux matrices $d \times d$.

L'algorithme implémenté utilise une multiplication matricielle naïve, la complexité de l'algorithme est donc en $\mathcal{O}(kd^3)$, mais il existe des algorithmes de multiplication matricielle [5] permettant d'atteindre pour l'algorithme 3 une complexité asymptotique de l'ordre de $\mathcal{O}(kd^{2,376})$.

On a trouvé un algorithme très rapide et peu gourmand en mémoire pour le calcul *exact* des k premiers termes de la suite définie par l'équation 2.1. Malheureusement les derniers termes de la suite sont des zonotopes d'ordre $\mathcal{O}(k)$ ($\mathcal{O}(kd)$ générateurs), la sortie de cette algorithme devient donc vite difficilement exploitable, nous allons donc en proposer une variante, permettant de limiter le nombre de générateurs des zonotopes constituant la sortie, tout en garantissant une erreur d'approximation raisonnable.

3.2 Réduction du nombre de générateurs

Pour réduire le nombre de générateurs, nous sommes obligés d'utiliser un algorithme d'approximation. Pour éviter un effet similaire à l'effet d'emballage, on doit éviter de faire des sur-approximations de sur-approximations.

On peut par exemple utiliser un algorithme d'approximation vérifiant la propriété suivante : pour tout ensembles A et B, on doit avoir

$$APPROX(A) \oplus APPROX(B) = APPROX(A \oplus B)$$

Remarque 1 *Il est clair que pour toute méthode d'approximation la somme des approximations de A et B est une approximation de leur somme. Mais ici $APPROX$ est un algorithme, et on veut que la somme des approximation de A et B soit exactement l'ensemble donné par l'algorithme $APPROX$ si on l'avait exécuté sur $A \oplus B$.*

L'approximation par un rectangle orthogonal aux axes du système, BOX vérifie cette propriété. En fait, l'approximation par n'importe quel zonotope non dégénéré d'ordre 1 vérifie cette propriété (si on approxime toujours par le *même* zonotope).

Nous pouvons utiliser cette idée pour obtenir l'algorithme 4.

Algorithme 4 Approximation Rectangulaire

Entrée : une matrice A, deux ensemble Ω_0 et U, et un entier k.
Sortie : l'approximation optimale par des rectangles orthogonaux aux axes des k premiers termes de la suite définie par $\Omega_{i+1} = A\Omega_i \oplus U$.

1: $X_0 \leftarrow \Omega_0$
2: $V_0 \leftarrow U$
3: $S_0 \leftarrow \{0\}$
4: $RS \leftarrow \text{Box}(\Omega_0)$
5: **for** i from 1 to k **do**
6: $\quad X_i \leftarrow AX_{i-1}$ $\quad \triangleright\ X_i = A^i\Omega_0$
7: $\quad S_i \leftarrow S_{i-1} \oplus \text{Box}(V_{i-1})$ $\quad \triangleright\ S_i = \text{Box}(\bigoplus_{j=0}^{i-1} A^jU)$
8: $\quad RS \leftarrow RS \cup (\text{Box}(X_i) \oplus S_i)$ $\quad \triangleright\ RS = \bigcup_{j=0}^{i} \text{Box}(\Omega_j)$
9: $\quad V_i \leftarrow AV_{i-1}$ $\quad \triangleright\ V_i = A^iU$
10: **end for**
11: **return** RS

Utilisé avec des zonotopes, cet algorithme a la même complexité temporelle que l'algorithme 3, mais il a une complexité en espace en $\mathcal{O}(kd + d^2)$, puisqu'il suffit de $\mathcal{O}(d)$ réels pour représenter un rectangle orthogonal aux axes.

Quant à sa précision, la proposition 1 garantie que pour tout i, $\tilde{\Omega}_i$ est une sur-approximation oprimale de Ω_i par un hyper-rectangle. En particulier, $\tilde{\Omega}_i$ et Ω_i ont le même diamètre ϕ.

Par ailleurs, la distance de Hausdorff entre $\tilde{\Omega}_i$ et Ω_i est toujours inférieure à $\frac{\sqrt{d}}{2}\phi$. La distance de Hausdorff étant définie pour deux ensembles A et B par :

$$d_H(A, B) = \max\left(\sup_{x\in A}\inf_{y\in B}\|x - y\|, \sup_{x\in B}\inf_{y\in A}\|x - y\|\right)$$

Ce pire cas est atteint quand $\tilde{\Omega}_i$ est un cube approximant Ω_i, une de ses diagonales.

On peut aussi appliquer cette idée, ou des variantes n'approximant que les plus petits générateurs pour l'ordre utilisé dans l'algorithme de Girard [11], en post-traitement sur la sortie de l'algorithme 3.

3.3 Accélération par changement de variables

On peut aussi vouloir améliorer la complexité temporelle de l'algorithme précédent, encore au détriment de la précision, mais sans provoquer un effet d'emballage, l'intersection de la frontière de $\tilde{\Omega}_i$ avec Ω_i reste non vide.

L'opération la plus coûteuse à chaque éxécution de la boucle sur i dans l'algorithme 4 est la transformation linéaire, la somme de Minkowski se faisant rapidement sur les rectangles orthogonaux aux axes, et l'approximation BOX se calculant rapidement pour la plupart des représentations. Pour améliorer la complexité temporelle, il faut donc s'intéresser à la transformation linéaire, en particulier on peut chercher à se placer dans un système de coordonnées dans lequel la matrice de la transformation linéaire, $\mathcal{A}$, est creuse, on pourra alors l'exécuter en $\mathcal{O}(d^2)$.

Mais il faut ensuite revenir dans le système de coordonnées initial, la matrice de passage P n'étant pas creuse à priori, il faut que les ensembles considérés ne soient représentés que par $\mathcal{O}(1)$ vecteurs, c'est pourquoi nous devons utiliser une approximation rectangulaire.

On note $SPARSE$ un algorithme qui décompose une matrice A en le produit $P\mathcal{A}P^{-1}$ avec $\mathcal{A}$ creuse. On peut alors écrire l'algorithme 5.

En utilisant cette algorithme avec des zonotopes d'ordre $\mathcal{O}(1)$ on a une complexité en $\mathcal{O}(kd^2 + \mathcal{M}(d) + \mathcal{S}(d))$, où $\mathcal{M}(d)$ est la complexité de la multiplication de deux matrices $d \times d$, et $\mathcal{S}(d)$ la complexité de l'algorithme $SPARSE$.

Quant à la précision de cet algorithme, elle dépend de la matrice P, c'est pourquoi on ne doit pas choisir n'importe quelle décomposition. En effet, si les angles entre les vecteurs de la nouvelle base sont importants, l'approximation sera d'autant plus grossière. Cette effet peut se voir sur la figure 3.1, la forme obtenue par l'approximation aura des arêtes colinéaires aux vecteurs de la nouvelle base.

On veut que l'approximation d'une boule de rayon 1 ait le plus petit diamètre possible, or l'approximation d'une boule de rayon 1 est :

$$P\mathrm{BOX}(P^{-1})$$

Algorithme 5 FRS

Entrée : une matrice A, deux ensembles Ω_0 et U, et un entier k.

Sortie : les k premiers termes de la suite définie par $\Omega_{i+1} = A\Omega_i \oplus U$ approximés, de façon optimale, par un zonotope d'ordre 1 définie à partir de A.

