Maintenir la viabilité ou la résilience d'un système : les machines à vecteurs de support pour rompre la malédiction de la dimensionnalité ?

Laëtitia Chapel

Université Blaise Pascal

N° d'ordre : 1774 382

THÈSE

présentée devant

l'Université Blaise Pascal - Clermont II

pour obtenir le grade de

DOCTEUR D'UNIVERSITÉ
Spécialité INFORMATIQUE

par

Laetitia CHAPEL

Équipe d'accueil : LISC - Cemagref
École Doctorale : Sciences Pour l'Ingénieur

Titre de la thèse :

Maintenir la viabilité ou la résilience d'un système : les machines à vecteurs de support pour rompre la malédiction de la dimensionnalité ?

Soutenue le 19 octobre 2007 devant le jury composé de :

M. Guillaume DEFFUANT	Directeur de recherche	Directeur de thèse
M. Luc DOYEN	Chargé de recherche	Examinateur
M. Philippe MAHEY	Professeur	Examinateur
M. Christian MULLON	Directeur de recherche	Examinateur
M. Rémi MUNOS	Directeur de recherche	Rapporteur
M. Patrick SAINT-PIERRE	Directeur de recherche	Rapporteur

Remerciements

De nombreuses personnes ont apporté une contribution particulière à ces trois années de thèse et à ce manuscrit, et je tiens à les remercier ici.

Je tiens tout d'abord à remercier Guillaume Deffuant pour avoir dirigé cette thèse et m'avoir fait découvrir un domaine passionnant. Remplissant pleinement son rôle, il a su me guider, me conseiller et m'encourager tout en respectant mon autonomie et mes limites. Ses intuitions, son enthousiasme, ses qualités scientifiques et humaines ont été déterminants dans l'aboutissement de ce travail.

Je remercie très vivement Rémi Munos et Patrick Saint-Pierre d'avoir accepté de rapporter cette thèse. L'attention qu'ils ont portée à ce document et leurs suggestions m'ont permis d'améliorer grandement ce manuscrit.

Je tiens également à remercier les membres du jury Luc Doyen, Philippe Mahey et Christian Mullon pour l'intérêt qu'ils portent à ce travail.

Un grand merci à Maria-Hélèna Ramos et Éric Sauquet pour m'avoir initiée à la recherche durant mon stage de DEA. Leur enthousiasme et encouragements m'ont donné envie de faire une thèse et je n'aurai sans doute pas fait ce choix sans eux.

Au cours de ces trois années, j'ai fait mes premiers pas dans l'enseignement. Je tiens à remercier Stéphane Lallich pour m'avoir donné l'opportunité de vivre cette expérience particulièrement instructive et enrichissante. J'ai également eu la chance de suivre les stages de Frédéric Coursol et Frédéric Verrier ; ils m'ont fait découvrir plusieurs facettes de l'encadrement et ont une part dans ce travail de thèse.

Cette thèse a été réalisée au sein du laboratoire d'ingénierie des systèmes complexes. Je tiens à remercier tous ses membres pour les moments de bonne humeur, qui ont fait que ces trois années ont été si agréables à vivre : Clarisse, François, Isabelle, Nabil, Nicolas, Sophie, Sylvie et Thierry. Un merci tout particulier à Marie-Ange, ma « jumelle » de thèse, pour le quotidien et les discussions sur tout et n'importe quoi. Je remercie Nicolas pour l'aide très précieuse qu'il m'a apportée pour la conception du logiciel.

Je terminerai ces remerciements en évoquant des personnes qui ont contribué moins directement à cette thèse, mais de façon essentielle. Un grand merci à ceux qui ont dû relire ce mémoire dans l'urgence, avec sérieux et bonne humeur : Cédric, Céline, Charlène, Colas, Françoise, Florian, Lila et Nico, Nelly, Nolwenn. Enfin, je remercie Romaric et ma famille pour m'avoir supportée et encouragée tout au long de ces trois années.

Résumé

La théorie de la viabilité propose des concepts et méthodes pour contrôler un système dynamique afin de le maintenir dans un ensemble de contraintes de viabilité. Les applications sont nombreuses en écologie, économie ou robotique, lorsqu'un système meurt ou se détériore lorsqu'il quitte une certaine zone de son espace d'état. A partir du calcul du noyau de viabilité ou du bassin de capture d'un système, elle permet de définir des politiques d'action qui maintiennent le système dans l'ensemble de contraintes choisi. Cependant, les algorithmes actuels d'approximation de noyau de viabilité ou de bassins de capture présentent certaines limitations ; notamment, ils souffrent de la malédiction de la dimensionnalité et leur application est réservée à des problèmes en petite dimension (dans l'espace d'état et des contrôles). L'objectif de cette thèse est de développer et évaluer de nouveaux algorithmes d'approximation de noyau de viabilité et de bassins de capture, en utilisant une méthode d'apprentissage statistique particulière : les machines à vecteur de support (SVMs).

Nous proposons un nouvel algorithme d'approximation de noyau de viabilité, basé sur l'algorithme de Patrick Saint-Pierre, qui utilise une méthode d'apprentissage pour définir la frontière du noyau. Après avoir déterminé les conditions mathématiques que la procédure doit respecter, nous considérons les SVMs dans ce contexte. La définition de l'approximation avec des SVMs permet d'utiliser des méthodes d'optimisation pour trouver un contrôle viable, et ainsi de travailler dans des espaces de contrôle plus importants. Cette fonction permet également de dériver des politiques de contrôle plus ou moins prudentes. Nous appliquons la procédure à un problème de gestion des pêches, en examinant quelles politiques de pêche permettent de garantir la viabilité d'un écosystème. Cet exemple illustre les performances de la méthode proposée : le système comporte six variables d'états et 17 variables de contrôle.

Nous dérivons un algorithme d'approximation de bassins de capture et de résolution de problèmes d'atteinte d'une cible en un temps minimal. Approcher la fonction de temps minimal revient à rechercher le noyau de viabilité d'un système étendu. Nous présentons une procédure qui permet de rester dans l'espace d'état initial, et ainsi d'éviter le coût (en temps de calcul et espace mémoire) de l'addition d'une dimension supplémentaire. Nous décrivons deux variantes de l'algorithme : la première procédure donne une approximation qui converge par l'extérieur et la deuxième par l'intérieur. L'approximation par l'intérieur permet de définir un contrôleur qui garantit d'atteindre la cible en un temps minimal. La procédure peut être étendue au problème de calcul de valeurs de résilience. Nous appliquons la procédure sur un problème de calcul de valeurs de résilience sur un modèle d'eutrophication des lacs.

Les algorithmes proposés permettent de résoudre le problème de l'augmentation exponentielle du temps de calcul avec la dimension de l'espace des contrôles mais souffrent toujours de la malédiction de la dimensionnalité pour l'espace d'état : la taille du vecteur d'apprentissage augmente exponentiellement avec la dimension de l'espace. Nous introduisons des techniques d'apprentissage actif pour sélectionner les états les plus « informatifs » pour définir la fonction SVM, et ainsi gagner en espace mémoire, tout en gardant une approximation précise du noyau. Nous illustrons la procédure sur un problème de conduite d'un vélo sur un circuit, système défini par six variables d'état.

Mots-clés : système dynamique, théorie de la viabilité, machines à vecteur de support, contrôleur lourd de viabilité, contrôle optimal, malédiction de la dimensionnalité, apprentissage actif.

Abstract

Viability theory proposes concepts and tools to control a dynamical system such that it can remain inside a viability constraint set. The applications are frequent in ecology, economics or robotics, where the systems die or badly deteriorate when they leave some regions of the state space. Starting from the viability kernel or capture basin of a system, it enables providing control functions that maintain viability. Nevertheless, current algorithms approximating viability kernel or capture basin show several restrictions; particularly, they suffer the dimensionality curse, and their application is thus limited to problems in low dimension (in the state and control space). The aim of this thesis is to develop and evaluate new algorithms using a specific learning method: Support Vector Machines (SVMs).

We propose a new viability kernel approximation algorithm, based on the Saint-Pierre algorithm, which uses a classification method to define the boundary of the kernel approximation. We establish the mathematical conditions that the classification procedure should fulfill, and we consider SVMs in this context. This classification method defines a kind of barrier function on the boundary of the approximation, which allows using optimisation techniques to compute a viable control, and thus work in higher dimensional control space. This function also enables to define more or less cautious controllers. We apply the algorithm on a fisheries management problem, examining which yield policies allow to ensure the sustainability of a marine ecosystem. This example shows the performance of the algorithm: the system is defined in six dimensions for the state space, and 17 dimensions for the control space.

Starting from the viability kernel approximation algorithm using SVMs, we derive a capture basin approximation algorithm and resolution of hitting target problems. Approximating minimal time function comes down to approximate the viability kernel of an extended system. We present a procedure that approximates capture basin in the initial state space and thus avoid the cost of adding one supplementary dimension (computing and time cost). We describe two variants of this algorithm: the first one provides an outer approximation and the second an inner approximation. Comparing the two results gives an evaluation of the approximating error. Inner approximation enables to define a controller that guarantees to reach the target in minimal time. The procedure can be extended to the problem of minimizing a cost function, when it meets some general conditions. We illustrate this point on the computation of resilience values. We apply the algorithm on a problem of computing resilience values on a model of lake eutrophication.

The proposed algorithms enable solving the exponential growth of the computing time with the control space dimension but still suffer the dimensionality curse for the state space: the training set size growth exponentially with the dimension of the space. We introduce active learning techniques to select the most “informative” states to define the SVM function, and then save memory, while keeping an accurate approximation. We illustrate the procedure on a bike control problem on a track, a 6 dimensional state space problem.

Key words: dynamical system, viability theory, Support Vector Machines, heavy viability controller, optimal control, dimensionality curse, active learning.

Sommaire

Introduction

Viabilité, résilience : le besoin de méthodes plus performantes

Problèmes de viabilité

Maintenir un système dynamique dans un ensemble de contraintes

Le problème principal auquel nous nous intéressons dans cette thèse est celui de définir une politique d'action sur un système, de manière à le maintenir dans un ensemble de contraintes. Ce problème est fréquent en écologie, économie ou robotique, lorsque le système meurt ou se détériore lorsqu'il quitte une certaine région de l'espace. Le but n'est alors pas de choisir une solution « optimale » en fonction d'un certain critère, mais de sélectionner des actions « viables », dans le sens où elles permettent au système de se maintenir dans un ensemble de contraintes.

En écologie, le problème peut être de maintenir la pérennité d'une ressource renouvelable. Dans ce cas, on cherche à maintenir la population au-dessus d'une certaine valeur pour laquelle l'extinction est inévitable. Par exemple, dans le domaine de la gestion d'une ressource marine, [De Lara *et al.*, 2007] analysent la pérennité d'un écosystème constitué de merlus et d'anchois dans le golfe de Gascogne, en fonction du niveau de pêche et du recrutement des espèces (nombre de jeunes poissons constituant la nouvelle classe d'âge annuelle). Ils ont identifié les configurations qui permettent de conserver la durabilité du système, c'est-à-dire maintenir la population de chaque espèce au-dessus d'une certaine valeur. [Bonneuil, 2003] étudie les conditions que doivent respecter les dynamiques d'un système proie-prédateur afin d'éviter l'extinction de l'une ou l'autre espèce.
Le problème peut également être de maintenir la pérennité d'un écosystème, en conciliant des objectifs comme exploitation et conservation. On doit alors évaluer des tailles de population d'espèces en dessous desquelles l'extinction ou l'effondrement de l'espèce est probable, mais également des objectifs économiques liés à l'exploitation de la ressource à atteindre. Le système doit donc se maintenir dans une configuration qui concilie les objectifs économiques et écologiques. Par exemple, [Martinet et Doyen, 2007] étudient les conditions qu'un système proie-prédateur, basé sur une ressource non-renouvelable, doit respecter afin de garantir la consommation des espèces et le stock des ressources. Dans le cas de gestion d'une ressource marine, on souhaite garantir la rentabilité économique des activités de pêche tout en maintenant la pérennité de l'écosystème marin. [Béné *et al.*, 2001] étudient un modèle économique simple de gestion de ressources marines et déterminent des options de management qui garantissent la pérennité écologique et économique du système. Ils mettent également en évidence des configurations irréversibles de sur-exploitation qui provoquent la disparition des espèces marines. [Mullon *et al.*, 2004] étudient un modèle dynamique d'évolution de la biomasse de cinq espèces dans un écosystème marin du sud du Benguela. Ils déterminent, pour des valeurs de pêche constantes, quelles sont les configurations

de l'écosystème qui garantissent le principe de précaution (en évitant l'effondrement des stocks).

Dans le domaine du contrôle en robotique, les automates peuvent avoir besoin d'une sécurité renforcée, en évitant certaines zones prédéfinies d'échec. Par exemple, [Kalisiak et Van de Panne, 2004] considèrent le problème de guidage d'une voiture sur un circuit. Le système se détériore lorsqu'il quitte la route ou qu'il commence à glisser. Un conducteur contrôle la trajectoire de la voiture, mais lorsque l'action demandée fait sortir la voiture de la zone de sécurité, un système automatique corrige la trajectoire afin de garder la voiture en sécurité. [Spiteri *et al.*, 2000] utilisent la même approche pour diriger un robot dans un certain chemin et avec une certaine vitesse, en déterminant les contrôles qui lui permettent de toujours rester en sécurité. Dans le domaine du transport aérien, mais avec toujours l'objectif d'assurer la sécurité du système, [Seube *et al.*, 2000] formulent le problème du décollage d'un avion en présence de rafales de vent comme un jeu différentiel. En fonction du vent, de la vitesse et position de l'avion, le pilote doit contrôler le système afin de respecter les conditions de sécurité et éviter le crash.

Pour terminer, en économie, le problème peut être de maintenir un système de gestion viable. Par exemple, [Aubin *et al.*, 2001] étudient les conditions de pérennité du système des retraites. Le système est composé d'actifs et d'inactifs, avec chacun un pouvoir d'achat. On souhaite alors déterminer des politiques de versement des actifs aux caisses de retraites, qui garantissent un revenu suffisant aux retraités, tout en assurant aux actifs de conserver un pouvoir d'achat garanti. Les seuils fixés sur le déficit et l'endettement des états européens dans le pacte de stabilité sont un autre exemple de contrôle viable.

Extensions de la viabilité : atteindre une cible et résilience

A partir du problème central de la viabilité, nous abordons un problème apparemment différent, celui d'agir sur un système de manière à ce qu'il atteigne une cible. La cible correspond alors à un état souhaité, en sachant que le système se détériore dans une certaine région de l'espace (lorsqu'il transgresse l'ensemble de contraintes). En fait, nous verrons que ce problème est étroitement apparenté à celui de la viabilité.

Ce problème est également très fréquent, notamment dans le domaine de la navigation de robots mobiles. Mais il se rencontre aussi dans des domaines très éloignés, comme la finance. Par exemple, dans un problème d'évaluation et de gestion d'actifs financiers, [Pujal et Saint-Pierre, 2004] posent des contraintes sur la valeur du portefeuille, et construisent des politiques de gestion d'actifs qui permettent d'atteindre un objectif d'obtention de contrats. [Aubin *et al.*, 2005] traitent le problème de la gestion d'un portefeuille d'actifs, et déduisent des politiques d'action qui permettent de contrôler l'évolution du portefeuille, jusqu'à atteindre un objectif de rentabilité fixé, tout en maintenant à chaque instant la valeur du portefeuille au-dessus d'un certain seuil.

Cette première extension du problème de la viabilité permet d'en aborder une seconde : celle de la résilience d'un système. On considère alors que le système peut survivre lorsqu'il quitte l'espace des contraintes. L'ensemble des contraintes peut définir par exemple la bonne santé, ou les états souhaitables du système. Transgresser les contraintes est alors désagréable ou non-souhaitable, mais pas forcément fatal. Le système peut alors quitter l'espace des contraintes, et éventuellement revenir à l'intérieur par la suite. On identifie des situations irréversibles (le système ne peut jamais revenir dans l'ensemble de contraintes) et des configurations réversibles (à partir d'une situation de crise, on peut contrôler le système afin qu'il revienne dans l'ensemble de contraintes). Le lien avec le problème d'atteinte de cible apparaît : il s'agit de contrôler le système de manière à ce qu'il revienne dans une région particulière de l'espace de contraintes. La capacité d'un système à retrouver sa propriété d'intérêt (ici, sa bonne santé ou sa survie) après une perturbation correspond à une définition récente de la résilience [Martin, 2004] sur laquelle nous nous appuyons.