1: $(\mathcal{A}, P, P^{-1}) \leftarrow$SPARSE$(A)$ ▷ $A = P\mathcal{A}P^{-1}$
2: $X_0 \leftarrow P^{-1}\Omega_0$
3: $V_0 \leftarrow P^{-1}U$
4: $S_0 \leftarrow \{0\}$
5: $RS \leftarrow P\text{Box}(P^{-1}\Omega_0)$
6: **for** i from 1 to k **do**
7: $\quad X_i \leftarrow \mathcal{A}X_{i-1}$ ▷ $X_i = P^{-1}A^i\Omega_0$
8: $\quad S_i \leftarrow S_{i-1} \oplus \text{Box}(V_{i-1})$ ▷ $S_i = \text{Box}(\bigoplus_{j=0}^{i-1} A^jU)$
9: $\quad RS \leftarrow RS \cup P(\text{Box}(X_i) \oplus S_i)$ ▷ $RS = \bigcup_{j=0}^{i} P\text{Box}(P^{-1}\Omega_j)$
10: $\quad V_i \leftarrow \mathcal{A}V_{i-1}$ ▷ $V_i = A^iU$
11: **end for**
12: **return** RS

FIG. 3.1 – Approximation d'un carré pour différentes matrices P

Le critère à minimiser est donc $\|P\|\|P^{-1}\|$. On peut pour faire la décomposition $A = P\mathcal{A}P^{-1}$ utiliser l'algorithme de [17].

Par ailleurs, quelle que soit la méthode utilisée pour faire cette décomposition, on peut toujours calculer $\|P\|\|P^{-1}\|$ et décider de ne pas utiliser cette algorithme si $\|P\|\|P^{-1}\|$ est trop grand.

3.4 Mise en œuvre et résultats

Les algorithmes 3 et 4, ainsi qu'une variante ne faisant l'approximation rectangulaire que sur la perturbation U, que nous appelerons algorithme U-rectangulaire, ont été implémentés en OCaml. Le programme se décompose en plusieurs modules :

- *zio.ml* pour la gestion des entrées/sorties.
- *zlinalg.ml* pour les opérations d'algèbre linéaire, un algorithme en $\mathcal{O}(d^3)$ est utilisé pour la multiplication matricielle. Une version pour les matrices creuse à aussi été implémentée, mais pour l'instant, elle ne sert que quand la matrice d'entrée est creuse, la décomposition nécessaire à l'algorithme 5 n'ayant pas été implémentée.
- *zonotope.ml* pour la manipulation des zonotopes.
- *reachset.ml* contient les algorithme 3 et 4, ainsi qu'une implémentation de l'algorithme 2 pour les zonotopes avec l'approximation proposée par Girard [11].
- *drawable.ml* définition d'une classe drawable, et des rectangles et zonotopes en dimension 2, ainsi que les procédures d'affichages.
- *main.ml* appelle reachset sur le problème entré et affiche les ensembles calculés.

Tous les calculs ont été effectués sur un Pentium III 800MHz disposant de 256Mo de RAM.

3.4.1 Un premier exemple

Ce premier exemple est issu de [11].

$$\dot{x} = (PDP^{-1})x + u$$

avec $\|u\| \leq 0.01$ et

$$P = \begin{pmatrix} 0.6 & -0.1 & 0.1 & 0.7 & -0.2 \\ -0.5 & 0.7 & -0.1 & -0.8 & 0. \\ 0.9 & -0.5 & 0.3 & -0.6 & 0.1 \\ 0.5 & -0.7 & 0.5 & 0.6 & 0.3 \\ 0.8 & 0.7 & 0.6 & -0.3 & 0.2 \end{pmatrix}$$

$$D = \begin{pmatrix} -1 & -4 & 0 & 0 & 0 \\ 4 & -1 & 0 & 0 & 0 \\ 0 & 0 & -3 & 1 & 0 \\ 0 & 0 & -1 & -3 & 0 \\ 0 & 0 & 0 & 0 & -2 \end{pmatrix}$$

L'ensemble initial est un cube de rayon 0.01 centré en $(1, 0, 0, 0, 0)$.

C'est un exemple en dimension 5, comme décrit en 1.3.1, on commence par choisir un pas de temps, $\Delta t = 0.005$, et on transforme ce système dynamique en un système en temps discret (par un algorithme implémenté en scilab).

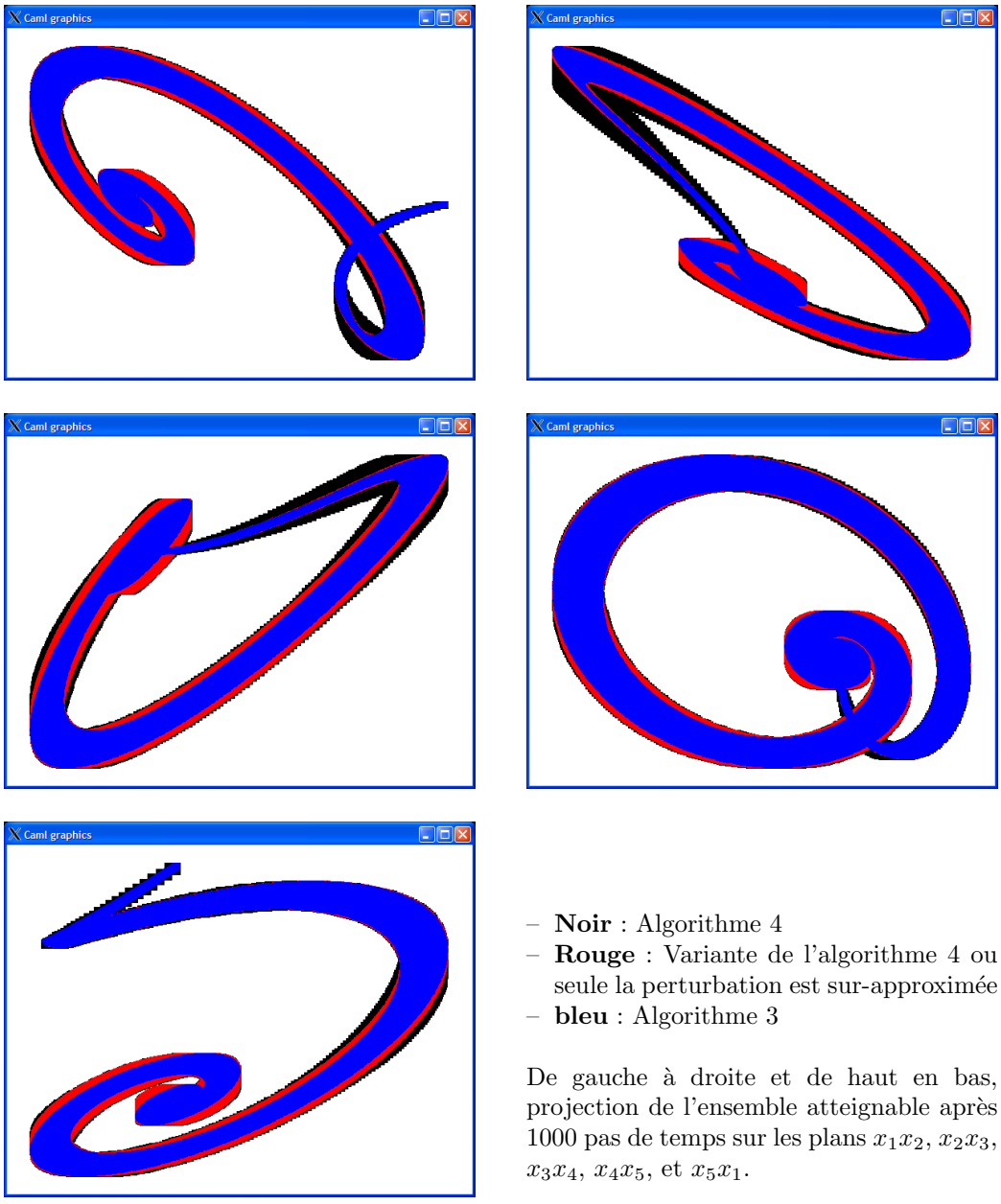
Sur la figure 3.2 on peut voir l'ensemble atteignable après 1000 pas de temps, en bleu, calculé par l'algorithme 3. Les approximations calculées par l'agorithme 4 (en noir) et sa variante n'approximant que la perturbation (en rouge) restent très proches de l'ensemble exact. En effet, comme on l'a déjà dit, pour tout i, Ω_i (ou $\bigoplus_{j=0}^{i-1} A^j U$) est approximé de façon *optimale* par un rectangle. Ainsi, quand Ω_i à une forme proche d'un rectangle, l'approximation est excellente, par contre, s'il a une forme allongée dans une direction non parallèle aux axes du système, l'approximation est plus mauvaise.

Calculer ces ensembles est très rapide avec notre algorithme, seulement quelques centièmes de secondes, et avec une consommation mémoire d'environ un megaoctet. La table 3.1 présente le temps de calcul et la consommation mémoire de nos algorithmes, ainsi que de l'algorithme de Girard [11] (avec différentes bornes pour l'ordre des zonotopes considérés) et de l'algorithme 1 implémenté avec des zonotopes[1], pour le calcul de l'ensemble atteignable par ce système apres 200, 400, 600, 800, ou 1000 pas de temps.

Plusieurs remarques peuvent être faites à partir de cette table. Tout d'abord, les algorithmes 3 et U-rectangulaire sont à peu près aussi rapide l'un que l'autre et ont une consommation mémoire semblable, ce qui ne contredit pas la théorie, puisque ces deux algorithmes ont la même complexité. Or l'algorithme 3 est exact, pourquoi alors utiliser l'approximation U-rectangulaire si elle n'est pas plus rapide ou plus compacte ? En fait, comme on l'a déjà dit, c'est parce que les zonotopes renvoyés par l'algorithme exact ont quelques milliers de générateurs (quand $k = 1000$), alors que ceux renvoyés par le second algorithme n'en ont qu'une vingtaine, ils sont donc plus facile à manipuler. En particulier, pour obtenir la figure 3.2 il a fallu attendre plusieurs secondes l'affichage de l'ensemble atteignable *exact*, quand son approximation est affichée quasi-instantanément.

Par ailleurs, on peut remarquer que l'algorithme 1 est plus rapide que

[1] Pour tous ces algorithmes, les temps de calcul et la consommation mémoire sont pris pour une implémentation en OCaml utilisant les mêmes modules pour la manipulation des zonotopes et pour les opérations d'algèbre linéaire, et les mêmes entrées.



- **Noir** : Algorithme 4
- **Rouge** : Variante de l'algorithme 4 ou seule la perturbation est sur-approximée
- **bleu** : Algorithme 3

De gauche à droite et de haut en bas, projection de l'ensemble atteignable après 1000 pas de temps sur les plans x_1x_2, x_2x_3, x_3x_4, x_4x_5, et x_5x_1.

FIG. 3.2 – Ensemble d'atteignabilité après 1000 pas de temps

$k =$	200	400	600	800	1000
Exacte (Alg 3)	0.01s	0.02s	0.04s	0.05s	0.07s
U-rectangulaire	0.01s	0.02s	0.03s	0.05s	0.06s
Rectangulaire	0.s	0.s	0.01s	0.01s	0.02s
Girard-5	0.08s	0.19s	0.28s	0.35s	0.45s
Girard-10	0.17s	0.32s	0.50s	0.64s	0.86s
Girard-20	0.34s	0.74s	1.14s	1.46s	2.16s
Girard-40	0.66s	1.48s	2.28s	3.27s	3.96s
Exacte (Alg 1)	0.42s	1.67s	3.91s	7.89s	12.56s

$k =$	200	400	600	800	1000
Exacte (Alg 3)	492ko	737ko	983ko	1.23Mo	1.47Mo
U-rectangulaire	246ko	737ko	983ko	1.23Mo	1.23Mo
Rectangulaire	246ko	246ko	246ko	246ko	246ko
Girard-5	492ko	983ko	1.23Mo	1.72Mo	1.97Mo
Girard-10	737ko	1.47Mo	2.21Mo	3.19Mo	3.69Mo
Girard-20	1.47Mo	2.95Mo	4.18Mo	5.65Mo	6.88Mo
Girard-40	2.70Mo	5.41Mo	8.36Mo	10.6Mo	13.8Mo
Exacte (Alg 1)	5.90Mo	23.1Mo	51.4Mo	90.9Mo	141.6Mo

Tab. 3.1 – Temps de calcul et consommation mémoire pour l'exemple 3.4.1

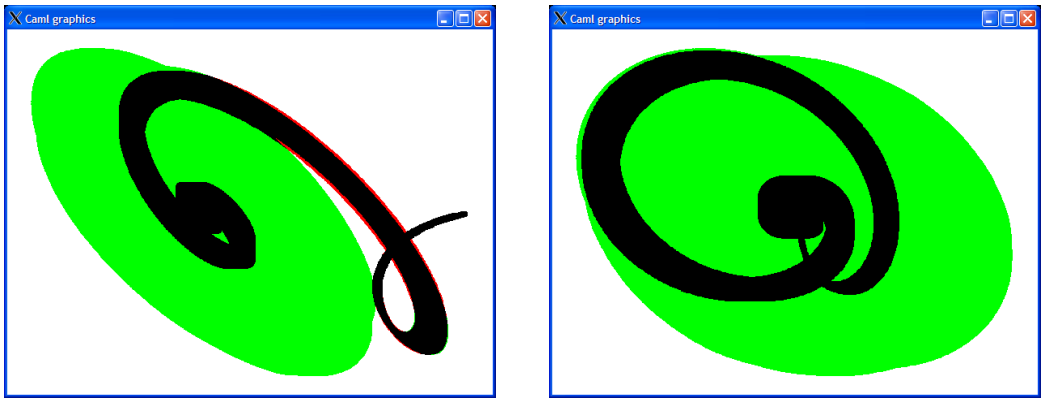


FIG. 3.3 – Comparaison de la précision de l'algorithme de Girard limité à des zonotopes d'ordre 20 (en vert) et une variante de l'algorithme 4 ou seule la perturbation est sur-approximée (en rouge). En noir, l'intersection des deux ensembles calculés.

l'algorithme de Girard limitant l'ordre des zonotopes à 40 sur les 200 premiers pas de temps. C'est à cause de la procédure d'approximation, qui nécessite, à chaque étape, de calculer la norme, et de trier, 200 vecteurs (40×5). Mais rapidement, le coût de cette opération est compensé par la diminution du nombre de générateurs, et donc l'accélération de la transformation linéaire.

Il est clair que nos algorithmes sont plus rapides que ceux auxquels on les a comparés. Mais on peut se demander si ils sont vraiment plus précis. En fait, l'effet d'emballage après 1000 pas de temps est tellement important quand on limite l'ordre des zonotopes à 5 où 10 que nous allons directement comparer l'ensemble calculé quand on limite l'ordre des zonotopes à 20, à celui calculé avec l'approximation U-rectangulaire (ordre des zonotopes : 4).

On voit sur la figure 3.3 que si les ensembles calculés sont relativement précis, et même parfois plus prćis que notre algorithme sur les premiers pas de temps, l'effet d'emballage produit un erreure importante au bout de 1000 pas de temps. Il faut aller jusqu'à l'odre 40 pour avoir un résultat comparable au notre (figure 3.4), mais, en plus d'être plus lent, cette algorithme renvoie des zonotopes d'ordre 40, alors que le notre renvoie des zonotopes d'ordre 4, plus faciles à manipuler.