Les problèmes de résilience, ainsi définis, sont très fréquents, notamment en écologie. Ainsi, lorsque l'on souhaite combiner des objectifs comme exploitation et conservation d'une ressource, l'espace des contraintes représente la bonne santé du système : la pérennité du système est assurée en même temps

que les objectifs de rentabilité économique sont atteints. Cependant, lorsque la durabilité de l'écosystème est en jeu, on peut vouloir accepter des configurations économiques non rentables (situation de crise), le temps de laisser se régénérer le système, avant de reprendre une activité viable. Par exemple, dans le domaine de gestion d'une ressource marine, [Béné *et al.*, 2001] identifient des configurations de gestion irréversibles, qui provoquent la disparition des espèces marines et donc de l'activité de pêche, mais également des configurations réversibles. Dans ce cas, les pêcheurs acceptent des revenus moindres pendant une certaine période afin de laisser régénérer la ressource marine. [Martinet *et al.*, Pres] considèrent dans ce contexte l'écosystème marin du golfe de Gascogne, et diminuent la flotte de pêche lorsque l'écosystème est en péril. Dans un esprit similaire, [Martin, 2004] considère le problème d'eutrophication des lacs. La bonne santé du lac est représentée par un niveau bas de phosphate et une activité économique rentable pour les agriculteurs, dont les revenus sont directement liés à l'apport de phosphate. Elle considère une augmentation soudaine de phosphate dans le lac, et identifie trois scénarios : la perturbation ne met pas en cause la viabilité du système, la perturbation entraîne une eutrophication des lacs mais qui peut être renversée en acceptant une période de faible rentabilité économique, la perturbation engendre une situation irréversible. Elle identifie les coûts de restauration liés à chaque état du système, en fonction de la perturbation anticipée.

On le voit, les problèmes de viabilité et leurs extensions (atteinte de cible, résilience) sont essentiels à de nombreux domaines applicatifs. Notre objectif est de développer de nouvelles méthodes, plus performantes, pour les résoudre. Nous présentons maintenant notre démarche générale pour atteindre cet objectif.

Choix d'un cadre théorique : la théorie de la viabilité

Deux cadres théoriques sont disponibles pour aborder les problèmes décrits précédemment : la théorie de la viabilité et le contrôle optimal.

Théorie de la viabilité

La théorie de la viabilité, initiée au début des années 1990 par Jean-Pierre Aubin et ses collaborateurs [Aubin, 1991], se focalise sur le problème de viabilité : maintenir un système dynamique dans un ensemble de contraintes. Elle fournit un ensemble de concepts et de résultats mathématiques, qui ont donné lieu par la suite au développement d'outils informatiques. Nous mentionnons rapidement ces concepts (qui seront définis précisément dans la suite du document), car ils sont importants pour décrire notre démarche générale.

Considérons un système dynamique composé de variables d'*état* qui décrivent le système et de variables de *contrôle*, qui permettent d'agir sur le système. L'ensemble des contraintes définit un sous-ensemble de l'espace d'état en dehors duquel le système meurt ou se détériore, et est appelé ensemble des contraintes de viabilité. La théorie de la viabilité introduit le concept d'*évolution viable* comme une évolution qui reste à chaque instant à l'intérieur de l'espace des contraintes. L'état initial à partir duquel part une évolution viable est appelé *état viable*, et l'ensemble des états viables constitue le *noyau de viabilité*. Le noyau de viabilité est composé de tous les états pour lesquels il existe au moins une politique de contrôle qui permet au système de survivre dans un ensemble de contraintes. Il correspond à l'ensemble des états qui peuvent rester en « vie » ou en « bonne santé ». Au contraire, pour les états situés à l'extérieur du noyau, la viabilité du système est mise en péril puisque les évolutions violent les contraintes en temps fini.

Le noyau de viabilité d'un système est déterminant pour définir des politiques d'action viables. A partir d'un état situé dans le noyau de viabilité, on sait qu'il existe au moins une suite de contrôle qui permet au système de rester dans l'ensemble des contraintes de viabilité. La règle la plus simple a été introduite par [Aubin, 1991] et est appelée *contrôleur lourd*. La procédure découle du principe d'inertie « les contrôles sont gardés constants tant que la viabilité du système n'est pas menacée ». Ainsi, on garde

les contrôles constants tant que le système n'atteint pas la frontière du noyau de viabilité du système, et on choisit un contrôle qui permet de revenir à l'intérieur sinon.

L'évolution du système peut être influencée par des perturbations non maîtrisables (qui peuvent également être appelées tyches [Aubin, 1997]). Ces perturbations peuvent être vues comme notre ignorance de certains comportements du système, ce qui introduit un écart entre la dynamique modélisée et le comportement réel du système. Le formalisme de la viabilité peut également être étendu aux jeux dynamiques. On recherche alors l'ensemble des états viables, quelle que soit la valeur de la perturbation. Dans ce cas, on parle de *noyaux discriminants*. [Cardaliaguet, 1994] définit les noyaux discriminants comme des intersections de noyau de viabilité.

La théorie de la viabilité traite également du problème d'atteinte d'une cible. Un état *capture* la cible lorsqu'il existe au moins une fonction de contrôle qui permette au système d'atteindre la cible en temps fini, tout en restant à l'intérieur des contraintes de viabilité. L'ensemble des états capturant la cible est appelé *bassin de capture*. A chaque état contenu dans le bassin de capture, on définit son *temps minimal de capture*, qui correspond au temps minimal mis par la trajectoire pour atteindre la cible sans violer les contraintes. Cette fonction correspond à celle obtenue en résolvant les équations Hamilton-Jacobi-Bellman (HJB) en programmation dynamique [Frankowska, 1989].
Le bassin de capture d'un système correspond au noyau de viabilité du système, auquel on a ajouté à la dynamique une dimension représentant le temps qui s'écoule. De la même façon que pour le noyau de viabilité, le bassin de capture permet de définir directement des politiques de contrôle, qui fournissent une trajectoire qui reste toujours dans l'espace des contraintes, tout en atteignant la cible en un temps *fini* ou *prescrit*. Il donne également les politiques qui permettent d'atteindre la cible en un temps *minimal*.

Comme nous l'avons indiqué précédemment, les problèmes de résilience s'apparentent à des problèmes d'atteinte de cible, dans le cadre proposé par Martin [Martin, 2004]. Une différence importante est l'introduction d'une fonction de coût lorsque le système se trouve hors de l'ensemble de contraintes. La résilience est alors évaluée comme l'inverse du coût de restauration des propriétés perdues après une perturbation.

Contrôle optimal

Les problèmes de viabilité et leurs extensions peuvent aussi être résolus dans le cadre du contrôle optimal, notamment en utilisant des méthodes comme la programmation dynamique [Bertsekas, 2005]. La programmation dynamique a été introduite par Bellman [Bellman, 1957] pour résoudre des problèmes d'optimisation en recherche opérationnelle. L'efficacité de cette méthode repose sur le principe d'optimalité de Bellman : « toute politique optimale est composée de sous-politiques optimales ».
Un problème de contrôle optimal est composé d'un ensemble d'*états initiaux*, d'un ensemble de *contrôles* liés à chaque état, d'une *fonction de transition* qui associe à un couple état-contrôle l'état suivant, et une *récompense* (ou un coût) obtenue pour avoir associé un contrôle à un état. Le but de la programmation dynamique est de maximiser la récompense totale d'une trajectoire dans l'espace d'état (ou de minimiser le coût de la trajectoire). La fonction qui associe à chaque état initial la valeur de la récompense associée à la trajectoire qui maximise la récompense est appelée la fonction valeur. Cette fonction est la solution de l'équation Hamilton-Jacobi-Bellman (HJB) liée au problème.

Un problème de viabilité se caractérise donc dans ce cadre par une fonction récompense négative en dehors de l'ensemble de contrainte, et nulle à l'intérieur. En utilisant une telle fonction récompense, [Coquelin *et al.*, 2007] caractérisent le noyau de viabilité d'un système comme l'ensemble des zéros d'une fonction valeur optimale $V^*(x)$. En outre, ils définissent un contrôleur prudent, qui construit une procédure de contrôle qui garde le système à l'intérieur des contraintes de viabilité. Ainsi, l'ensemble des méthodes du contrôle optimal sont disponibles pour résoudre des problèmes de viabilité.

Le problème d'atteinte d'une cible en un *temps minimal* est un problème classique en contrôle optimal. On choisit alors une récompense telle que maximiser cette récompense revient à minimiser le nombre de pas pour atteindre la cible. Un choix possible est une récompense égale à -1 partout, sauf

dans la cible où elle est égale à 0 et à l'extérieur de l'espace des contraintes où elle est égale à $-\infty$. Dans ce cas, la fonction valeur optimale correspond à l'inverse du nombre de pas nécessaires pour atteindre la cible.
Dans un problème de résilience, comme nous l'avons vu, il est nécessaire d'ajouter une fonction de coût des états non désirés (et éventuellement des actions), et de calculer les politiques permettant de minimiser ce coût de restauration. Il s'agit typiquement d'un problème de contrôle optimal.

Pourquoi choisir la théorie de la viabilité ?

Le contrôle optimal semble un cadre plus général que la théorie de la viabilité, qui ne s'attaque qu'à un problème particulier de contrôle optimal. De plus, les problèmes d'atteinte de cible et de calcul de politiques de coût de restauration minimal se posent très naturellement dans le cadre du contrôle optimal. Ces arguments plaident a priori en faveur de ce cadre théorique.

Cependant, cette plus grande généralité du contrôle optimal n'est qu'apparente. En effet, on peut montrer que, sous certaines conditions assez générales, un problème de contrôle optimal peut être traduit en un problème de viabilité [Frankowska, 1989 ; Frankowska, 1993]. Ainsi, les techniques de viabilité peuvent être utilisées pour résoudre un problème de contrôle optimal. Donc l'argument de la généralité n'est pas déterminant.

La raison principale de notre choix pour la théorie de la viabilité est notre volonté d'utiliser des méthodes de discrimination particulières pour résoudre les problèmes de viabilité. Or ces méthodes de discrimination sont beaucoup plus directement utilisables dans le cadre de la théorie de la viabilité. En effet, dans ce cadre, le problème central est d'approcher le noyau de viabilité qui est une variété dans l'espace, frontière entre des points viables et des points non-viables. Le problème de déterminer la frontière de cette variété à partir d'un échantillon de points viables et non-viables est un problème de discrimination typique.

Le cadre du contrôle optimal oriente plutôt vers un problème d'approximation de fonction (la fonction valeur), pour lequel les méthodes que nous souhaitons utiliser sont moins performantes. Il convient de moduler ce propos car certains ont cherché à utiliser des méthodes de discrimination dans un cadre de programmation dynamique [Lagoudakis et Parr, 2003], en approchant directement une fonction qui à un état associe une action. Nous n'avons pas retenu cette approche, car elle se limite à des cas où l'ensemble d'actions est de petite taille, alors que nous avons privilégié un ensemble d'actions continu.

Un pari sur les SVMs

Pour poursuivre la présentation de notre démarche, et notre pari sur une méthode de discrimination particulière, les SVMs, un détour par l'état de l'art sur les algorithmes d'approximation de noyau de viabilité est nécessaire.

Les algorithmes d'approximation de noyau de viabilité

Différents algorithmes ont été développés pour approcher le noyau de viabilité ou le bassin de capture d'un système. Le premier a été introduit par Patrick Saint-Pierre [Saint-Pierre, 1994], qui a développé « l'algorithme de viabilité », basé sur la discrétisation de l'espace d'état, qui fournit une approximation du noyau de viabilité d'un système. Des algorithmes semblables ont été développés pour approcher des bassins de capture [Pujal et Saint-Pierre, 2004 ; Cardaliaguet *et al.*, 1998], et [Cardaliaguet *et al.*, 2000] estiment l'erreur de l'approximation. Ces algorithmes sont très rapides mais le résultat est l'ensemble des points viables (ou qui capturent la cible), ce qui n'est pas très pratique à manipuler. De plus, ils requièrent l'exploration exhaustive des contrôles pour trouver un contrôle viable, et le temps de recherche croît exponentiellement avec la dimension de l'espace des contrôles. Ils sont basés sur un

schéma diffusif, qui entraîne des sur-estimations des résultats réels. L'algorithme de viabilité est repris en détail dans le chapitre 1.

A partir de cet algorithme de viabilité, différents chercheurs ont proposé d'autres algorithmes. Une direction de travail a été de développer des procédures qui limitent l'effet diffusif de l'algorithme. [Bokanowski *et al.*, 2006] ont utilisé un schéma Ultra-Bee pour résoudre des problèmes de viabilité. Le problème est alors vu comme un problème de contrôle optimal, résolu en utilisant une fonction valeur. La méthode montre de bons résultats sur les exemples traités, tout particulièrement grâce à ses propriétés anti-diffusives. L'algorithme est basé sur un schéma qui n'est défini qu'en dimension 2, ce qui limite son utilisation aux problèmes en dimension inférieure ou égale à 2. De plus, la preuve de la convergence de l'approximation vers le vrai noyau de viabilité ou bassin de capture n'est pas faite.
D'autres chercheurs ont proposé des algorithmes qui explorent d'autres pistes pour travailler dans des espaces d'état plus importants. En utilisant une technique de recuit simulé, [Bonneuil, 2006] propose une procédure qui approche le noyau de viabilité ou le bassin de capture d'un système. L'algorithme permet de tester si un point est viable ou capture la cible pour des systèmes en grande dimension. Cependant, la procédure est dépendante de l'horizon de temps : on teste uniquement si un point est viable (ou s'il capture la cible) pendant T pas de temps. Plus le nombre de variables de contrôle augmente, plus l'horizon de temps pouvant être testé est faible. De plus, pour approcher le noyau de viabilité ou le bassin de capture du système, la procédure discrétise une partie du bord de l'espace des contraintes, ce qui revient à gagner uniquement une dimension sur le problème. Dans une autre direction, [Mullon *et al.*, 2004] définissent un algorithme d'approximation de noyau de viabilité lorsque celui-ci est convexe. Cet algorithme leur permet de résoudre des problèmes de dimension 5. L'algorithme approche l'enveloppe convexe du noyau comme l'intersection de demi-espaces (systèmes linéaires). A chaque itération, l'algorithme définit une direction et recherche les demi-espaces dont la frontière est orthogonale à cette direction. Le plus grand demi-espace admissible définit alors une contrainte qui restreint l'ensemble viable. L'algorithme est très rapide et marche avec des contrôles en dimension importante. Cependant, il est limité au cas où le noyau de viabilité est convexe, ce qu'on ne sait pas en général a priori.

[Coquelin *et al.*, 2007] ont proposé une approche alternative pour approcher des noyaux de viabilité en utilisant la fonction valeur d'un problème de programmation dynamique. Ils ont défini le noyau comme l'ensemble des états pour lesquels la fonction valeur est en dessous d'un certain seuil. Mais la valeur de ce seuil n'est pas facile à définir.
[Lhommeau *et al.*, 2007] ont proposé un algorithme d'approximation de bassins de capture basé sur l'analyse par intervalles. L'algorithme comporte deux variantes : la première variante fournit une approximation incluse dans le bassin de capture et la deuxième une approximation qui inclut le bassin de capture. Cependant, la complexité de l'algorithme augmente avec le nombre de variables d'état du système, et requiert l'exploration exhaustive des contrôles.

Les algorithmes décrits dans cette sous-section requièrent la discrétisation de l'espace d'état (sauf l'algorithme proposé dans [Mullon *et al.*, 2004]). Cependant, la taille de la grille explose avec la dimension : c'est la malédiction de la dimensionnalité. Ces algorithmes utilisent également un ensemble discret de contrôles, ce qui limite également leur utilisation à des systèmes avec des contrôles en petites dimensions. Les algorithmes ne sont donc applicables que pour des systèmes avec un espace d'état en petite dimension (3 ou 4) et peu de variables de contrôle.
Dans cette thèse, nous faisons le pari que l'introduction d'une méthode de discrimination particulière, les SVMs, permet de travailler dans des espaces de dimension plus importante.