3.4.2 Influence de la dimension

Dans cette section nous étudions l'influence de la dimension du système considéré sur le temps de calcul et la consommation mémoire de nos algorithmes et notre implémentation de l'algorithme de Girard. Les résultats de

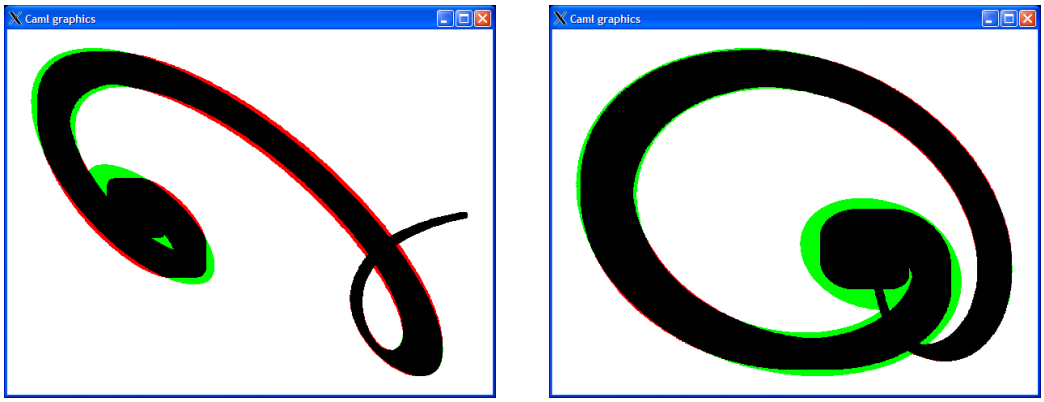


FIG. 3.4 – Comparaison de la précision de l'algorithme de Girard limité à des zonotopes d'ordre 40 (en vert) et une variante de l'algorithme 4 ou seule la perturbation est sur-approximée (en rouge). En noir, l'intersection des deux ensembles calculés.

nos simulations sont présentés dans la table 3.2.

Pour construire cette table, nous avons généré des matrices $d \times d$ dont les coefficients sont tirés aléatoirement selon une distribution uniforme entre -1 et 1. Ces matrices définissent des systèmes :

$$\dot{x} = Ax + u$$

avec $\|u\| \leq 0.01$ et $x(0) \in [0.99; 1.01] \times [-0.01; 0.01] \times \ldots \times [-0.01; 0.01]$.

Le système est ensuite discrétisé avec un pas de temps de 0.01 comme décrit en 1.3.1.

3.4.3 Influence du pas de temps

Une autre propriété de notre algorithme qu'il est intéressant de mettre en évidence, est que, puisqu'il n'est pas victime de l'effet d'emballage, diminuer le pas de temps utilisé lors de la discrétisation du système dynamique permet d'obtenir une sur-approximation plus précise de l'ensemble atteignable par le système.

Ceci n'est pas vrai pour les autres algorithmes, en effet, si on diminue le pas de temps, on augmente le nombre de pas nécessaires pour couvrir le même intervalle de temps, or l'effet d'emballage augmente avec le nombre de pas considérés.

Sur la figure 3.5, on peut voir l'ensemble atteignable par un système de dimension 20 (généré comme précédemment) pour t entre 0 et 1, discrétisé avec un pas de temps de 0.1 (en rouge), 0.01 (en vert), et 0.001 (en bleu). À

$d =$	5	10	20	50	100	150
Exacte (Alg 3)	0.0s	0.02s	0.11s	1.11s	8.43s	35.9s
U-rectangulaire	0.0s	0.02s	0.10s	1.08s	8.42s	34.2s
Rectangulaire	0.0s	0.01s	0.07s	0.91s	8.08s	28.8s
Girard-5	0.03s	0.16s	0.80s	6.09s	32.6s	103s
Girard-10	0.07s	0.34s	1.51s	11.7s	62.3s	199s
Girard-20	0.16s	0.61s	3.32s	22.6s	152s	
Girard-40	0.25s	1.09s	4.84s	38.3s		
Exacte (Alg 1)	0.08s	0.44s	2.34s	17.3s		

$d =$	5	10	20	50	100	150
Exacte (Alg 3)	246ko	492ko	1.72Mo	8.85Mo	33.7Mo	75.2Mo
U-rectangulaire	246ko	492ko	1.47Mo	8.60Mo	33.7Mo	75.2Mo
Rectangulaire	246ko	246ko	246ko	492ko	983ko	2.21Mo
Girard-5	246ko	737ko	2.70Mo	15.0Mo	57.5Mo	125Mo
Girard-10	492ko	1.47Mo	4.67Mo	25.6Mo	99.5Mo	215Mo
Girard-20	737ko	2.46Mo	8.36Mo	44.5Mo	177Mo	
Girard-40	1.2Mo	4.18Mo	14.5Mo	77.9Mo		
Exacte (Alg 1)	1.7Mo	5.41Mo	19.4Mo	114Mo		

TAB. 3.2 – Temps de calcul et consommation mémoire en fonction de la dimension du système, pour $k = 100$.

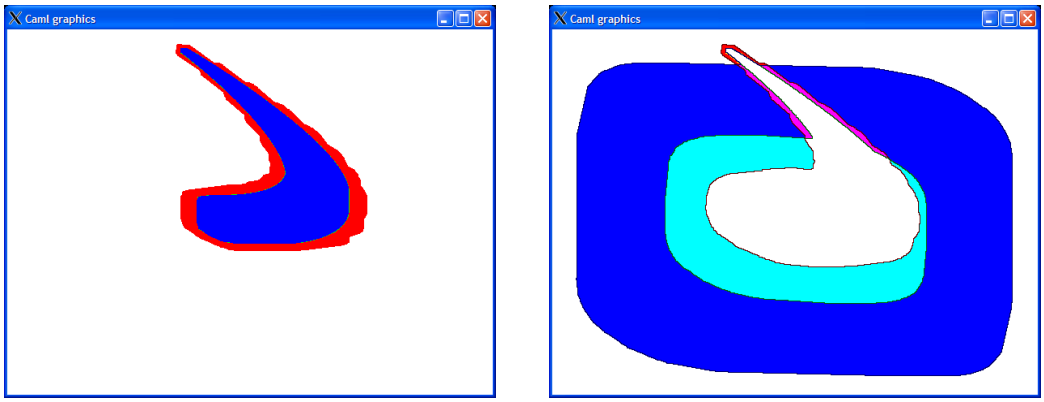


FIG. 3.5 – Mise en evidence de l'effet d'emballage.

gauche, les ensembles calculés par notre algorithme avec l'approximation U-rectangulaire sont bien plus précis quand on prend un pas de temps plus petit. Par contre, à droite, l'algorithme de Girard limité à des zonotopes d'ordre 5 ne vérifie pas cette propriété, l'ensemble calculé avec un pas de temps de 0.001 prend presque toute la fenêtre. Pour pouvoir observer les trois ensembles sur la même figure, nous avons dû ici représenter l'intersection en sommant les couleurs, ainsi, l'intersection des trois ensembles est la zone centrale blanche (rouge+vert+bleu).