Un pari : les SVMs

Les algorithmes d'approximation de noyau de viabilité et de bassin de capture souffrent de la malédiction de la dimensionnalité dans l'espace d'état et des contrôles. Dans les différentes applications détaillées au début de ce chapitre, les systèmes sont généralement de dimension élevée ; il est donc indispensable d'améliorer les performances des algorithmes d'approximation pour aider à la détermination de politiques d'action sur ces systèmes. Pour cela, nous introduisons un nouvel algorithme d'approxi-

mation de noyau de viabilité, de bassin de capture et de calcul de valeurs de résilience qui utilise une méthode d'apprentissage statistique particulière : les machines à vecteurs de support. Cet algorithme est basé sur celui de Saint-Pierre, il associe une étiquette $+1$ à un état s'il est viable (ou s'il capture la cible) à l'itération courante et -1 sinon, et construit une fonction de discrimination à partir de l'ensemble des états étiquetés. Nous utilisons les machines à vecteurs de support pour construire une telle fonction de discrimination.

Les machines à vecteurs de support (ou séparateurs à vaste marge, Support Vector Machines en anglais - SVMs) sont une méthode de discrimination qui sépare les données en fonction de leur étiquette. Elles ont été introduites par Vapnik dans les années 1990 [Vapnik, 1995 ; Vapnik, 1998]. Elles permettent la construction d'un *hyperplan optimal* pour calculer la fonction de discrimination entre les catégories d'exemples. La frontière de décision est définie à l'aide d'un sous-ensemble des points initiaux, appelés *vecteurs de support* (Support Vectors en anglais - SVs). Les SVMs identifient quels sont les points SVs, et définissent l'hyperplan optimal comme une combinaison linéaire de ces points. Lorsque les données sont non-linéairement séparables, les SVMs projettent les données dans un *espace déployé* à l'aide d'une *fonction noyau* et construisent l'hyperplan optimal dans cet espace.

Dans cette thèse, nous faisons le pari que les SVMs vont permettre de résoudre des problèmes définis dans des espaces en plus grande dimension. Trois principaux arguments nous laissent penser que cela sera le cas :

- les SVMs permettent de représenter des formes complexes dans des espaces d'état de très grande dimension ;
- la fonction SVM est définie à l'aide d'un nombre de points réduit : les vecteurs de support. Ainsi, même si le nombre de points est très important, seulement une partie d'entre eux est pertinente. Dans notre cas, même si la taille de la grille définie sur l'espace d'état est très importante, seulement la partie la plus « informative » sera utilisée pour définir l'approximation du noyau de viabilité ou du bassin de capture ;
- la fonction SVM fournit une sorte de fonction barrière sur la frontière de l'approximation, qui peut être facilement utilisée pour trouver des contrôles viables et définir des contrôleurs.

Prouver la convergence des algorithmes et les tester sur des cas concrets

L'objectif de la thèse est d'exploiter les propriétés des SVMs pour proposer de nouveaux algorithmes d'approximation de noyau de viabilité et de bassin de capture qui puissent traiter des problèmes en grande dimension, dans l'espace d'état et des contrôles. L'exigence de cette thèse est double : les algorithmes d'approximation doivent être performants et opérationnels et leur convergence doit être prouvée mathématiquement. Pour répondre à ces exigences, nous nous sommes attachés à prouver la convergence de tous les algorithmes proposés, tout en les testant sur de nombreux cas d'application. Les preuves de convergence sont dérivées pour des méthodes de discrimination quelconques, et donc les algorithmes proposés pourraient être adaptés à d'autres méthodes de discrimination.

Dans cette thèse, nous faisons l'hypothèse que nous connaissons la dynamique du système étudié, et que celle-ci est déterministe. Nous ne traitons pas en profondeur le problème des jeux dynamiques et des noyaux discriminants, ils seront uniquement abordés très légèrement dans le chapitre 5.

Organisation du document

Dans une première partie, nous traitons de l'approximation de noyau de viabilité et du contrôleur lourd.

Dans le chapitre 1, nous détaillons les différents concepts de la théorie de la viabilité liés au noyau de viabilité et la procédure de contrôle lourd d'un système. Nous décrivons également l'algorithme de viabilité de Saint-Pierre. Nous proposons ensuite un algorithme d'approximation de noyau de viabilité en utilisant une méthode d'apprentissage, algorithme basé sur celui de Saint-Pierre. Nous donnons les

conditions sur la méthode d'apprentissage qui garantissent que l'algorithme converge vers le noyau de viabilité lorsque la résolution de la grille des points d'apprentissage tend vers 0. Nous proposons ensuite une procédure de contrôle lourd, basée sur le résultat fourni par l'algorithme.
Le chapitre 2 traite de l'approximation de noyau de viabilité en utilisant des SVMs. Nous donnons les principales caractéristiques de l'algorithme ; notamment, il permet d'utiliser des méthodes d'optimisation pour trouver un contrôle viable, et ainsi travailler dans des espaces de contrôle plus importants. Nous montrons également qu'il permet de dériver des politiques de contrôle en utilisant une approximation de la distance entre un point et la frontière du noyau approché. Nous appliquons ensuite la procédure sur un problème de gestion des pêches, en examinant quelles politiques de pêche permettent de garantir la viabilité d'un écosystème marin, système qui comporte 6 variables d'état et 17 variables de contrôle.

La seconde partie est consacrée à l'approximation de bassin de capture et des valeurs de résilience, ainsi qu'à la définition de contrôleurs optimaux.

Dans le chapitre 3, nous détaillons les différents concepts de la théorie de la viabilité liés au bassin de capture et la procédure de contrôle optimal d'un système. Nous proposons ensuite un algorithme d'approximation de bassins de capture en utilisant une méthode d'apprentissage. Nous définissons deux variantes de l'algorithme : la première permet d'obtenir une approximation incluse dans le bassin de capture, et la deuxième une approximation qui inclut le bassin de capture. Nous donnons les conditions sur la procédure d'apprentissage qui garantissent la convergence de l'approximation vers le bassin de capture lorsque la résolution de la grille des points d'apprentissage tend vers 0. Nous définissons enfin une procédure de contrôle optimal.
Le chapitre 4 considère les SVMs pour approcher un bassin de capture. Nous proposons un algorithme qui permet de gagner en temps de calcul lorsque l'on considère un système dynamique auxiliaire. Nous détaillons la procédure lorsque l'on travaille dans l'espace d'état initial et nous proposons ensuite une procédure de contrôle optimal. Nous illustrons l'algorithme sur quelques exemples simples d'application.
Le chapitre 5 traite de l'approximation des valeurs de résilience. Nous montrons tout d'abord que le problème peut se ramener à un problème de calcul de bassin de capture. Nous proposons un algorithme, basé sur les SVMs, qui permet de calculer les valeurs de résilience d'un système dans l'espace d'état initial. Nous appliquons ensuite la procédure sur un problème d'eutrophication des lacs en 3 dimensions.

Dans une troisième partie, nous introduisons une procédure d'apprentissage actif.

Le chapitre 6 considère les techniques d'apprentissage actif comme un premier pas pour vaincre la malédiction de la dimensionnalité liée aux algorithmes proposés dans cette thèse. Nous proposons une procédure de sélection des états les plus « informatifs » pour définir la fonction SVM, et ainsi gagner en espace mémoire et complexité calculatoire, tout en gardant une approximation précise du noyau. Nous illustrons ensuite la procédure sur un problème de conduite d'un vélo sur un circuit, système défini par 6 variables d'état.

La plupart des travaux présentés ici ont été publiés ou soumis à des journaux, ou publiés dans des actes de conférence. Tous sont le résultat de collaborations étroites, mais l'ordre des auteurs reflète l'importance des contributions, le premier auteur étant le contributeur principal :

- les chapitres 1 et 2, pour la partie algorithme d'approximation de noyau de viabilité en utilisant une méthode d'apprentissage en général, et les SVMs en particulier, sont basés sur [Deffuant *et al.*, 2007] et [Deffuant *et al.*, 2005] ;
- les chapitres 3 et 4 sur l'algorithme d'approximation des bassins de capture dans l'espace d'état initial sont basés sur [Chapel et Deffuant, Subm] ;
- l'application à un modèle d'écosystème marin (chapitre 2) a été présentée à [Chapel *et al.*, 2005] et publiée dans [Chapel *et al.*, Acc] ;
- l'application de l'algorithme pour le calcul des valeurs de résilience dans le cas de l'eutrophisation des lacs (chapitre 5) a été présentée dans [Chapel *et al.*, 2007] ;
- le chapitre 6 présentant un algorithme d'approximation de noyau de viabilité en utilisant l'apprentissage actif est basé sur [Chapel et Deffuant, 2007].

Première partie

Approcher des noyaux de viabilité

Chapitre 1

Approcher des noyaux de viabilité en utilisant une méthode d'apprentissage

Dans ce chapitre, nous proposons un algorithme d'approximation de noyau de viabilité en utilisant une méthode d'apprentissage et nous prouvons la convergence de cet algorithme vers le vrai noyau du système. Nous proposons également un contrôleur lourd de viabilité, basé sur l'approximation obtenue en utilisant cet algorithme.

Dans la section 1.1, nous définissons le noyau de viabilité et nous décrivons comment ce noyau peut être utilisé pour contrôler un système. Nous définissons notamment la procédure de contrôle lourde. Nous décrivons ensuite l'algorithme de viabilité de Saint-Pierre, qui permet d'approcher des noyaux de viabilité en utilisant des approximations discrètes dans le temps et dans l'espace. Dans la section 1.2, nous proposons un algorithme d'approximation de noyau de viabilité, basé sur l'algorithme de viabilité de Saint-Pierre et qui utilise une méthode d'apprentissage, et établissons les conditions mathématiques que la procédure d'apprentissage doit respecter afin de garantir la convergence de l'approximation vers le vrai noyau de viabilité. Pour terminer, nous adaptons la procédure de contrôle lourd aux approximations obtenues.

Sommaire

1.1 Viabilité

1.1.1 Définitions

La théorie de la viabilité considère un système dynamique défini par son état $x(t)$ au cours du temps $t \in \mathbb{R}^+$ et suppose que son évolution peut être influencée par un contrôle $u(t)$:

$$x'(t) \in F(x(t)) = \begin{cases} \varphi(x(t), u(t)) & \text{(action)} \\ u(t) \in U(x(t)) & \text{(rétroaction),} \end{cases} \tag{1.1}$$

où $x(t)$ est un vecteur $\in \mathcal{X} \subset \mathbb{R}^d$ et $F : \mathcal{X} \rightsquigarrow \mathcal{X}$ est une correspondance. L'ensemble des contrôles disponibles dépend de l'état du système, $u(t)$ est un vecteur $\in \mathbb{R}^q$. A partir d'un point x, il peut exister plusieurs évolutions possibles, en fonction des différents contrôles choisis au cours du temps $t \rightarrow u(t)$. Une solution du système (1.1) est une évolution $t \rightarrow x(t)$ telles que les conditions de (1.1) sont vérifiées presque partout.
Aubin [Aubin, 1991] définit le concept d'*état viable* dans un ensemble de contraintes de viabilité $K \subset \mathbb{R}^d$ comme un état x_0 pour lequel il existe au moins une trajectoire $x(t)$, satisfaisant (1.1), qui reste dans K indéfiniment :

$$\begin{cases} x(0) = x_0 \\ \forall t \geq 0, x(t) \in K. \end{cases} \tag{1.2}$$

Dans ce cas, $x(t)$ est appelé *évolution viable*.
Le problème de viabilité est donc de déterminer une fonction de contrôle $t \rightarrow u(t)$ qui permet de garder un état du système indéfiniment dans K. L'ensemble de tous les états viables pour la correspondance F est appelé le *noyau de viabilité* de K (figure 1.1) :

$$Viab_F(K) = \{x_0 \in K | \exists x(t) \text{ vérifiant } (1.1), x(0) = x_0 \text{ tel que } \forall t \geq 0,\, x(t) \in K\}\,. \tag{1.3}$$

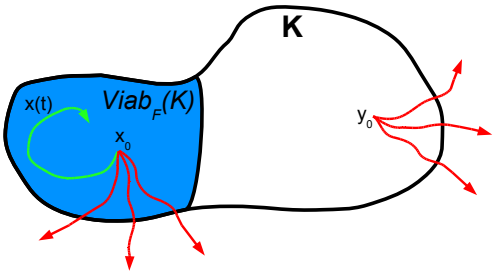


FIG. 1.1 – Exemple de noyau de viabilité (représenté en bleu). L'espace des contraintes K est délimité par la ligne continue. x_0 et y_0 sont deux états initiaux, et quelques évolutions possibles sont représentées par les flèches. x_0 est un point viable car il existe au moins une évolution qui lui permet de rester dans K (évolution viable en vert), tandis que y_0 est non viable : il n'existe aucune fonction de contrôle qui permette au système de rester viable (évolutions non viables en rouge).

Aubin [Aubin, 1991] a prouvé que le noyau de viabilité de K est le plus grand sous-ensemble viable de K pour le système contrôlé (1.1). Le noyau est composé de l'ensemble des états pour lesquels il existe au moins une trajectoire viable dans K, et toutes les évolutions partant de $K \backslash Viab_F(K)$ violent les contraintes de viabilité en temps fini. La frontière du noyau de viabilité agit comme une barrière : elle laisse sortir des évolutions mais n'en laisse rentrer aucune.
Les noyaux de viabilité ont des propriétés intéressantes, comme par exemple la semi-perméabilité [Quincampoix, 1991] : sous certaines conditions, pour $x_0 \in \partial Viab_F(K) \backslash \partial K$ ($\partial Viab_F(K)$ représente la

frontière du noyau et ∂K la frontière de K), il existe une solution viable dans K partant de x_0 qui reste à la frontière de $Viab_F(K)$ tant que la trajectoire n'atteint pas ∂K.

La principale tâche pour résoudre un problème de viabilité est de déterminer le noyau de viabilité d'un système. Les théorèmes de viabilité [Aubin, 1991] permettent de déterminer les états viables sans considérer l'exploration des suites de contrôles sur l'ensemble du temps. Ces théorèmes sont valides pour une large classe de systèmes appelés *systèmes de Marchaud*[1]

1.1.2 Contrôleur de viabilité

Le noyau de viabilité d'un système est déterminant pour définir des politiques d'actions viables. A partir d'un point appartenant à $Viab_F(K)$, on sait qu'il existe une suite de contrôles qui va permettre au système de rester indéfiniment dans K. La question est : comment choisir ces contrôles ?

La règle la plus simple a été introduite par [Aubin et Frankowska, 1985] et est appelée *contrôleur lourd*. La procédure découle du principe d'inertie : « les contrôles sont gardés constants tant que la viabilité du système n'est pas menacée ». Ainsi, lorsque l'état du système se trouve à l'intérieur du noyau, on peut conserver le même contrôle. A un moment donné, avec u constant, le système peut atteindre la frontière du noyau au pas de temps suivant. Cet événement est appelé *période de crise* : pour survivre, le système doit trouver un autre contrôle qui le force à revenir à l'intérieur du noyau (par définition, on sait qu'un tel contrôle existe). Une solution réside dans le choix du contrôle qui a la plus petite vitesse, le plus « paresseux » (parmi ceux qui produisent une trajectoire viable). L'évolution viable correspondante sera alors qualifiée d'*évolution lourde*.

En général, il n'existe pas de formule explicite pour déterminer le noyau de viabilité d'un système. Les théorèmes de viabilité sont la base de l'algorithme développé par Saint-Pierre [Saint-Pierre, 1994] (voir aussi [Quincampoix et Saint-Pierre, 1995]). Dans la sous-section suivante, nous en décrivons rapidement le principe car les procédures proposées dans cette thèse sont basées sur cet algorithme.

1.1.3 Algorithme de viabilité de Saint-Pierre

L'algorithme proposé par Saint-Pierre [Saint-Pierre, 1994] approche le noyau de viabilité $Viab_F(K)$ du système en utilisant des approximations discrètes du système dynamique (1.1) et de l'espace des contraintes K. Nous détaillons ici le cas où la correspondance F est μ-Lipschitz[2].

Soit $G : \mathcal{X} \rightsquigarrow \mathcal{X}$ la correspondance discrète définie par $G = 1 + F.dt$:

$$\begin{cases} x^{n+1} \in G(x^n), \text{ pour tout } n \geq 0 \\ x^0 = x_0, \end{cases} \tag{1.4}$$

et le noyau de viabilité discret associé est noté $Viab_G(K)$. $Viab_G(K)$ coïncide avec le plus grand ensemble de points initiaux x_0 tel qu'une séquence $(x^n)_n$, solution de (1.4), reste toujours dans K :

$$\forall n \geq 0,\ x^n \in K. \tag{1.5}$$

Saint-Pierre considère la séquence $(K^n)_n$, avec $K^0 = K$, définie comme suit :

$$K^{n+1} = \{x \in K^n \text{ tels que } G(x) \cap K^n \neq \emptyset\}, \tag{1.6}$$

et montre que $Viab_G(K) = \bigcap_{n=0}^{+\infty} K^n$ et qu'il existe un entier p tel que $Viab_G(K) = K^p$.

[1] Une correspondance $F : \mathcal{X} \rightsquigarrow \mathcal{X}$ est Marchaud si F est semi-continue supérieurement, à valeurs non vides, convexes, compactes et à croissance linéaire.

[2] F est μ-Lipschitz s'il existe une constante μ telle que $\forall x, y \in \mathcal{X}, F(x) \subset F(y) + \boldsymbol{B}(0, \mu \|x - y\|)$, avec $\boldsymbol{B}(c, r)$ une boule de centre c et de rayon r.

Il opère ensuite une discrétisation de l'espace d'état. Définissons une grille de points X_h telle que :

$$\forall x \in \mathcal{X}, \exists x_h \in X_h \text{ tel que } \|x - x_h\| \leq \beta(h), \tag{1.7}$$

avec

$$\lim_{h \to 0} \beta(h) = 0, \tag{1.8}$$

et la grille K_h associée :

$$K_h = (K + \beta(h)\boldsymbol{B}) \cap X_h, \tag{1.9}$$

avec $\boldsymbol{B}$ une boule de centre 0 et de rayon 1. Il introduit la correspondance $G_h : X_h \rightsquigarrow X_h$, et le système dynamique associé :

$$\begin{cases} x_h^{n+1} \in G_h(x_h^n), \text{ pour tout } n \geq 0 \\ x^0 = x_0. \end{cases} \tag{1.10}$$

Saint-Pierre considère ensuite la séquence $(K_h^n)_n$, avec $K_h^0 = K_h$:

$$K_h^{n+1} = \{x \in K_h^n \text{ tels que } G_h(x) \cap K_h^n \neq \emptyset\}. \tag{1.11}$$

Il obtient alors $Viab_{G_h}(K) = \bigcap_{n=0}^{+\infty} K_h^n$, et il existe un entier p tel que $Viab_{G_h}(K) = K_h^p$.
Cette séquence fournit une méthode pour approcher le noyau de viabilité en utilisant des approximations discrètes : à partir de K_h^n, l'algorithme supprime à l'étape $n+1$ les points de la grille qui n'ont pas de successeurs dans K_h^n.

Saint-Pierre [Saint-Pierre, 1994] montre que, lorsque F est μ-Lipschitz, le noyau de viabilité discret tend vers le noyau du système initial lorsque le pas de la grille h tend vers 0.

L'algorithme est très rapide, mais à chaque étape, l'approximation du noyau de viabilité courant K_h^n est définie comme un ensemble de points, ce qui n'est pas très pratique à manipuler. De plus, la taille de la grille augmente exponentiellement avec la dimension de l'espace d'état du système et l'algorithme requiert le test exhaustif de tous les contrôles possibles (ou d'un ensemble de contrôles discrétisés dans le cas où les contrôles sont continus). Ces deux aspects limitent l'utilisation pratique de l'algorithme à des systèmes en petite dimension dans l'espace d'état et l'espace des contrôles. D'autre part, le schéma numérique utilisé est diffusif, ce qui entraîne des surestimations des noyaux réels.

1.2 Algorithme d'approximation de noyau de viabilité en utilisant une méthode d'apprentissage

Dans cette section, nous définissons tout d'abord la notion d'apprentissage statistique. Nous proposons ensuite un algorithme d'approximation de noyau de viabilité en utilisant une technique d'apprentissage, basé sur l'algorithme de viabilité, et nous donnons les preuves de convergence de cet algorithme vers le vrai noyau de viabilité.

1.2.1 Apprentissage statistique

L'apprentissage statistique regroupe l'ensemble des méthodes et algorithmes qui permettent d'extraire de l'information à partir d'une base de N exemples $x \in \mathcal{X} \subset \mathbb{R}^d$. Les algorithmes d'apprentissage peuvent se catégoriser en deux principales familles :

– apprentissage supervisé : un expert étiquette les données. Le but est alors d'approcher une fonction qui affecte la bonne étiquette à ces exemples.
– apprentissage non supervisé : aucun expert n'est disponible. Le but est de découvrir la structure des données. On parle de méthode de classification.

On se focalise ici sur l'apprentissage supervisé.

Dans l'apprentissage supervisé, on associe à chaque exemple une étiquette $y \in \mathcal{Y} \subset \mathbb{R}$ qui permet de caractériser x. A partir d'une base d'apprentissage $\mathcal{S} = \{(x_i, y_i)\}_{i=1}^{N}$ qui regroupe l'ensemble des exemples connus, on définit un apprenti $\mathcal{A} : \mathcal{S} \rightarrow l(x)$. La fonction $l(x)$ associe à un exemple x une étiquette $y : l : \mathcal{X} \rightarrow \mathcal{Y}$. La théorie de l'apprentissage [Vapnik, 1995] montre que le choix de la fonction $l(x)$ doit être basé sur des critères de qualité de prévision et de complexité. Lorsque l'étiquette que l'on cherche à associer à un exemple est une valeur dans un ensemble de réels, on parle de méthode de régression ; lorsque les exemples sont placés dans des groupes en fonction de certaines de leurs caractéristiques, et que l'étiquette est donc discrète, on parle de méthode de discrimination.

Parmi les méthodes de discrimination, on peut citer :
– les k-plus proches voisins ($k - ppv$) : technique qui consiste à rechercher les étiquettes d'un certain nombre de k plus proches voisins d'un point x, et qui lui attribue l'étiquette majoritaire parmi les k points lui ressemblant le plus dans la base d'apprentissage. Ici, aucun apprentissage n'a vraiment lieu et aucun modèle n'est appris à partir de la base d'apprentissage.
– les machines à vecteurs de support (SVMs) [Vapnik, 1995 ; Vapnik, 1998] : technique de construction d'un hyperplan optimal séparant n classes et qui fournit une fonction continue de discrimination. On projette les données dans un espace « déployé » de grande dimension afin de définir une séparation linéaire dans cet espace, avant de revenir dans l'espace initial.

Dans le cadre de l'apprentissage de noyau de viabilité, on parle de discrimination binaire $y \in \{-1; +1\}$. Les exemples de la base d'apprentissage sont les points de la grille $x_h \in K_h$, les étiquettes y sont notées $+1$ si le point est viable à l'itération courante et -1 sinon. La base d'apprentissage est composée de l'ensemble des points de la grille, associés à leur étiquette. La procédure l permet d'associer à n'importe quel état $x \in K$ une étiquette $y : l : K \rightarrow \{-1; +1\}$.

1.2.2 Algorithme d'approximation de noyau de viabilité et contrôleur lourd

L'algorithme proposé dans ce chapitre est basé sur l'algorithme de Saint-Pierre. On utilise une méthode d'apprentissage pour généraliser l'ensemble des points qui constituent l'approximation courante du noyau de viabilité.

1.2.2.1 Notations

On discrétise le système dynamique dans le temps (avec la correspondance G), mais pas dans l'espace (comme cela était fait avec G_h). On considère un intervalle de temps dt, et on définit la correspondance $G : X \rightsquigarrow X$:

$$G(x) = \{\vec{x} + \varphi(x, u)dt \text{ pour } u \in U(x)\}. \tag{1.12}$$

On suppose que G est μ-Lipschitz à images fermées. On définit l'inclusion différentielle discrète :

$$\begin{cases} x^{n+1} \in G(x^n) \\ x^0 = x_0. \end{cases} \tag{1.13}$$

Soit K un ensemble compact de $\mathcal{X}$, on recherche l'approximation du noyau de viabilité $Viab_G(K)$ de K pour la dynamique discrète (1.13). Les théorèmes de viabilité montrent que $Viab_G(K)$ est le plus grand sous-ensemble E de K tel que :

$$\forall x \in E, G(x) \cap E \neq \emptyset. \tag{1.14}$$

On définit une grille K_h comme un ensemble fini d'éléments de K telle que :

$$\forall x \in K, \exists x_h \in K_h \text{ tel que } \|x - x_h\| \leq \beta(h), \tag{1.15}$$

et $\beta(h) \to 0$ lorsque $h \to 0$. Une telle grille existe car K est compact.

A chaque itération n, on définit un ensemble discret $K_h^n \subset K_h^{n-1} \subset K_h$, et un ensemble continu L_h^n, qui est une généralisation de cet ensemble discret, et qui constitue l'approximation courante du noyau de viabilité. On utilise également les notations suivantes :

- $d(E, F) = \inf\{d(e, f)/(e, f) \in (E, F)\}$ est la distance entre deux ensembles E et F ;
- $E \backslash F$ est l'ensemble complémentaire de F dans E (on suppose que $F \subset E$) ;
- $\boldsymbol{B}$ est la boule de centre 0 et de rayon 1.

1.2.2.2 Algorithme d'approximation

Au départ, on considère que tous les points $x_h \in K_h$ sont viables. A l'itération $n+1$, l'algorithme supprime les points qui sortent « franchement » de la généralisation de l'approximation du noyau de viabilité à l'itération n (L_h^n). On définit ensuite un apprenti $\mathcal{A}$ sur une base d'apprentissage, constituée des points x_h de la grille K_h étiquetés -1 si le point sort franchement de L_h^n, et $+1$ sinon. On obtient alors la fonction de discrimination l^{n+1}, qui définit le noyau de viabilité courant :

$$L(K_h^{n+1}) = \left\{x \in K \text{ tels que } l^{n+1}(x) = +1\right\}. \tag{1.16}$$

Lorsque les états contenus dans le noyau courant sont tous viables, l'algorithme s'arrête et le noyau courant est alors une approximation du noyau de viabilité.

Procédure 1.1 (Algorithme d'approximation d'un noyau de viabilité)
On définit itérativement les ensembles L_h^n tels que :

$$\begin{array}{l} K_h^0 = K_h \\ L_h^0 = K \\ K_h^{n+1} = \{x_h \in K_h^n \text{ tels que } d(G(x_h), L_h^n) \leq \mu\beta(h)\} \\ L_h^{n+1} = L(K_h^{n+1}) \end{array} \tag{1.17}$$

jusqu'à ce que $K_h^{n+1} = K_h^n = K_h^p$.
Si la procédure de discrimination respecte les conditions suivantes :

$$\forall x \in L_h^n, d(x, K_h^n) \leq \lambda\beta(h), \tag{1.18}$$

$$\forall x \in K \backslash L_h^n, d(x, K_h \backslash K_h^n) \leq \beta(h), \tag{1.19}$$

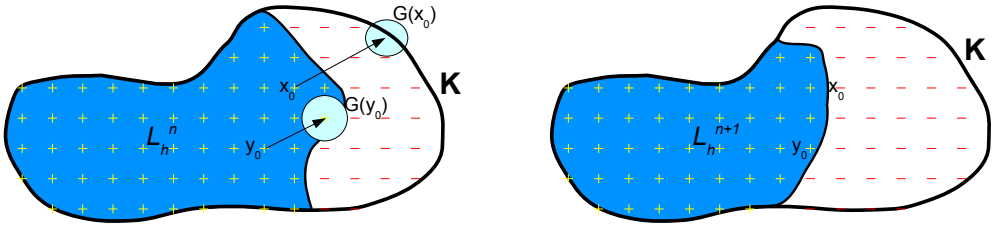
alors

$$L_h^p \to Viab_G(K) \text{ lorsque } h \to 0. \tag{1.20}$$

La figure 1.2 donne une illustration du passage de l'itération n à l'itération $n+1$.

1.2.2.3 Preuve de la convergence de l'algorithme

La convergence de l'approximation obtenue avec l'algorithme 1.1 vers le vrai noyau de viabilité, lorsque la résolution de la grille h tend vers 0, est garantie s'il existe un réel $\lambda \geq 1$ tel qu'à chaque itération n, l'approximation L_h^n satisfait les conditions (1.18) et (1.19). Ces deux conditions signifient respectivement que chaque point de L_h^n doit être proche d'un point de K_h avec une étiquette positive, et que chaque point de $K \backslash L_h^n$ doit être proche d'un point de K_h avec une étiquette négative. La seconde condition relative aux points avec une étiquette négative est plus stricte car si L_h^n est inclus



(a) (b)

FIG. 1.2 – Exemple de passage de l'itération n (figure (a)) à l'itération $n+1$ (figure (b)). Les approximations du noyau de viabilité courant L_h^n et L_h^{n+1} sont représentées en bleu. L'espace des contraintes est délimité par la ligne continue. Les ensembles K_h^n et K_h^{n+1} sont représentés par les points « + » en jaune et les ensembles $K_h \backslash K_h^n$ et $K_h \backslash K_h^{n+1}$ par les points « - » en rouge. La grille K_h est composée des points K_h^n et $K_h \backslash K_h^n$ sur la figure (a) et des points K_h^{n+1} et $K_h \backslash K_h^{n+1}$ sur la figure (b). Les ensembles L_h^n et L_h^{n+1} sont des généralisations des ensembles K_h^n et K_h^{n+1}. A l'itération n, on teste pour tous les points $x_h \in K_h^n$ si $d(G(x_h), L_h^n) \leq \mu\beta(h)$. Si oui, alors $x_h \in K_h^{n+1}$ (point y_0 par exemple sur la figure), et sinon, $x_h \notin K_h^{n+1}$ (point x_0 par exemple).

dans $Viab(K)$, l'erreur ne peut pas être fixée à l'itération suivante (alors qu'elle peut être fixée lorsque l'erreur se situe de l'autre coté).

La preuve de la convergence comporte quatre étapes :

1. **L'algorithme s'arrête après un nombre fini d'itérations p.**
   A chaque itération, on a $K_h^{n+1} \subset K_h^n$ par construction. Or K_h est fini, on obtient donc après un nombre fini d'itérations p : $K_h^{p+1} = K_h^p$.
2. **Pour chaque itération n, le noyau de viabilité de K pour G est inclus dans L_h^n.**
   On montre ici que si un point x n'appartient pas à l'approximation courante du noyau L_h^{n+1}, alors il est non-viable.
   On a $Viab_G(K) \subset K = L_h^0$. Supposons que $Viab_G(K) \subset L_h^n$ et considérons $x \notin L_h^{n+1}$.
   – Si $x \notin L_h^n$ alors x est non-viable par hypothèse.
   – Si $x \in L_h^n$ alors, à cause de la condition (1.19), on a :

   $$\exists x_h \in K_h \backslash K_h^{n+1} \text{ tel que } \|x - x_h\| \leq \beta(h). \tag{1.21}$$

   De plus, par construction de l'algorithme, on a :

   $$x_h \in K_h \backslash K_h^{n+1} \Rightarrow d(G(x_h), L_h^n) > \mu\beta(h). \tag{1.22}$$

   Or, G est μ-Lipschitz :

   $$G(x) \subset (G(x_h) + \mu\beta(h)\boldsymbol{B}). \tag{1.23}$$

   On a donc $G(x) \subset (K \backslash L_h^n)$. Par hypothèse, tous les points de $K \backslash L_h^n$ sont non-viables, donc tous les successeurs de x sont non-viables, ce qui implique que x est non-viable.
   Par conséquent, $x \notin L_h^{n+1} \Rightarrow x$ est non viable, donc $Viab_G(K) \subset L_h^{n+1}$.
3. **Le résultat de l'algorithme L_h^p est inclus dans le noyau de viabilité de K pour $G + \mu(1 + \lambda)\beta(h)\boldsymbol{B}$.**
   Posons $G_h : X \rightsquigarrow X$ tel que $G_h(x) = G(x) + \mu(1+\lambda)\beta(h)\boldsymbol{B}$. G_h est à images fermées. Considérons un point $x \in L_h^p$. A cause de la condition (1.18) :

   $$\exists x_h \in K_h^p \text{ tel que } \|x - x_h\| \leq \lambda\beta(h). \tag{1.24}$$

Comme $K_h^{p+1} = K_h^p$ avec p le nombre d'itération où s'arrête l'algorithme, $x_h \in K_h^{p+1}$ et, par définition de l'algorithme :

$$x_h \in K_h^{p+1} \Rightarrow d(G(x_h), L_h^p) \leq \mu\beta(h). \tag{1.25}$$

Parce que G est μ-Lipschitz, et en utilisant l'inégalité triangulaire,

$$d(G(x), L_h^p) \leq \mu(1+\lambda)\beta(h). \tag{1.26}$$

Cela implique que $G_h(x) \cap L_h^p \neq 0$. Par conséquent, à partir de n'importe quel point $x \in L_h^p$, il existe une trajectoire qui reste indéfiniment dans L_h^p. On a donc $L_h^p \subset Viab_{G_h}(K)$.