3.4.4 Comparaison avec l'algorithme FWF

Pour l'instant, nous n'avons pas comparé notre algorithme à celui de Kühn [14, 15], car nous ne l'avons pas implémenté en OCaml. Mais sur son site web, on peut avoir accès à une version de son algorithme limitée à la dimension 2 via une applet Java :

`http://www.decatur.de/personal/Java1.0/ivp/Ivp.html`

Nous allons nous en servir pour étudier le système suivant :

$$\Omega_{i+1} = \begin{pmatrix} 0 & 0.99 \\ -1 & 1.6 \end{pmatrix} \Omega_i \oplus \square_{0.001} \tag{3.1}$$

Avec $\Omega_0 = \{(1, 0.8)\}$.

La figure 3.6 montre l'ensemble atteignable au bout de 500 pas de temps calculé exactement par l'algorithme de Kühn (en limitant l'ordre des zonotopes à 501, qui n'est atteint qu'après 500 pas de temps).

Sur cet exemple, l'algorithme de Girard limité à des zonotopes d'ordre 5 se comporte bien, il est en fait même meilleur que l'algorithme 4. Par contre, l'algorithme de Kühn engendre une erreur très importante, même en autorisant des zonotopes d'ordre 50 l'erreur reste considérable.

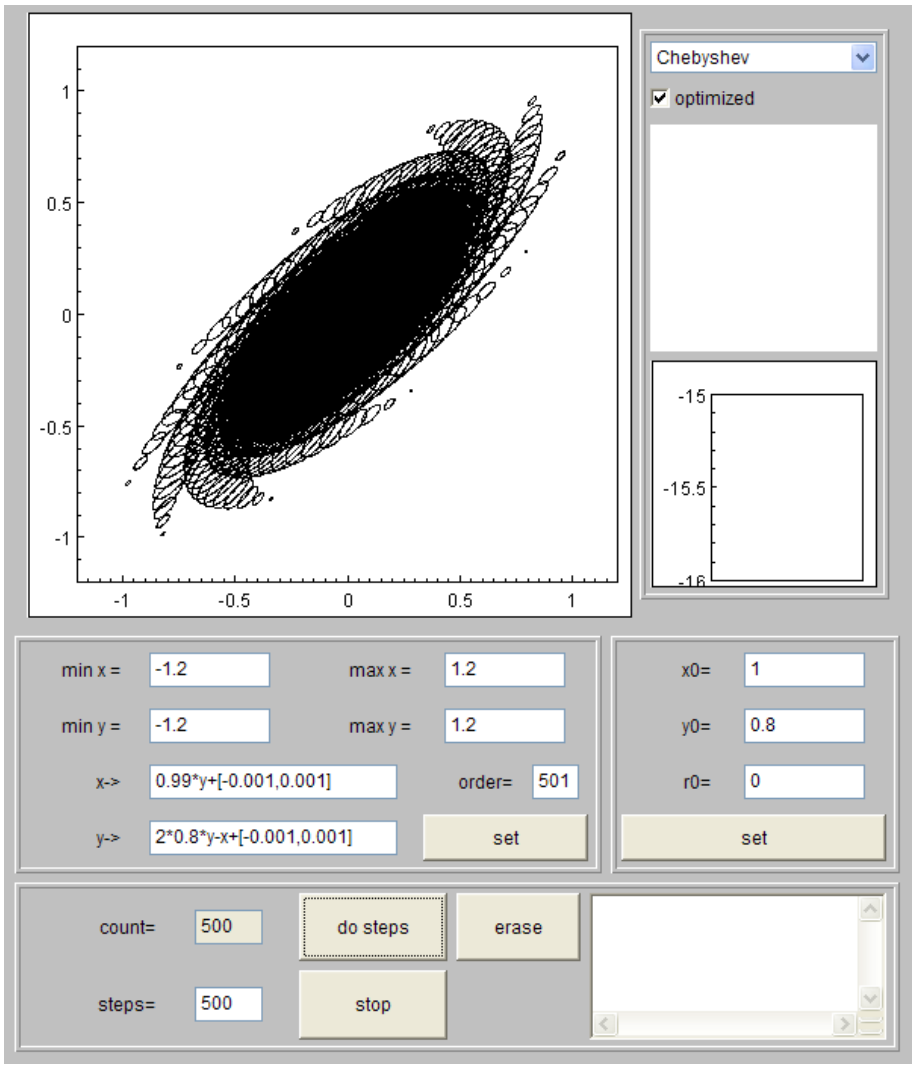
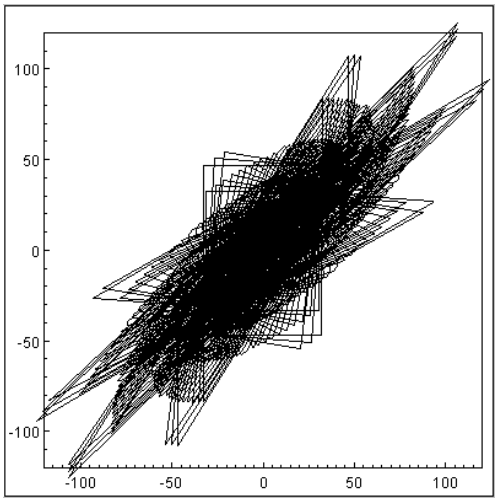
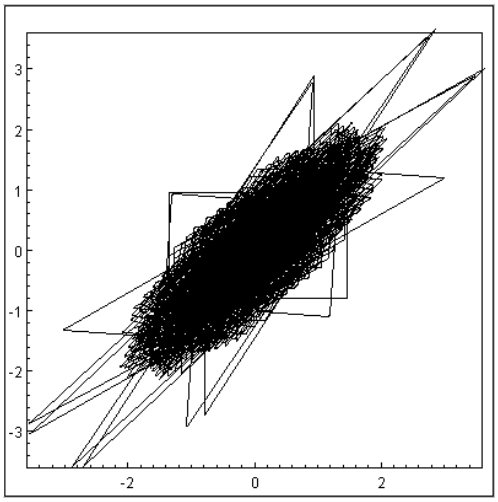


FIG. 3.6 – Applet Java [13] de l'algorithme FWF (ou *cascade reduction*)





En haut : Ensemble atteignable calculé par FWF avec des zonotopes d'ordre 5 (à gauche) et 50 (à droite).
En bas à gauche[a] :
- **rouge** : Algorithme 4
- **vert** : Algorithme de Girard avec des zonotopes d'ordre 5
- **noir** : intersection des deux ensembles calculés

[a] les bornes de la fenêtre sont environ $[-1.2; 1.2] \times [-1.2; 1.2]$

FIG. 3.7 – Comparaison de FWF avec l'algorithme de Girard et l'algorithme 4 sur l'exemple 3.1 en dimension 2 après 500 pas de temps

3.4.5 Arithmétique flottante

L'algorithme de Kühn est donc beaucoup moins bon que celui de Girard sur l'exemple précédent. Mais une remarque que l'on peut faire, est que celui de Kühn prend en compte les erreurs d'arrondis dues à l'utilisation de l'arithmétique flottante pour garantir que l'ensemble calculé est bien une sur-approximation de l'ensemble d'accessibilité, contrairement au notre.

Le problème avec notre algorithme est qu'il faut que A et U ne dépendent pas de i, or pour pouvoir prendre en compte les erreurs d'arrondis, il faudrait rajouter, à chaque étape un u_i prenant en compte les erreurs d'arrondis de l'étape i :

$$\Omega_{i+1} = A\Omega_i \oplus U \oplus u_i$$

Pour les prendre en compte, une solution est de calculer, avec l'algorithme 2 (par exemple celui de Girard) les termes de la suite :

$$E_{i+1} = AE_i \oplus u_i$$

avec $E_0 = \emptyset$ et u_i les erreurs d'arrondis engendrées à l'étape i.

Mais alors, l'algorithme devient victime de l'effet d'emballage.