4. **Conclusion.**
Puisque $\text{Lim}_{h\to 0} Viab_{G_h}(K) = Viab_G(K)$, alors

$$\text{Lim}_{h\to 0} L_h^p = Viab_G(K), \tag{1.27}$$

la limite étant prise ici au sens de Painlevé-Kuratowski.

1.3 Contrôleur lourd de viabilité

L'idée d'une procédure de contrôle lourd vient d'Aubin [Aubin, 1991]. Le principe est d'appliquer un contrôle constant jusqu'à ce que le système sorte du noyau de viabilité au pas de temps suivant, et d'appliquer ensuite n'importe quel contrôle qui permette à la trajectoire de rester dans le noyau de viabilité (par définition, au moins un tel contrôle existe). Le contrôleur de viabilité, défini dans le cas où l'on a une approximation avec une méthode d'apprentissage, adapte la procédure du contrôleur lourd en ajoutant une distance de sécurité à la frontière de l'approximation du noyau. En effet, on sait que le vrai noyau de viabilité est inclus dans l'approximation. Il faut donc choisir une distance de sécurité qui permette de rester toujours dans le vrai noyau de viabilité, et non pas seulement dans l'approximation. De plus, au lieu de choisir le premier contrôle qui permette à la trajectoire de rester dans le noyau, on choisit celle qui maximise la distance entre le point suivant et la frontière de l'approximation.

On note ∂L_h^p la frontière de l'approximation du noyau de viabilité L_h^p. Afin d'introduire la notion de distance de sécurité, on définit le paramètre $\Delta \in \mathbb{R}^+$. On introduit la fonction de la distance algébrique d'un point à la frontière $d(x, \partial L_h^p)$ telle que la distance est positive si $x \in L_h^p$ et négative sinon. L'algorithme 1.2 associe un contrôle u_{n+1} à la $(n+1)^{\text{ème}}$ itération :

Procédure 1.2 (Contrôleur lourd de viabilité)
Considérant un état initial x_0 tel que $d(x_0, \partial L_h^p) > \Delta$ et un contrôle initial $u_0 \in U(x_0)$ choisi aléatoirement, la procédure associe un contrôle u_{n+1} à l'itération $n+1$ comme suit :
– *si $d((x_n + \varphi(x_n, u_n)dt), \partial L_h^p) > \Delta$, on garde le même contrôle $u_{n+1} = u_n$;*
– *sinon, $u_{n+1} = \arg\max_{u\in U(x)} \{d((x_n + \varphi(x_n, u)dt), \partial L_h^p)\}$.*

Si les contrôles adéquats sont trouvés par cet algorithme, la procédure de contrôle garantit de garder un point $x \in A_\Delta$ indéfiniment dans A_Δ. Les contrôles adéquats existent s'il existe $\epsilon > 0$ tel que, pour chaque point x à la frontière de A_Δ :
– il existe un contrôle tel que $d(x + \varphi(x,u)dt, \partial Viab(K)) > d(x + \epsilon, \partial Viab(K))$,
– $d\left(\arg\max_{y\in G(x)} d(y, \partial Viab(K)), \arg\max_{y\in G(x)} d(y, \partial L_h^p)\right) \leq \epsilon$,
avec $\partial Viab(K))$ la frontière du vrai noyau de viabilité. La première condition signifie que les meilleures trajectoires partant de points situés à l'intérieur du noyau de viabilité ne sont pas tangentes à la frontière, mais permettent au système de revenir légèrement à l'intérieur du noyau. De plus, cela requiert que l'ensemble A_Δ soit strictement inclus dans $Viab(K)$. La seconde condition signifie que les gradients de la fonction ∂L_h^p et ceux de $\partial Viab(K)$ sont semblables.

En pratique, la valeur de Δ qui respecte ces conditions peut être approchée expérimentalement, en testant les valeurs minimales de $d(x, \partial L_h^p)$ pour lesquelles il existe un contrôle qui permet d'augmenter $d(x, \partial L_h^p)$. Cependant, des examens plus précis sont nécessaires ici pour pouvoir garantir que la trajectoire reste indéfiniment dans A_Δ.

Chapitre 2

Apprendre un noyau de viabilité en utilisant des SVMs

Dans ce chapitre, nous proposons un algorithme d'approximation de noyau de viabilité qui utilise des SVMs. Nous donnons les principales caractéristiques de l'algorithme et montrons qu'il permet de définir facilement des procédures de contrôle. Nous illustrons ensuite l'algorithme sur quelques exemples d'application.

Dans la section 2.1, nous définissons les SVMs. Nous montrons notamment qu'elles ont une propriété intéressante : la parcimonie. Nous considérons ensuite les SVMs comme méthode d'apprentissage pour approcher un noyau de viabilité dans la section 2.2. Nous étudions les conditions d'application de l'algorithme, puis nous montrons comment les SVMs permettent d'utiliser une méthode d'optimisation pour trouver un contrôle viable, et éviter ainsi la recherche exhaustive de toutes les valeurs des contrôles. Nous montrons également que la procédure peut être étendue à plusieurs pas de temps. Nous adaptons ensuite le contrôleur lourd à l'approximation obtenue (section 2.3). Nous illustrons tout d'abord l'algorithme sur quelques exemples simples d'application, puis sur un modèle d'écosystème marin en 6 dimensions pour l'espace d'état, et 17 dimensions pour l'espace des contrôles (section 2.4).

Sommaire

2.1 Machines à vecteurs de support pour la discrimination

Le but général des méthodes de discrimination est de construire une fonction qui permette de discriminer des exemples, en fonction de leur classe connue a priori. Il existe de nombreux algorithmes de discrimination, les SVMs [Vapnik, 1995 ; Vapnik, 1998] en sont une méthode particulière qui montre de bonnes performances dans la résolution de problèmes variés tels que la reconnaissance de formes [Burges, 1998] ou la catégorisation de textes [Joachims, 1998]. Montrer de bonnes performances signifie ici :

- avoir de bonnes capacités de *généralisation* : une fois qu'on a appris une SVM sur un ensemble d'apprentissage, la fonction permet de classer très souvent correctement de nouveaux exemples (qui n'ont pas servi à construire la fonction SVM) ;
- la fonction de discrimination ne doit pas être trop *complexe*. La *capacité* est une mesure de la complexité de la fonction.

Les SVMs sont populaires car elles sont basées sur de solides fondements mathématiques : la théorie statistique de l'apprentissage de Vapnik [Vapnik, 1995]. La dimension de Vapnik-Chervonenkis (dimension VC) permet d'exprimer le lien entre le risque empirique (erreur de classement sur les données d'apprentissage) d'une fonction de discrimination et son risque réel (erreur de classement sur les données qui n'ont pas servi à construire la fonction). Cette relation exprime un compromis entre l'erreur empirique et la complexité : une faible dimension VC signifie que la fonction est capable de bien apprendre les données, tout en étant assez régulière et simple pour les généraliser. Dans le cas des SVMs, les fonctions produites discriminent bien les données (maximisent la marge) et sont simples (parcimonieuses).
Dans cette section, nous rappelons brièvement le principe des SVMs dans le cas binaire. Pour plus de détails, le lecteur pourra se référer à [Cristianini et Shawe-Taylor, 2000], [Scholkopf et Smola, 2002] ou [Muller *et al.*, 2001].

2.1.1 Maximisation de la marge

Considérons une base d'exemples $\mathcal{S} = \{(x_i, y_i)\}_{i=1}^{N}$ où x_i est un vecteur dans un espace $\mathcal{X} \in \mathbb{R}^d$ et $y_i \in \{-1, +1\}$ est l'étiquette associée à l'exemple x_i. Dans le cas le plus simple, lorsque les données sont linéairement séparables, on construit un hyperplan qui sépare les exemples positifs des exemples négatifs :

$$f(x) = w.x + b = 0, \tag{2.1}$$

où $w \in \mathbb{R}^N$, $b \in \mathbb{R}$ et « . » signifiant produit scalaire. La fonction de décision est alors de la forme $signe(f(x))$. Vapnik [Vapnik, 1995] a proposé un algorithme d'apprentissage qui discrimine les exemples en utilisant un hyperplan pour les problèmes qui sont linéairement séparables (c'est-à-dire sans faire aucune erreur de classement). Cet algorithme est basé sur le fait que, parmi tous les hyperplans possibles, il existe un unique hyperplan optimal, qui maximise la distance minimale (marge γ) entre les exemples et l'hyperplan (voir figure 2.1). Les points les plus proches de la marge sont appelés vecteurs de support (SVs). L'hyperplan est alors solution de :

$$\text{maximiser}_{w,b} \min \{\|x_i - x\| \, | w.x + b = 0, i = 1, ..., N\}. \tag{2.2}$$

Pour construire cet hyperplan optimal, on doit donc résoudre l'équation suivante :

$$\begin{array}{l} \text{minimiser}_{w,b} \frac{1}{2} \|w\|^2 \\ \text{sous contrainte } y_i(w.x_i + b) \geq 1, \text{ pour tout } i = 1, ..., N. \end{array} \tag{2.3}$$

Cependant, dans la grande majorité des cas, les données ne sont pas linéairement séparables. On peut alors garder l'idée d'un modèle linéaire simple, tout en obtenant des fonctions de décision non-linéaires

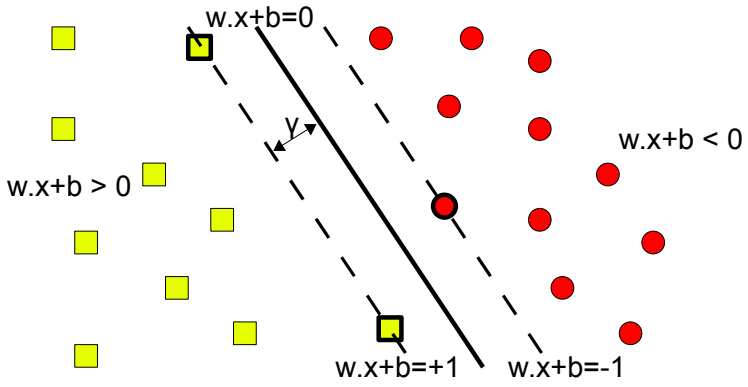


FIG. 2.1 – Hyperplan séparateur et marge. L'hyperplan séparateur est défini par la fonction $w.x + b = 0$ et la fonction de décision est $signe(w.x + b)$. La marge γ est la distance minimale des points à l'hyperplan. Les points situés sur les lignes pointillées sont les SVs.

en utilisant « l'astuce du noyau » [Boser *et al.*, 1992]. Les exemples x_i sont projetés dans un ensemble de redescription $\mathcal{F}$ de grande dimension à l'aide d'une fonction Φ :

$$\begin{array}{rccc} \Phi : & \mathcal{X} & \rightarrow & \mathcal{F} \\ & x & \rightarrow & \Phi(x). \end{array} \tag{2.4}$$

Le point intéressant est qu'il n'est pas nécessaire de définir explicitement la fonction Φ : le noyau k, qui définit un produit scalaire entre deux projections dans l'espace $\mathcal{F}$ ($k(x, y) = \Phi(x).\Phi(y)$), satisfaisant la condition de Mercer[1], est suffisant pour faire tous les calculs. L'équation de l'hyperplan devient alors :

$$f(x) = w.\Phi(x) + b = 0. \tag{2.5}$$

Lorsque les données sont bruitées, on introduit des variables « ressorts » ξ_i afin de relâcher la contrainte de l'équation (2.3) :

$$y_i(w.\Phi(x_i) + b) \geq 1 - \xi_i, \xi_i \geq 0, i = 1, ..., N. \tag{2.6}$$

On veut déterminer le vecteur $w \in \mathcal{F}$ et b tel qu'on minimise la marge, en autorisant des erreurs de classement :

$$\begin{array}{l} \text{minimiser}_{w,b} \frac{1}{2} \|w\|^2 + C \sum_{i=1}^{N} \xi_i \\ \text{sous contraintes } (2.6), \end{array} \tag{2.7}$$

où $C > 0$ est un paramètre de régularisation qui détermine le compromis entre le problème de maximisation de la marge et le nombre de points mal-classés. Lorsque la valeur de C est grande, peu d'exemples seront mal classés par la fonction SVM.

Pour résoudre le problème (2.7), on introduit les multiplicateurs de Lagrange $\alpha_i \geq 0$, où α_i sont des vecteurs, et le lagrangien :

$$L(w, b, x) = \frac{1}{2} \|w\|^2 - \sum_{i=1}^{N} \alpha_i (y_i(w.\Phi(x_i) + b) - 1). \tag{2.8}$$

[1]Une fonction $k(x, y)$ respecte la condition de Mercer si pour toute fonction $g(x)$ telle que $\int g(x)^2 \partial x$ est fini, on a $\int \int k(x, y)g(x)g(y)\partial x \partial y \geq 0$.

On minimise alors $L(w, b, x)$ en fonction de w et b en passant par le problème dual, qui introduit la fonction noyau k :

$$\begin{aligned} &\text{maximiser}_\alpha \sum_{i=1}^{n} \alpha_i - \frac{1}{2} \sum_{i,j=1}^{N} \alpha_i \alpha_j y_i y_j k(x_i, x_j) \\ &\text{sous contraintes :} \\ &\begin{cases} 0 \leq \alpha_i \leq C \text{ , } i = 1 \text{ à } N \\ \sum_{i=1}^{n} \alpha_i y_i = 0. \end{cases} \end{aligned} \tag{2.9}$$

Résoudre le problème dual d'optimisation permet d'obtenir les coefficients α_i, ce qui conduit à la fonction séparatrice non-linéaire suivante (aussi appelée fonction SVM) :

$$f(x) = \sum_{i=0}^{N} \alpha_i y_i k(x, x_i) + b = 0. \tag{2.10}$$

2.1.2 Fonction noyau

La fonction Φ permet de trouver une transformation de l'espace initial $\mathcal{X}$ vers un espace de redescription $\mathcal{F}$ de plus grande dimension. Or, on peut calculer des produits scalaires dans l'espace $\mathcal{F}$ sans définir explicitement la fonction Φ en introduisant une fonction noyau $k(x, y) = \Phi(x).\Phi(y)$. La condition de Mercer permet de déterminer si une fonction $\mathcal{X} \times \mathcal{X} \rightarrow \mathbb{R}$ correspond à un produit scalaire dans l'espace $\mathcal{F}$. Parmi les fonctions qui satisfont cette condition, on peut citer :

- le noyau polynomial de degré p : $k(x, y) = (a(x.y) + b)^p$;
- le noyau gaussien :

$$k(x, y) = \exp\left(-\frac{\|x - y\|^2}{2\sigma^2}\right). \tag{2.11}$$

L'avantage du noyau gaussien est que la fonction SVM produite peut approcher n'importe quelle fonction de discrimination, grâce au paramètre $\sigma > 0$ qui permet de régler l'écart type de la gaussienne. Ainsi, en diminuant la valeur de σ, on peut approcher des formes très irrégulières, tandis qu'une forte valeur de σ produira des formes plus lisses.

2.1.3 Parcimonie

La solution α du problème (2.9) vérifie les conditions de Karush-Kuhn-Tucker (KKT) pour chaque α_i. La plupart des exemples x_i est située à l'extérieur de la marge, et la valeur de α_i associée est égale à 0 :

- $\alpha_i = 0 \rightarrow f(x_i)y_i > 1$: les exemples bien classés par la fonction SVM et non situés sur la marge ont leur coefficient α_i nul ;
- $0 < \alpha_i < C \rightarrow f(x_i)y_i = 1$: les exemples situés sur la marge sont tels que $0 < \alpha_i < C$;
- $\alpha_i = C \rightarrow f(x_i)y_i < 1$: les exemples mal classés par la fonction SVM ont leur coefficient $\alpha_i = C$.

Les exemples tels que $\alpha_i > 0$ sont appelés vecteurs de supports (SVs). On observe que seuls les exemples non contraints de la solution (tels que $\alpha_i \neq 0$) nécessitent le calcul de leur coefficient et sont seuls utilisés pour la détermination de l'hyperplan (2.10). En général, une faible proportion des exemples de $\mathcal{S}$ constitue la solution : la méthode est donc parcimonieuse.

2.1.4 Résolution du problème quadratique

Apprendre une fonction SVM requiert la résolution d'un problème d'optimisation quadratique (2.9) (problème QP) : la matrice $k(x_i, x_j)$ doit être stockée dans une matrice $N \times N$. La résolution du pro-

blème requiert donc un espace mémoire $O(N^2)$ et la complexité calculatoire est en $O(N^3)$. Différents algorithmes ont été développés pour résoudre des SVMs en grande dimension. Parmi ces algorithmes, on peut citer :

- shrinking [Joachims, 1999] : définit une heuristique qui permet de déterminer quels points seront exclus de la solution ou bornés ;
- SMO (Sequential Minimal Optimization) [Platt, 1999] : ne considère que deux points à chaque étape en fonction de certaines heuristiques. L'algorithme SMO utilise le shrinking pour restreindre l'espace de recherche des α_i.

2.1.5 Malédiction de la dimensionnalité et SVMs

Lorsque le nombre de dimensions du problème est trop grand en fonction de la taille de la base d'exemples, il y a un risque que la fonction de discrimination montre une mauvaise capacité de généralisation (la fonction apprise sur la base d'apprentissage ne discrimine pas bien de nouveaux exemples). Dans le domaine de l'apprentissage, ce problème est connu sous le nom de « malédiction de la dimensionnalité » (terme introduit par Bellman [Bellman, 1961]).

La malédiction de la dimensionnalité est liée aux SVMs à deux niveaux :

- Tout d'abord, la fonction noyau $k(x, y)$ définit une transformation implicite dans un espace de redescription $\mathcal{F}$ de très grande dimension. Cependant, grâce à l'astuce du noyau, aucun calcul n'est réalisé directement dans $\mathcal{F}$.
- De plus, la dimension VC des hyperplans dans $\mathbb{R}^d$ est égale à $d + 1$, ce qui signifie que lorsque la dimension de $\mathcal{F}$ tend vers l'infini, la dimension VC tend également vers l'infini. Cependant, en utilisant le principe de maximisation de la marge γ, il est possible de borner la dimension VC par une nouvelle borne, liée à γ et indépendante de la dimension des données. Plus la marge est importante, meilleures sont les performances des SVMs. Maximiser la marge permet de ne pas souffrir de la malédiction de la dimensionnalité, et d'obtenir des fonctions simples de discrimination qui séparent bien les données.

Ainsi, les SVMs sont une méthode de discrimination qui montre de bonnes performances dans des espaces de dimension importante.

2.2 Algorithme d'approximation de noyau de viabilité avec des SVMs

2.2.1 Malédiction de la dimensionnalité et approximation de noyau de viabilité

Dans de nombreuses applications, l'espace d'état est continu et les méthodes décrites dans le chapitre 1 requièrent la discrétisation de l'espace d'état. Cependant, la taille de la grille augmente exponentiellement avec la dimension (le nombre total de points de la grille est de n^d, avec n le nombre de points par dimension et d la dimension de l'espace d'état) et on se heurte donc aux limites de la machine. Ce problème est connu sous le nom de la « malédiction de la dimensionnalité » : les coûts augmentent exponentiellement avec la dimension de l'espace d'état. Les coûts se situent à deux niveaux :

- en terme de capacité de mémoire, pour le stockage de la grille ;
- en terme de ressources de calcul (dans notre cas, pour la détermination de l'étiquette des points à chaque itération de l'algorithme et le calcul de L_h^n).

Ainsi, les algorithmes proposés dans cette thèse souffrent de la malédiction de la dimensionnalité et ne sont applicables que pour des systèmes de faible dimension (5 ou 6), en utilisant une grille avec peu de points par dimension.

2.2.2 Respect des conditions d'application de l'algorithme

Nous n'avons pas de démonstration rigoureuse qui montre qu'il est toujours possible de trouver des paramètres d'apprentissage tels que les SVMs respectent les conditions d'application (1.18) et (1.19) de l'algorithme 1.1. Cependant, en pratique, nous avons facilement trouvé des paramètres de la SVM qui permettent d'obtenir une fonction qui ne fait aucune erreur de classification, et qui n'a aucune irrégularité supérieure à $\beta(h)$, ce qui implique que les conditions sont respectées pour $\lambda = 1$. Certains arguments nous font penser que cela sera généralement le cas.

Considérons l'algorithme des $k - ppv$ (voir section 1.2.1) comme méthode de discrimination, pour $k = 1$. Cette méthode associe à un point x l'étiquette ($+1$ ou -1) de son plus proche voisin sur K_h. On peut facilement montrer que le résultat obtenu respecte les conditions d'application (1.18) et (1.19), avec $\lambda = 1$. Si on définit L_h^n comme le sous-ensemble de K composé des exemples ayant une étiquette positive avec l'algorithme des $k - ppv$, alors par définition, chaque point de L_h^n sera à une distance inférieure à $\beta(h)$ d'un point de la grille K_h^n, et chaque point de $K \backslash L_h^n$ sera également à une distance inférieure à $\beta(h)$ d'un point de la grille $K_h \backslash K_h^n$.

Considérons maintenant les SVMs. En utilisant un noyau gaussien (2.11), la fonction SVM produite $f_n(x)$ (si on considère l'itération n de l'algorithme d'approximation) peut approcher n'importe quelle fonction de discrimination, et il est toujours possible de trouver des paramètres C et σ qui donneront une fonction qui ne fait aucune erreur de classement. Dès que $\sigma \geq \beta(h)$, la frontière de la fonction SVM sera plus lisse que celle obtenue en utilisant l'algorithme des $k - ppv$. Dans ce cas, si on définit l'approximation L_h^n comme suit :

$$L_h^n = \{x \in K \text{ tels que } f_n(x) \geq 0\}, \tag{2.12}$$

les conditions (1.18) et (1.19) sont respectées, avec $\lambda = 1$.

On peut se demander alors pourquoi utiliser les SVMs plutôt que les $k - ppv$. Une première raison essentielle est liée à l'efficacité des SVMs, particulièrement lorsque l'espace d'état est en grande dimension. De plus, les $k-ppv$ retiennent tous les exemples de la base d'apprentissage pour définir la frontière de décision : si la taille de K_h est importante, l'algorithme nécessite une grande ressource mémoire et une complexité calculatoire élevée. Les SVMs fournissent quant à elles un résultat parcimonieux. Enfin, les SVMs permettent d'utiliser des méthodes pour optimiser les contrôles, en utilisant une méthode de descente de gradient par exemple, et permettent ainsi d'éviter l'explosion du temps de calcul lorsque la dimension de l'espace des contrôles augmente.

2.2.3 Méthode d'optimisation pour trouver un contrôle viable

L'expression analytique de L_h^n peut être utilisée pour trouver un contrôle qui permette au système de rester à l'intérieur de l'approximation courante du noyau. Dans le voisinage de la frontière de L_h^n, les directions pour lesquelles $f_n(x)$ augmente vont vers l'intérieur de L_h^n, et celles pour lesquelles $f_n(x)$ diminue vont vers l'extérieur. $f_n(x)$ fournit aussi une sorte de fonction barrière, similaire à celle utilisée par [Spiteri *et al.*, 2000]. Deux différences néanmoins : elle n'est pas située sur le bord de K mais sur l'approximation du noyau et elle n'est pas basée sur une fonction logarithmique. Dans ce contexte, maximiser $f_n(x)$ fournit un contrôle qui garde le système à l'intérieur de L_h^n, si un tel contrôle existe. Nous décrivons maintenant la procédure d'optimisation plus en détail, tout d'abord en considérant une optimisation sur un pas de temps, puis sur plusieurs pas de temps. On utilise l'algorithme de descente de gradient car c'est une méthode courante facile à mettre en oeuvre.

2.2.3.1 Algorithme de descente de gradient

Le principe de l'algorithme de descente de gradient est de se déplacer dans l'espace dans le sens de la plus grande pente d'une fonction jusqu'à atteindre un minimum. Cette méthode ne garantit que

l'obtention d'un minimum local.
Considérons une fonction dérivable $g(x)$. La méthode de descente de gradient recherche le point x^* qui minimise la fonction $g(x)$. Pour cela, on fixe un point initial x_0 et on construit une suite de points $x^{(k)}$ telle que :

$$x^{k+1} = x^k + \eta \nabla g(x^k) \tag{2.13}$$

avec η pas positif, jusqu'a ce que $\nabla g(x^k)$ soit très petit. On a alors $x^* = x^k$.
Dans le cas où l'on cherche le maximum d'une fonction, il suffit alors de changer le signe de la fonction : maximiser $g(x)$ revient à minimiser $-g(x)$.

2.2.3.2 Optimisation sur un pas de temps

Définissons tout d'abord la fonction $D(x, u)$ à maximiser. Lorsque l'on considère un point $x \in K$, on peut directement utiliser la fonction SVM f_n pour définir la fonction à maximiser :

$$\text{si } x \in K, D(x, u) = f_n(x + \varphi(x, u)dt). \tag{2.14}$$

Il y a cependant deux cas où on ne peut pas utiliser la fonction f_n telle que définie. Lorsque $(x + \varphi(x, u)dt) \notin K$, nous ne pouvons pas utiliser la fonction car on se situe dans une zone non-définie par la base d'apprentissage, et le comportement de f_n n'est alors pas contraint. Le second cas correspond à la première itération de l'algorithme lorsque aucune fonction SVM n'est disponible. Dans ces deux cas, on utilise la distance d'un point à $c \in \mathbb{R}^d$, le centre de K, pour définir la fonction D :

$$\text{si } x \notin K \text{ ou } x + \varphi(x, u)dt \notin K, D(x, u) = - \parallel x + \varphi(x, u)dt - c \parallel . \tag{2.15}$$

La maximisation de cette distance garantit de trouver un contrôle qui tend à ramener le système à l'intérieur de K.

On définit maintenant le contrôle optimal comme la valeur u^* qui maximise la fonction $D(x, u)$:

$$u^* = \arg \max_{u \in U(x)} D(x, u). \tag{2.16}$$

On utilise une méthode de descente de gradient afin de déterminer u^*. On construit une séquence $u^{(k)}$, où u_i^k est la $i^{\text{ème}}$ composante de u^k, définie comme suit :

$$u_i^{k+1} = u_i^k + \eta \frac{\partial D(x, u^k)}{\partial u_i}. \tag{2.17}$$

Le paramètre η règle la magnitude de la mise à jour à chaque itération.

2.2.3.3 Optimisation sur j pas de temps

Cette méthode peut être étendue pour déterminer une séquence de contrôles optimaux sur j pas de temps, avec un temps de calcul raisonnable. On considère une trajectoire partant d'un point x, définie itérativement :

$$\begin{cases} t(x, u_1) = x + \varphi(x, u_1)dt, \\ t(x, u_1, ..., u_j) = t(x, u_1, ..., u_{j-1}) + \varphi(t(x, u_1, ..., u_{j-1}), u_j)dt. \end{cases} \tag{2.18}$$

La fonction D est alors définie comme suit :

$$D(x, u_1, ..., u_j) = \begin{cases} f_n(t(x, u_1, ..., u_j)) & \text{si } t(x, u_1, ..., u_j) \in K, \\ - \parallel t(x, u_1, ..., u_j) - c \parallel & \text{sinon.} \end{cases} \tag{2.19}$$

L'ensemble des contrôles optimaux $(u_1^*, .., u_j^*)$ est celui qui résout le problème suivant :

$$\arg \max_{(u_1, .., u_j)} D(x, u_1, ..., u_j). \tag{2.20}$$

Le problème peut être résolu en utilisant une procédure de descente de gradient, généralisée à partir du problème précédent.

2.2.3.4 Avantages de la procédure

L'avantage principal d'utiliser une méthode d'optimisation pour trouver s'il existe un contrôle viable est qu'il n'est plus nécessaire de discrétiser l'espace des contrôles (lorsque la fonction de contrôle est continue). Ainsi, on peut travailler avec des problèmes dont l'espace des contrôles est en grande dimension.

De plus, il est possible d'utiliser plusieurs pas de temps à chaque itération, ce qui permet potentiellement d'obtenir une meilleure approximation de la dynamique continue F. Pour un pas de discrétisation en temps donné, la qualité de l'approximation de la dynamique F dépend du schéma utilisé : plus la valeur de dt est petite, plus la trajectoire associée à un point approchera la dynamique continue. En effet, on approche le point $x(t+dt)$ en utilisant un schéma d'Euler ($x(t+dt) = x(t) + \varphi(x(t), u(t)dt)$ et l'erreur d'approximation est en $O(dt^2)$. L'approximation de la dynamique continue peut être améliorée en utilisant j pas de temps pour calculer le successeur d'un point. Pour une valeur de pas de temps dt, on obtient un successeur au temps $j.dt = dt'$. L'erreur de l'approximation est donc dans ce cas en $O(j.dt^2)$, au lieu de $O(dt'^2)$ si on considère une approximation avec un pas de temps dt'. Pour un pas de discrétisation donné, considérer plusieurs pas de permet d'obtenir potentiellement une trajectoire qui approche mieux la dynamique, et on obtient ainsi une approximation de meilleure qualité pour une discrétisation dans l'espace donnée.

Enfin, nous avons constaté, pour au moins quelques exemples, que l'utilisation de plusieurs pas de temps permet de diminuer l'effet diffusif de l'algorithme et d'obtenir ainsi une approximation plus fine du noyau de viabilité. Cependant, des investigations supplémentaires sont nécessaires pour prouver mathématiquement ce constat.

2.2.4 Algorithme simplifié

Dans l'algorithme d'approximation 1.1, on doit déterminer si le meilleur contrôle conduit à un point qui est à une distance supérieure à $\mu\beta(h)$ de l'ensemble L_h^n. Cette distance n'est pas directe à calculer et dans un premier temps, nous l'avons approchée en utilisant un schéma d'Euler, en suivant le gradient de la fonction D. Cependant, cette opération est coûteuse en temps de calcul et nous avons constaté sur plusieurs exemples qu'elle tend à augmenter l'effet diffusif de l'algorithme. Nous avons remarqué que, pour au moins certains problèmes, la règle simple suivante pour définir K_h^{n+1} (écrite ici dans le cas d'une optimisation à un pas de temps, mais qui peut être facilement étendue à plusieurs pas de temps) permet de limiter l'effet diffusif :

$$K_h^{n+1} = \{x_h \in K_h^n \text{ tels que } f_n(x_h + \varphi(x_h, u^*)dt) \geq -\delta \text{ et } (x_h + \varphi(x_h, u^*)dt) \in K\}. \quad (2.21)$$

On choisit $\delta = 1$ car cela limite la définition de l'approximation du noyau de viabilité aux marges -1 de la fonction SVM. Cela signifie que les vecteurs de support d'étiquette -1 sont situés à l'intérieur de l'approximation, et généralement proches de sa frontière (voir figure 2.2). Cette définition tend à satisfaire une condition plus stricte que (1.19), mais ne la garantit pas.

2.2.5 Algorithme général d'approximation de noyau de viabilité avec des SVMs

L'algorithme 2.1 décrit l'algorithme général simplifié d'approximation d'un noyau de viabilité en utilisant des SVMs. Le point x^* est le point obtenu après avoir utilisé le contrôle u^* (ou $(u_1^*, ..., u_j^*)$ pour une optimisation sur j pas de temps).