Chapitre 4

Extension aux systèmes hybrides

Notre algorithme peut être incorporé dans un outils de vérification pour les systèmes hybrides. Pour chaque état où la dynamique est linéaire avec incertitude, on peut calculer efficacement une sur-approximation de l'ensemble atteignable sur un intervalle de temps borné. Il faut ensuite avoir un outil pour détecter et calculer l'intersection avec les gardes.

4.1 Intersection

Nous supposerons ici que les gardes sont des hyperplans.

$$G = \{x \in \mathbb{R} \mid n_G^T x = e\}$$

avec $n_G \in \mathbb{R}^d$ et $e \in \mathbb{R}$.

Les zonotopes ne sont pas clos par intersection avec un hyperplan ou un demi-espace. De plus, si décider du vide de cette intersection est très facile, nous verrons qu'il est difficile d'obtenir une bonne approximation de cette intersection.

4.1.1 Détection de l'intersection

Pour les zontopes, détecter l'intersection avec un hyperplan est très facile. En effet, un zonotope $\langle u;\, v_1 \dots v_m \rangle$ intersecte une garde G si et seulement si il existe $(\alpha_1 \dots \alpha_m)$ dans $[-1;\, 1]^m$ tel que :

$$n_G^T u + \sum_{i=0}^{m} \alpha_i n_G^T v_i = e$$

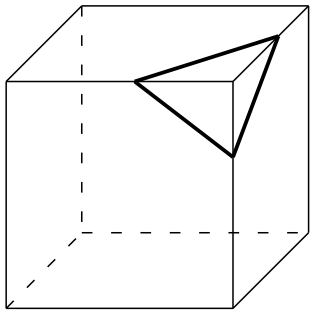


FIG. 4.1 – Les zonotopes ne sont pas clos par intersection

Il suffit donc de déterminer si e appartient au zonotope de dimension 1 défini par :

$$\langle n_G^T u;\ n_G^T v_1, \ldots, n_G^T v_m \rangle$$

Ce qui revient à déterminer si :

$$(e - n_G^T u) \in [-\sum_{i=0}^{m} |n_G^T v_i|;\ \sum_{i=0}^{m} |n_G^T v_i|]$$

Ainsi détecter l'intersection d'un zonotope avec un hyperplan se fait en temps linéaire en m, le nombre de générateurs du zonotope, et linéaire en d, la dimension du système.

Par ailleurs, sur le même principe que l'approximation rectangulaire de l'algorithme 4, on peut calculer efficacement, et exactement, l'intevalle :

$$[-\sum_{i=0}^{m} |n_G^T v_i|;\ \sum_{i=0}^{m} |n_G^T v_i|]$$

À chaque étape, il suffit de projeter les zonotopes $A^i\Omega_0$ et $A^{i-1}U$ sur n_G, la projection de $\bigoplus_{j=0}^{i-2} A^j U$ ayant déjà été calculée à l'étape précédente.

4.1.2 Intersection avec un hyper-plan

La classe des zonotopes, n'est pas close par intersection avec un hyperplan, prenons par exemple le cas de l'intersection d'un cube avec un hyperplan coupant un de ses coins : l'intersection est un triangle (figure 4.1), qui, n'ayant pas même une symmétrie centrale, ne peut être un zonotope.

Il nous faut donc trouver un moyen de sur-approximer cette intersection. Pour cela, on peut projeter le zonotope sur l'hyperplan considéré suivant une certaine direction n_π.

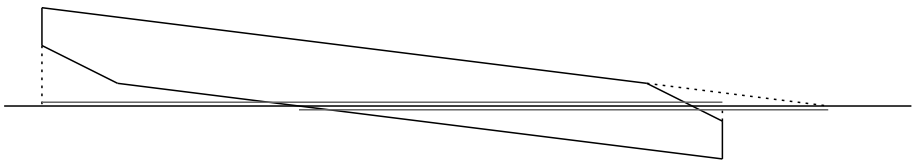


FIG. 4.2 – Choix de la direction de projection

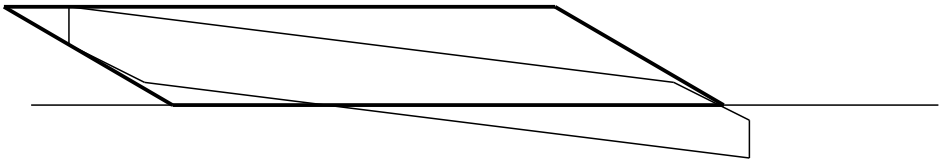


FIG. 4.3 – Intersection avec un demi-espace

Malheureusement si cette direction est mal choisie, cela peut produire une sur-approximation importante. Le choix de la normale à l'hyperplan, ou du générateur v du zonotope tel que $\|v\|$, ou $\frac{(v|n_G)}{\|v\|}$ ou même $(v|n_G)$ soit maximal n'est pas nécessairement judicieux, comme l'illustre la figure 4.2.

Une heuristique pour choisir la direction de la projection est de prendre :

$$n_\pi = \sum_{i=0}^{m} \frac{(v_i|n_G)}{\|v_i\|} v_i$$

L'idée est de choisir la direction à partir des générateurs du zonotope en favorisant les générateurs dont la projection sur la normale, n_G, à la garde, est importante. Par ailleurs, si deux générateurs v_i et v_j sont colinéaires, les remplacer par v_i+v_j ne changera pas la direction choisie par cette heuristique.

4.1.3 Intersection avec un demi-espace

Pour l'intersection avec un demi-espace la solution retenue est de prendre comme sur-approximation le cylindre dont la base est la sur-approximation de l'intersection avec l'hyperplan obtenue par la méthode précédente et de direction n_π.

Cette sur-approximation présente l'intérêt de garantir que le zonotope obtenu est dans le demi-espace considéré. Malheureusement, comme toute méthode ayant cette propriété, il engendre une sur-approximation qui peut être

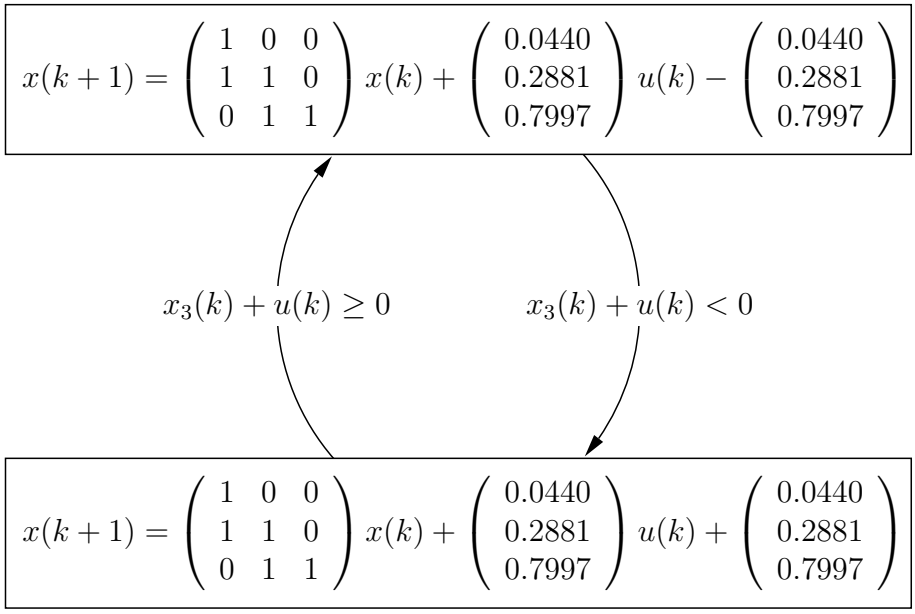


FIG. 4.4 – Automate hybride modélisant un modulateur Δ–Σ

importante, en effet, un zonotope ayant un centre de symétrie, l'image de l'intersection avec la garde par ce centre doit appartenir à la sur-approximation obtenue.