Algorithme 2.1 Algorithme général simplifié d'approximation d'un noyau de viabilité en utilisant des SVMs

$\mathcal{S} \leftarrow \emptyset$
$n = 0$
$K_h^0 \leftarrow K_h$, $K_h^1 \leftarrow \emptyset$

// Première itération
Pour tout $x_h \in K_h^0$ **Faire**
  Si $x_h^* \in K$ **Alors**
    $\mathcal{S} \leftarrow \mathcal{S} \bigcup (x_h, +1)$
    $K_h^1 \leftarrow K_h^1 \bigcup x_h$
  Sinon
    $\mathcal{S} \leftarrow \mathcal{S} \bigcup (x_h, -1)$
  Fin si
Fin pour

// Itérations suivantes
Faire
  $n = n + 1$
  Calculer $f_n(x)$ à partir de $\mathcal{S}$
  $\mathcal{S} \leftarrow \emptyset$
  $K_h^{n+1} \leftarrow \emptyset$
  Pour tout $x_h \in K_h^n$ **Faire**
    Si $(f_n(x_h^*) \geq -1)$ et $(x_h^* \in K)$ **Alors**
      $\mathcal{S} \leftarrow \mathcal{S} \bigcup (x_h, +1)$
      $K_h^{n+1} \leftarrow K_h^{n+1} \bigcup x_h$
    Sinon
      $\mathcal{S} \leftarrow \mathcal{S} \bigcup (x_h, -1)$
    Fin si
  Fin pour
Jusqu'à $K_h^{n+1} = K_h^n$
Définir L_h^n à partir de $f_n(x)$
Renvoyer L_h^n

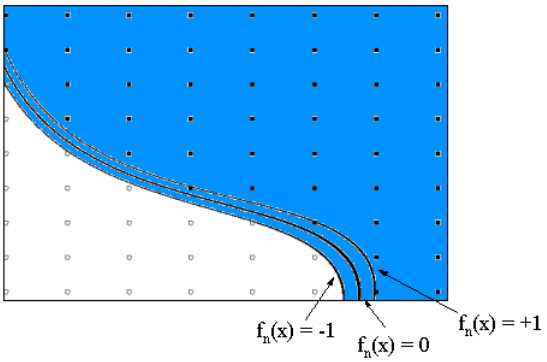


FIG. 2.2 – Les vecteurs de support sont situés sur les marges $+1$ et -1. Dans la simplification de l'algorithme, les vecteurs de support avec une étiquette -1 sont situés à l'intérieur de l'approximation du noyau (représentée en bleu), ce qui tend à satisfaire une condition plus stricte que (1.19).

2.3 Contrôleurs de viabilité utilisant des SVMs

Le contrôleur lourd de viabilité défini par l'algorithme 1.2 nécessite le calcul de la distance d'un point à la frontière de l'approximation ∂L_h^p. Cependant, on a vu dans la section précédente que la distance entre un point et une fonction SVM n'est pas directe à calculer. On adapte donc la procédure au cas où l'on choisit les SVMs comme méthode d'apprentissage.

On utilise la propriété que $f_p(x)$, la fonction associée à L_h^p, peut jouer le rôle d'une fonction barrière dans le voisinage de la frontière de l'approximation du noyau de viabilité. On définit la procédure suivante :

Procédure 2.1 (Contrôleur lourd de viabilité en utilisant une fonction SVM)
Pour Δ un réel positif donné, on définit :

$$A_\Delta = \{x \text{ tels que } f_p(x) \geq \Delta\}\,. \tag{2.22}$$

Considérant un état initial $x_0 \in A_\Delta$ et un contrôle initial $u_0 \in U(x)$ choisi aléatoirement, la procédure associe un contrôle u_{n+1} à l'itération $n+1$ comme suit :
– *si $(x_n + \varphi(x_n, u_n)dt) \in A_\Delta$, on garde le même contrôle $u_{n+1} = u_n$;*
– *sinon, $u_{n+1} = \arg\max_{u \in U(x)} f_p(x_n + \varphi(x_n, u)dt)$.*

En pratique, on peut définir des contrôleurs plus ou moins prudents, en anticipant sur k pas de temps. A partir d'un point x_n, on vérifie pour $i = 1, ..., k$, si on applique k fois le contrôle u_n, si le point $t(x_n, u_n, ..., u_n)$ appartient à l'ensemble A_Δ. Si oui, on avance d'un pas avec $u_{n+1} = u_n$. Si non, on recherche une séquence de contrôles qui permette au point $t(x_n, u_{n+1}, ...u_{n+k})$ de rester dans A_Δ, et on applique le contrôle u_{n+1}. Ainsi, plus le nombre de pas d'anticipation k est élevé, plus le contrôleur est prudent. L'algorithme 2.1 devient alors :

Procédure 2.2 (Contrôleur lourd de viabilité en utilisant une fonction SVM - version prudente)
Considérant un état initial $x_0 \in A_\Delta$ et un contrôle initial $u_0 \in U(x)$ choisi aléatoirement, la procédure associe un contrôle u_{n+1} à l'itération $n+1$ comme suit :
– *si $t(x_n, u_n, ..., u_n) \in A_\Delta$, on garde le même contrôle $u_{n+1} = u_n$;*
– *sinon, $(u_{n+1}, ..., u_{n+k}) = \arg\max_{u \in U(x)} f(t(x_n, u_{n+1}, ...u_{n+k}))$ et on applique le contrôle u_{n+1}.*

Il n'y pas de garantie mathématique que cette procédure permette de garder le système indéfiniment dans K. Cependant, en pratique, il apparaît que l'anticipation sur plusieurs pas de temps augmente considérablement les chances de rester dans K. Dans tous les cas, mettre la fonction barrière sur la

frontière d'une approximation fine du noyau donne de meilleures garanties qu'une barrière sur les bords de K (comme cela est défini dans [Spiteri *et al.*, 2000]).

L'avantage de définir une procédure de contrôle lorsque l'approximation du noyau est donnée par une fonction SVM est la parcimonie du contrôleur : même si le temps d'approximation du noyau est long, la fonction résultante est définie à l'aide d'un nombre de points réduit, et le temps de calcul d'une trajectoire est faible.

2.4 Exemples d'application

Dans cette section, nous illustrons l'algorithme d'approximation avec des SVMs sur trois exemples : deux exemples simples de problème de population et de consommation et un troisième plus complexe de gestion d'un écosystème marin. Nous montrons un exemple d'approximation progressive d'approximation de noyau de viabilité. Nous constatons sur le problème de population que l'utilisation de l'algorithme simplifié et de plusieurs pas de temps permet également de réduire l'effet diffusif. Nous donnons également quelques exemples de contrôleurs. Nous illustrons ensuite l'algorithme sur un problème plus complexe de gestion d'un écosystème marin.
Les résultats ont été obtenus en utilisant un logiciel spécifique d'approximation de noyau de viabilité, de bassin de capture et de calcul des valeurs de résilience, développé durant la thèse. Ce logiciel est décrit en annexe A.

2.4.1 Modèle simple de croissance d'une population dans un espace limité

Le système représente une population avec un apport constant de ressources, sans prédateurs, mais qui évolue dans un espace limité. L'état du système $(x(t), y(t))$ représente la taille d'une population $x(t)$, qui fluctue avec le taux d'évolution $y(t)$. La population doit rester dans un certain intervalle $K = [a; b] \times [d; e]$, avec $a > 0$. Ce système a été étudié par Maltus, puis par Verhulst et ensuite redéveloppé par Aubin [Aubin, 2002], qui a ajouté une borne d'inertie. La borne d'inertie c limite la dérivée du taux d'évolution à chaque pas de temps. Le système en temps discret, défini avec un intervalle de temps dt, peut être réécrit comme suit :

$$\begin{cases} x(t+dt) = x(t) + x(t)y(t)dt \\ y(t+dt) = y(t) + u(t)dt \text{ avec } |u(t)| \leq c. \end{cases} \tag{2.23}$$

Pour ce système simple, il est possible de dériver analytiquement le noyau de viabilité du système [Aubin, 2002], son expression (pour $dt \to 0$) est la suivante :

$$Viab(K) = \left\{ (x,y) \text{ tels que } \begin{array}{l} x \in [a;b] \text{ et} \\ y \in \left[-\sqrt{2c\log\left(\frac{b}{x}\right)}; \sqrt{2c\log\left(\frac{x}{a}\right)} \right] \end{array} \right\}. \tag{2.24}$$

Pour les simulations suivantes, on pose $a = 0,2$; $b = 3$; $c = 0,5$; $d = -2$ et $e = 2$.

La figure 2.3 montre un exemple d'approximation progressive du noyau en utilisant l'algorithme initial (avec la constante de Lipschitz). La figure 2.4 présente l'approximation finale du noyau, en utilisant une optimisation sur 1 pas de temps, comparée à une optimisation sur 6 pas de temps. Les paramètres des simulations sont donnés dans le tableau 2.1 (simu 1 et simu 2 respectivement). On remarque que l'approximation est plus proche du noyau théorique (représenté par les lignes noires) lorsque l'optimisation est faite sur 6 pas de temps.

La figure 2.5 illustre l'efficacité de l'algorithme simplifié proposé dans la section 2.2.4. Les paramètres utilisés pour cette simulation sont donnés dans le tableau 2.1 (simu 3 et simu 4). L'approximation donnée par la figure 2.5 *b* peut être comparée à celle donnée par la figure 2.4 *b*. On remarque, sur ces

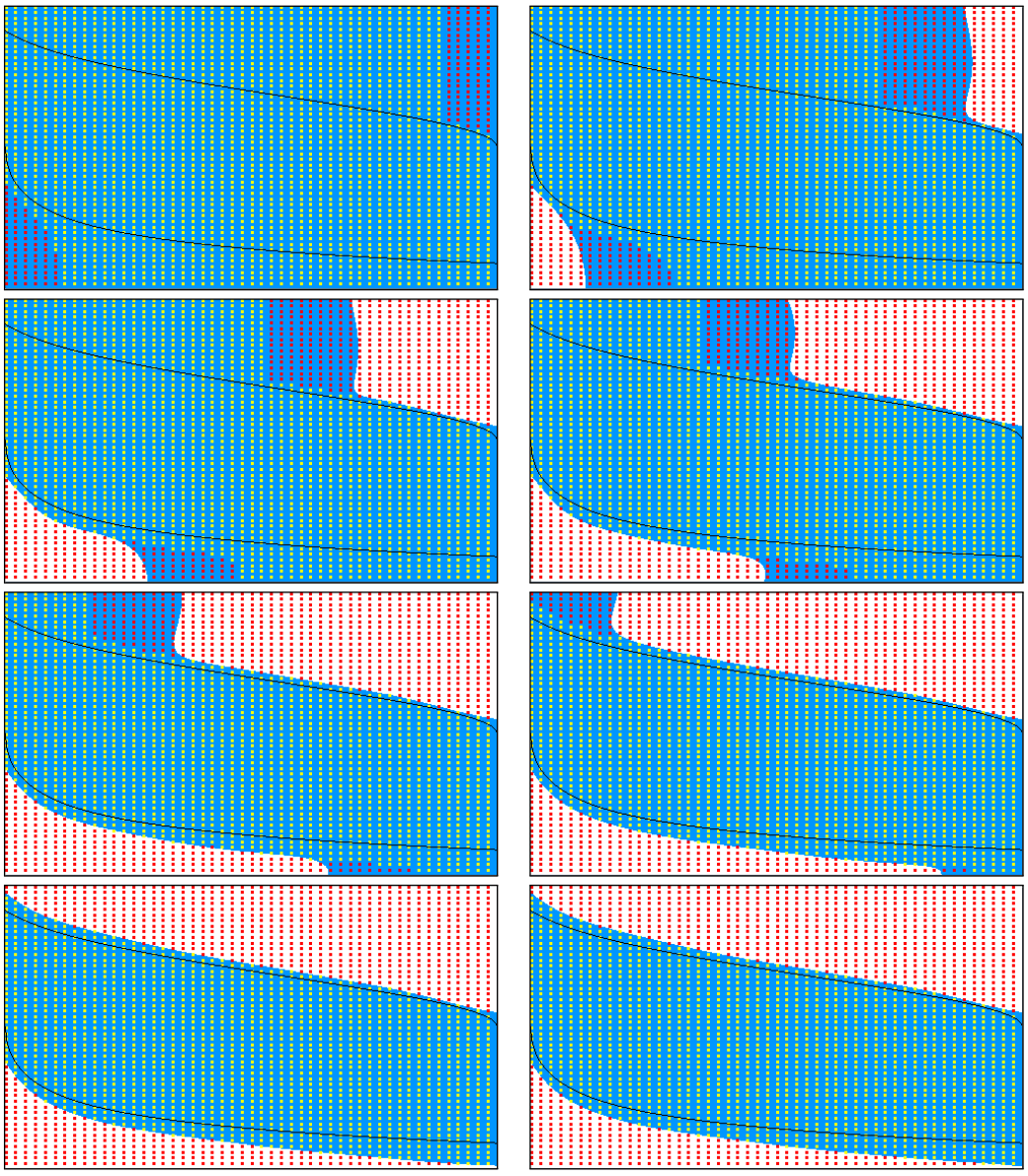
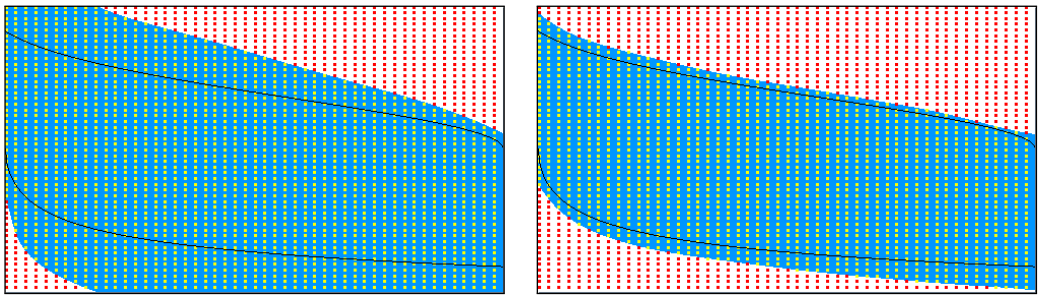


FIG. 2.3 – Exemple de l'approximation progressive du noyau de viabilité pour le problème population (algorithme initial). L'axe horizontal représente la taille de la population (x) et l'axe horizontal le taux d'évolution (y). K est le rectangle qui limite les points de la grille. Les différentes approximations L_h^n sont représentées en bleu. Les points jaunes sont ceux appartenant à K_h^{n+1}.

simulations, que l'algorithme simplifié limite l'effet diffusif de l'algorithme. On constate également que la qualité de l'approximation augmente lorsque le pas de la grille h diminue.

La figure 2.6 montre également deux exemples de trajectoires sur 120 pas de temps, obtenues à partir de la procédure du contrôleur lourd, partant d'un point initial $x_0 = (1,9; 0,5)$. Les paramètres du calcul du noyau sont donnés dans le tableau 2.1 (simu 4). On pose $\Delta = 12$ et on compare deux types de contrôleur : un premier, en anticipant sur 2 pas de temps, et un plus prudent, en anticipant sur 6 pas de temps.

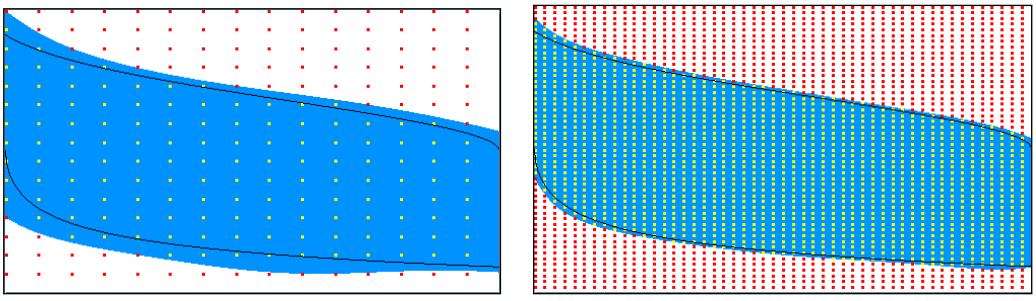


(a) (b)

FIG. 2.4 – Augmenter le nombre de pas de temps pour l'optimisation diminue l'effet diffusif de l'algorithme (algorithme initial). L'optimisation est faite sur *(a)* 1 pas de temps *(b)* 6 pas de temps.

	simu 1	**simu 2**	**simu 3**	**simu 4**	**simu 5**
Dimension	2	2	2	2	2
Pas de la grille h Nombre total de points	$0,02$ 2601	$0,02$ 2601	$0,05$ 441	$0,02$ 2601	$0,02$ 2601
dt	$0,03$	$0,03$	$0,03$	$0,03$	$0,03$
Nombre de pas	1	6	6	6	4
C	30000	30000	30000	30000	30000
σ^2	$0,05$	$0,05$	$0,05$	$0,05$	$0,05$
Nombre d'itérations	29	7	7	7	26
Nombre final de SV	18	25	21	26	25
Temps (secondes)	17	13	2	13	34

TAB. 2.1 – Paramètres et résultats des simulations pour les problèmes population et consommation.