Par ailleurs, cette méthode augmente le nombre de générateurs (+1) et renvoie un zonotope avec de nombreux générateurs coplanaires (tous, sauf un), ce qui réduit le nombre de faces[1].

4.2 Δ–Σ

La modulation Δ–Σ est une technique répandue de conversion d'un signal analogique en un signal numérique [3].

Dans [7], un modèle hybride d'un modulateur Δ–Σ, généré à l'aide de [21], est étudié. Il s'agit d'un automate à deux états décrit dans la figure 4.4.

On peut remarquer que pour cet automate, la garde dépend de l'entrée u, nous avons donc légèrement modifié notre procédure calculant l'intersection avec la garde pour tenter de prendre en compte cette incertitude.

La propriété de sûreté à vérifier est que $\mid x_1 \mid$ doit être inférieur à 0.2. En 26.1 secondes notre algorithme a pu déterminer que cette propriété est

[1] Un des arguments pour l'utilisation des zonotopes était qu'un petit nombre de générateurs peut éngendrer une structure complexe ayant un grand nombre de faces.

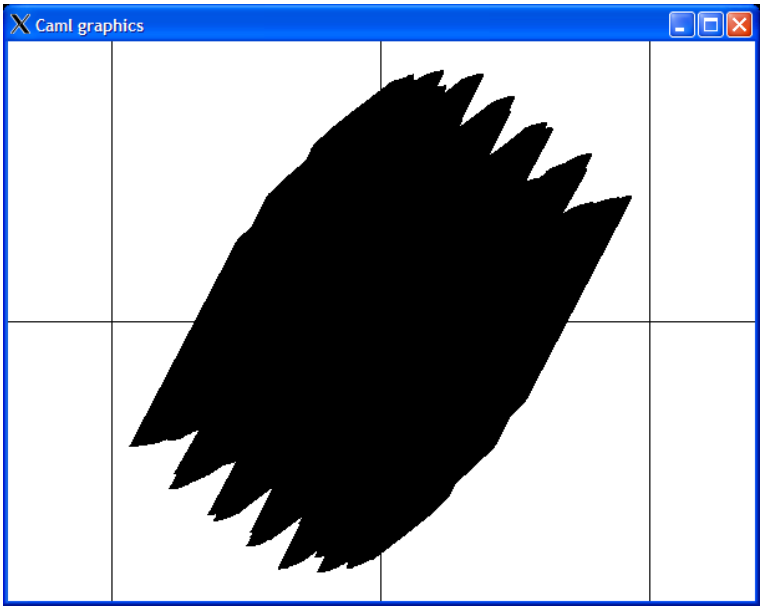


FIG. 4.5 – Sur-approximation de l'ensemble atteignable par le modulateur Δ-Σ après 20 pas de temps

vérifié pendant, au moins, les 20 premiers pas de temps, avec un ensemble initial égal au cube $[-0.01; 0.01]^3$ et une incertitude u comprise entre -0.1 et 0.1. L'ensemble atteignable au bout de 20 pas de temps est présenté sur la figure 4.5. La lenteur de l'algorithme s'explique par le fait que cet automate hybride change souvent d'état discret, opération peu favorable à l'utilisation des zonotopes, ainsi, après 20 pas de temps, l'algorithme à consommé 152Mo, et à engendré 145445 zonotopes pour décrire l'ensemble atteignable.

4.3 Système à deux réservoirs

Le système à deux réservoirs (figure 4.6) à d'abord été présenté dans [20]. Il s'agit de deux réservoirs et deux valves, la première valve permet d'ajouter de l'eau dans le premier réservoir, et la deuxième d'en enlever du second réservoir. Il y a aussi une arrivé d'eau constante dans le premier réservoir, et une fuite dans le second.

Le système hybride est obtenu par linéarisation autour d'un point de fonctionnement. Une stratégie de commutation ON/OFF permet de rester autour de ce point. La dynamique discrète est donnée par l'automate présenté

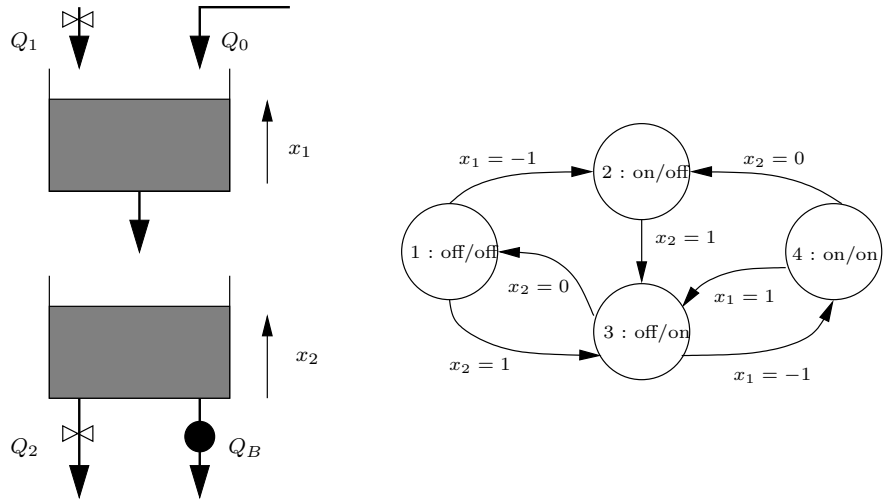


FIG. 4.6 – Système à deux réservoirs

sur la figure 4.6, et la dynamique continue par :

$$\dot{x} = A_q x + b_q$$

avec

$$A_1 = \begin{pmatrix} -1 & 0 \\ 1 & 0 \end{pmatrix} A_2 = \begin{pmatrix} -1 & 0 \\ 1 & 0 \end{pmatrix} A_3 = \begin{pmatrix} -1 & 0 \\ 1 & -1 \end{pmatrix} A_4 = \begin{pmatrix} -1 & 0 \\ 1 & -1 \end{pmatrix}$$

et

$$b_1 = \begin{pmatrix} -2 \\ 0 \end{pmatrix} \quad b_2 = \begin{pmatrix} 3 \\ 0 \end{pmatrix} \quad b_3 = \begin{pmatrix} -2 \\ -5 \end{pmatrix} \quad b_4 = \begin{pmatrix} 3 \\ -5 \end{pmatrix}$$

Ce système admet un cycle limite, dans [11], Girard propose d'ajouter une perturbation au système pour tester la robustesse de ce cycle. Cette perturbation peut modéliser les incertitudes sur les caractéristiques des valves par exemple. la nouvelle dynamique est donnée par :

$$\dot{x}(t) = A_q x(t) + b_q + u(t)$$

avec $\|u\| < \mu$.

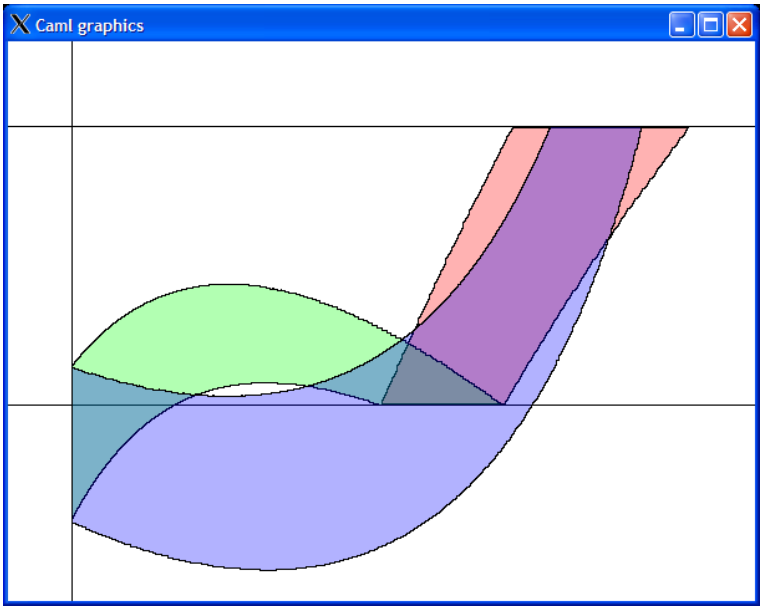


FIG. 4.7 – Ensemble atteignable par le système à deux réservoirs

Nous avons calculé l'ensemble atteignable par ce système avec $\mu = 0.1$ et en partant de $[1.5; 2.5] \times \{1\}$ dans l'état discret 3. La figure 4.7[2] montre que le cycle limite est conservé. En rouge, le système est dans l'état discret 3, dans la zone verte, il est dans l'état discret 1, et enfin, dans la zone bleue, il est dans l'état discret 2. Le système revient alors dans l'état 3 avec des variables continues contenues dans l'ensemble initial.

[2]temps de calcul : 0., consommation mémoire : 246ko.

Conclusion - Perspectives

Dans ce rapport nous avons présenté plusieurs nouveaux algorithmes pour le calcul de l'ensemble atteignable en un temps fini par un système affine avec incertitude. Le résultat peut être calculé exactement ou sur-approché de façon *optimale* par rapport à la forme de la sur-approximation (hyper-rectangle ou parallélépipède), ils sont donc beaucoup plus précis que les méthodes existantes, victimes de l'effet d'emballage. Cette affranchissement par rapport à l'effet d'emballage leur permet de réellement converger quand on diminue le pas de temps. Ils sont par ailleurs plus rapides et moins demandeurs en mémoire que la plupart des méthodes existantes. Mais contrairement à ces autres méthodes, il ne permet pas de prendre efficacement en compte les erreurs dues à l'utilisation d'une arithmétique flottante. Il ne permet pas non plus d'étudier la dynamique de systèmes dépendant du temps. Par ailleurs, l'utilisation des zonotopes rend difficile l'extension au cas *hybride*, puisque nous ne disposons pas de méthode efficace pour sur-approximer l'intersection avec une garde.

Un objectif pour de futurs travaux est donc de trouver des algorithmes efficaces pour l'intersection.

Il va falloir aussi étudier les problèmes liés à l'utilisation de l'arithmétique flottante, ainsi que l'algorithme utilisant des matices creuses, en particulier il serait intéressant d'avoir un algorithme efficace pour la décomposition creuse.

Avant d'inclure ces algorithmes dans un outil de vérification il va aussi falloir modifier l'implémentation pour utiliser une biliothèque d'algèbre linéaire efficace.

Bibliographie

[1] E. Asarin, T. Dang, and A. Girard. Reachability analysis of nonlinear systems using conservative approximation. In O. Maler and A. Pnueli, editors, *HSCC*, volume 2623 of *Lecture Notes in Computer Science*, pages 20–35. Springer, 2003.

[2] J.-P. Aubin and A. Cellina. *Differential inclusion*. Springer-Verlag, 1984.

[3] P. M. Aziz, H. V. Sorensen, and J. Van der Spiegel. An overview of sigma-delta converters. *IEEE Signal Processing Magazine*, 13 :61–84, Jan. 1996.

[4] M. S. Branicky, V. S. Borkar, and S. K. Mitter. A unified framework for hybrid control : Model and optimal control theory. *IEEE Transactions on automatic control*, 43(1) :31–45, Jan. 1998.

[5] D. Coppersmith and S. Winograd. Matrix multiplication via arithmetic progressions. *J. Symb. Comput.*, 9(3) :251–280, 1990.

[6] T. Dang. *Vérification et synthèse des systèmes hybrides*. PhD thesis, INPG, Oct. 2000.

[7] T. Dang, A. Donzé, and O. Maler. Verification of analog and mixed-signal circuits using hybrid system techniques. In A. J. Hu and A. K. Martin, editors, *FMCAD*, volume 3312 of *Lecture Notes in Computer Science*, pages 21–36. Springer, 2004.

[8] T. Dang and O. Maler. Reachability analysis via face lifting. In T. A. Henzinger and S. Sastry, editors, *HSCC*, volume 1386 of *Lecture Notes in Computer Science*, pages 96–109. Springer, 1998.

[9] A. Girard. Approximate solutions of ODEs using piecewise linear vector fields. In V. Ganzha, E. Mayr, and E. Vorozhtsov, editors, *CASC*, pages 107–120. Technische Universität München, Germany, Sept. 2002.

[10] A. Girard. *Analyse algorithmique des systèmes hybrides*. PhD thesis, INPG, Sept. 2004.

[11] A. Girard. Reachability of uncertain linear systems using zonotopes. In M. Morari and L. Thiele, editors, *HSCC*, volume 3414 of *Lecture Notes in Computer Science*, pages 291–305. Springer, 2005.

[12] P. Gritzmann and B. Sturmfels. Minkowski addition of polytopes : computational complexity and applications to gröbner bases. *SIAM J. Discret. Math.*, 6(2) :246–269, 1993.

[13] W. Kühn. The rigorous inital value problem solver. `http://www.decatur.de/personal/Java1.0/ivp/Ivp.html`.

[14] W. Kühn. Rigorously computed orbits of dynamical systems without the wrapping effect. *Computing*, 61(1) :47–68, 1998.

[15] W. Kühn. Towards an optimal control of the wrapping effect. In T. Csendes, editor, *SCAN 98*, Developments in Reliable Computing, pages 43–51. Kluwer Academic Publishers, 1999.

[16] A. Kurzhanski and I. Vályi. *Ellipsoidal Calculus for Estimation and Control*. Systems & Control : Foundations & Applications. Birkhäuser, 1997.

[17] P.-F. Lavallée and M. Sadkane. Une méthode stable de bloc diagonalisation de matrices : Application au calcul de portrait spectral. Rapport de Recherche 3141, INRIA, Mar. 1997.

[18] R. J. Lohner. Enclosing the solutions of ordinary initial and boundary value problems. In *Computer Arithmetic : Scientific Computation and Programming Languages*, Wiley-Teubner Series in Computer Science, pages 255–286. Stuttgart, 1987.

[19] R. Moore. Automatic local coordinate transformations to reduce the growth of error bounds in interval computation of solutions of ordinary differential equations. In L. B. Rall, editor, *Error in Digital Computation*, volume 2, pages 103–140, New York, New York, 1965. John Wiley and Sons, Inc.

[20] M. Rubensson, B. Lennartson, and S. Pettersson. Convergence to limit cycles in hybrid systems : an example. In *LSS 98*, pages 704–709, 1998.

[21] R. Schreier. Delta sigma toolbox.

[22] S. N. Simic, K. H. Johansson, S. Sastry, and J. Lygeros. Towards a geometric theory of hybrid systems. In N. A. Lynch and B. H. Krogh, editors, *HSCC*, volume 1790 of *Lecture Notes in Computer Science*, pages 421–436. Springer, 2000.

[23] T. Zaslavsky. *Facing Up to Arrangements : Face-Count Formulas for Partitions of Space by Hyperplanes*. Number 154 in Memoirs of the American Mathematical Society. American Mathematical Society, 1975.