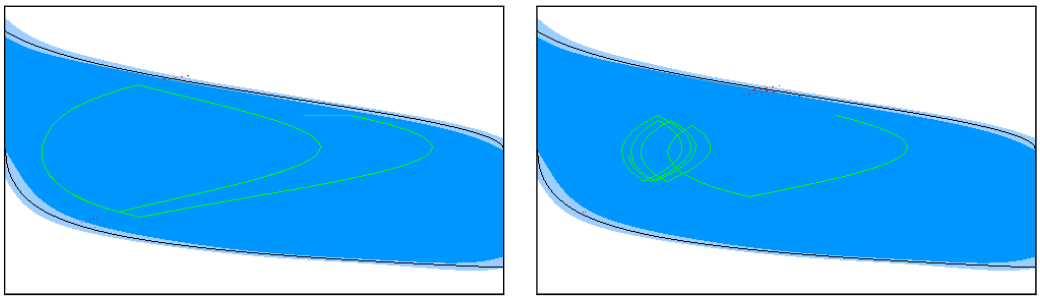
(a) (b)

FIG. 2.5 – Augmenter la résolution de la grille améliore la qualité de l'approximation (algorithme simplifié). La grille contient *(a)* 21 points par dimension *(b)* 51 points par dimension.

2.4.2 Problème de consommation

On considère maintenant un problème de consommation [Aubin, 1991], défini en deux dimensions $x(t)$ et $y(t)$. La variable $x(t)$ représente la consommation d'une matière première et $y(t)$ son prix. L'ensemble des contraintes est l'ensemble $K = [0;b] \times [0;e]$. L'évolution des prix entre deux pas de temps est bornée. La dynamique du système représente la consommation d'une matière première, freinée par les prix :

$$\begin{cases} x(t+dt) = x(t) + (x(t) - y(t))dt \\ y(t+dt) = y(t) + u(t)dt \text{ avec } |u(t)| \leq c. \end{cases} \tag{2.25}$$



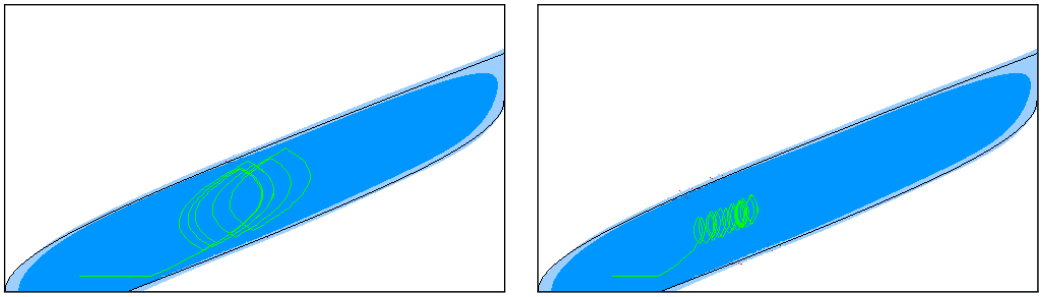
(a) (b)

FIG. 2.6 – Exemples de trajectoires lourdes à partir d'un point $x_0 \in A_\Delta$ pour le problème population. La différence entre L_h^p et A_Δ est représentée en bleu clair, la trajectoire en vert. Anticipation sur *(a)* 2 pas de temps *(b)* 6 pas de temps.

Pour ce système simple, il est possible de dériver analytiquement le noyau de viabilité du système [Aubin, 1991], son expression (pour $dt \to 0$) est la suivante :

$$Viab(K) = \left\{ (x,y) \text{ tels que } \begin{array}{l} x \in [a;b] \text{ et} \\ y \in \left[x - c + c\exp^{-\frac{x}{c}}; x + c - c\exp^{-\frac{x-b}{c}}\right] \end{array} \right\}. \tag{2.26}$$

Pour les simulations suivantes, on pose $b = 2$; $c = 0,5$ et $e = 3$. Les paramètres de simulation sont donnés dans le tableau 2.1 (simu 5).
La figure 2.7 présente un exemple d'approximation de noyau de viabilité en utilisant l'algorithme simplifié. Elle montre également 2 exemples de trajectoires sur 100 pas de temps en partant d'un point initial $x_0 = (0,3; 0,15)$. On pose $\Delta = 12$ et on compare deux types de contrôleur : un premier, en anticipant sur 2 pas de temps, et un plus prudent, en anticipant sur 6 pas de temps.



(a) (b)

FIG. 2.7 – Exemple d'approximation du noyau de viabilité et trajectoires lourdes à partir d'un point $x_0 \in A_\Delta$ pour le problème consommation. L'axe horizontal représente la consommation (x) et l'axe vertical le prix (y). K est le rectangle qui limite les points de la grille. La différence entre L_h^p et A_Δ est représentée en bleu clair. Anticipation sur *(a)* 2 pas de temps *(b)* 6 pas de temps.

On peut comparer l'approximation du noyau de viabilité obtenue ici avec celles données dans [Bokanowski *et al.*, 2006], qui ont approché le noyau de viabilité pour le problème consommation, et comparé leur résultat avec celui obtenu avec l'algorithme de viabilité de Saint-Pierre. Avec une grille de 50 points par dimension, les approximations obtenues en utilisant le schéma anti-diffusif Ultra-Bee fournissent un résultat proche de celui obtenu en utilisant l'algorithme d'approximation de noyau de viabilité avec des SVMs. Par contre, l'approximation obtenue en utilisant l'algorithme de viabilité est beaucoup moins précise lorsque l'on utilise le même nombre de points par dimension pour la discrétisation.

2.4.3 Problème en six dimensions : écosystème marin

Mullon *et al.* [Mullon *et al.*, 2004] ont proposé un modèle dynamique d'évolution de la biomasse dans l'écosystème marin du Sud Benguela, composé de 5 compartiments (détritus, phytoplancton, zooplancton, poisson pélagique et poisson benthique). Ils ont étudié le modèle dans une perspective de viabilité (voir également [Cury *et al.*, 2005] pour une étude du même système, et [Le Fur *et al.*, 1999] pour l'application de la viabilité en halieutique), en déterminant, pour un volume de pêche constant, si la biomasse de chaque espèce reste dans un certain intervalle, en prenant en compte les incertitudes sur les coefficients d'interaction dans le modèle. Le but est d'étudier les valeurs de pêche constantes qui permettent d'assurer la pérennité de l'écosystème. Dans cette section, nous étendons le problème et nous nous focalisons sur les politiques de pêche qui permettent de garder le système viable (au lieu de considérer une pêche constante).

La principale difficulté de ce problème est que l'espace des contrôles est en dimension 16. [Mullon *et al.*, 2004] ont résolu ce problème en proposant une méthode adaptée aux équations linéaires des évolutions. On utilise ici l'algorithme simplifié d'approximation d'un noyau de viabilité en utilisant les SVMs. Grâce à la définition continue de la frontière de l'approximation, on peut utiliser des méthodes d'optimisation pour rechercher s'il existe au moins un contrôle qui permette de garder le système viable, ce qui permet de travailler dans des espaces de contrôle importants.

2.4.3.1 Modèle

En suivant une approche classique [Walters *et al.*, 1997], on suppose que la variation de la biomasse d'une espèce i due à sa prédation par une autre espèce j dépend linéairement des biomasses des deux espèces (B_i et B_j), avec des coefficients r_{ji} et d_{ji}. La biomasse perdue par une espèce i à cause de sa prédation par les autres espèces en un pas de temps dt est donnée par l'équation suivante :

$$\frac{dB_i(i \rightarrow)}{dt} = \sum_j (r_{ji} B_j + d_{ji} B_i). \tag{2.27}$$

La variation de biomasse B_i due à cette interaction prend également en compte l'assimilation de la biomasse des autres espèces j, multipliée par un coefficient de croissance, noté g_i. Le gain de biomasse de l'espèce i grâce à la consommation des autres espèces est donné par l'équation suivante :

$$\frac{dB_i(i \leftarrow)}{dt} = g_i \sum_j (r_{ij} B_i + d_{ij} B_j). \tag{2.28}$$

Pour les détritus, la variation de biomasse suit le même principe, et intégre également la biomasse non-assimilée par les autres espèces (sauf le phytoplancton), qui est ajoutée à la biomasse des détritus B_1 :

$$\frac{dB_1(\text{non-assimilé})}{dt} = \sum_{l=3}^{5} \sum_{k=1}^{5} g_1 (1 - g_l)(r_{lk} B_l + d_{kl} B_k). \tag{2.29}$$

Le modèle du Sud Benguela prend en compte les interactions trophiques (prédation, consommation et pêche) parmi les cinq compartiments suivants : détritus ($i = 1$), phytoplancton ($i = 2$), zooplancton ($i = 3$), poisson pélagique ($i = 4$) et poisson benthique ($i = 5$). L'équation finale de l'évolution de la biomasse peut être écrite de la façon suivante :

$$\begin{cases} \frac{dB_1}{dt} = \frac{dB_1(1\leftarrow)}{dt} - \frac{dB_1(1\rightarrow)}{dt} + \frac{dB_1(\text{non-assimilé})}{dt} - Y_1 \\ \frac{dB_i}{dt} = \frac{dB_i(i\leftarrow)}{dt} - \frac{dB_1(i\rightarrow)}{dt} - Y_i, \end{cases} \tag{2.30}$$

avec Y_i la pêche sur l'espèce i.

[Mullon *et al.*, 2004] ont aussi pris en compte l'incertitude sur les paramètres r_{ij} et d_{ij}, exprimée par :

$$r_{ij} \in [\overline{r_{ij}} - \delta r_{ij}; \overline{r_{ij}} + \delta r_{ij}], d_{ij} \in \left[\overline{d_{ij}} - \delta d_{ij}; \overline{d_{ij}} + \delta d_{ij}\right]. \tag{2.31}$$

Ils ont considéré les paramètres r_{ij} et d_{ij} comme des variables de contrôle, ce qui signifie qu'on suppose ici qu'à chaque instant, la valeur de ces paramètres peut être modifiée, dans l'intervalle défini par l'équation (2.31). A chaque itération, l'évolution de la biomasse B_i d'une espèce est donc fonction d'un ensemble admissible de consommation et de prédation.

Dans cette étude, on ajoute les pêches comme une variable d'état du système, afin de trouver les états initiaux du système qui permettent de garder l'écosystème viable. Pour assurer un système durable, les contraintes de viabilité sont définies par :

$$\begin{cases} 0 \leq m_i \leq B_i \leq M_i \\ 0 \leq y_{min} \leq Y_i \leq y_{max}, Y_i' \in [-\delta y, +\delta y], i = 4, 5. \end{cases} \tag{2.32}$$

où m_i est le niveau minimal de la ressource, M_i la biomasse maximale que peut contenir le système, y_{min} et y_{max} sont respectivement les niveaux minimum et maximum de pêche pour les poissons pélagiques et benthiques. Le paramètre δy limite l'évolution du niveau de pêche entre deux pas de temps. On suppose ici que les niveaux de pêche pour les deux espèces sont les mêmes. Certaines valeurs de r_{ij} et d_{ij} sont nulles, l'espace des contrôles pour les incertitudes est en dimension 16, et le contrôle sur la pêche est en dimension 1. Ces contraintes, qui sont les valeurs critiques d'un système « en bonne santé », permettent de lier les objectifs de pêche avec le principe de durabilité de l'écosystème. La figure 2.8 résume la structure de l'écosystème.

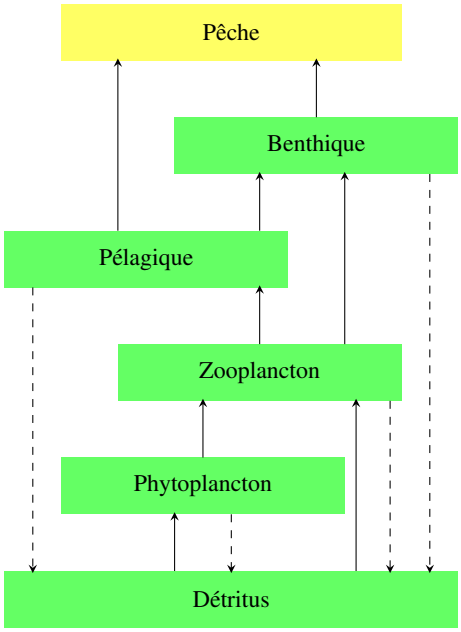


FIG. 2.8 – Composants et structure de l'écosystème du Sud Benguela. Les flèches représentent les flux entre les compartiments : les flèches pleines indiquent une consommation ou une prédation, et les flèches en pointillé représentent les flux vers les détritus (inspiré par [Mullon *et al.*, 2004]).

Compartiment	m_i (tonnes/km^2)	M_i (tonnes/km^2)
Détritus	100	2000
Phytoplancton	30	400
Zooplancton	20	200
Poisson pélagique	5	60
Poisson benthique	5	30

TAB. 2.2 – Valeurs minimales et maximales des biomasses B_i pour les cinq espèces.

2.4.3.2 Problème de viabilité

Dans le problème de viabilité, les contrôles sont la variation de la pêche δy et les incertitudes sur les coefficients r_{ij} et d_{ij}. Cela signifie que, pour chaque état du système situé à l'intérieur du noyau, il existe des valeurs de ces variables qui permettent de rester dans le noyau à l'instant suivant. Ici, on suppose que $Y_4 = Y_5 = Y$, condition certes peu réaliste mais qui permet de travailler avec un système en dimension 6 (il aurait été en dimension 7 sinon).

Le problème de contrôle de viabilité est donc de déterminer la fonction de contrôle :

$$t \rightarrow r_{ij}(t), d_{ij}(t), \delta y(t) \text{ avec } i = 1, ..., 5 \tag{2.33}$$

qui permette de garder les contraintes de viabilité indéfiniment satisfaites. On approche alors le noyau de viabilité, qui correspond à l'ensemble des états initiaux pour lesquels une telle fonction existe.

2.4.3.3 Résultats

Les coefficients de contrôle sont dérivés d'un modèle Ecopath [Shannon *et al.*, 2003]. On utilise les valeurs de paramètres donnés par [Mullon *et al.*, 2004]. La table 2.2 donne les bornes de l'espace des contraintes et on pose $y_{min} = 0$ (pas de pêche du tout), $y_{max} = 5$ (qui correspond à la valeur maximale testée par [Mullon *et al.*, 2004]) et $\delta y = 0,5$ (soit 10% de la pêche maximale). Les valeurs de pêche pour les autres espèces sont mises à 0, sauf pour les détritus ($Y_1 = -1000$ tonnes/km^2, ce qui correspond à un apport de détritus).

Les figures suivantes présentent des exemples de résultats pour différentes valeurs des biomasses de chaque espèce. Le noyau obtenu est semblable à celui donné dans [Mullon *et al.*, 2004], pour un niveau de pêche donné. L'approximation du noyau est donnée en bleu, et les bords des axes sont les contraintes données par les espèces représentées. Les valeurs des paramètres pour l'algorithme d'approximation (algorithme simplifié) sont données dans le tableau 2.3.

Cet exemple illustre la capacité de l'algorithme d'approximation de noyau de viabilité utilisant des SVMs à travailler dans des espaces de contrôle importants. Cependant, il montre également ses limites pour des espaces d'état en grande dimension : l'approximation obtenue n'est pas très précise car on utilise une grille avec seulement 6 points par dimension. Cette limitation est due au temps de calcul important pour approcher le noyau de viabilité lorsque la taille de la grille augmente. En effet, le temps de calcul de l'étiquette des points est linéaire avec la taille de la grille, et de plus, le temps de calcul d'une fonction SVM est quadratique avec la taille de la base d'apprentissage. Il est ainsi difficile de travailler avec beaucoup de points par dimension, car la taille de la grille explose (par exemple, avec 10 points par dimension, la grille contient 1000000 points) et les temps de calcul également. On atteint ici les limites de l'algorithme présenté ; dans le chapitre 6, nous proposerons une procédure pour obtenir des résultats plus précis en dimension importante. Même si l'approximation obtenue ici n'est pas très précise, nous donnons rapidement quelques exemples de résultats.

On se concentre sur les valeurs de biomasse de détritus $= 2000$ tonnes / km^2 car ces valeurs assurent d'avoir un noyau non vide pour presque toutes les autres valeurs des compartiments. Pour une biomasse de détritus égale à 1600 tonnes/km^2, certains niveaux de zooplancton et phytoplancton sont nécessaires

pour assurer la viabilité du système. Pour des biomasses de détritus plus faibles, il n'y a aucun état viable : une valeur élevée de biomasse de détritus est nécessaire pour assurer la durabilité du système. On se focalise sur l'impact de la pêche sur les poissons pélagiques et benthiques.

	simulation
Dimension	6
Pas de la grille h	$0,2$
Nombre total de points	46656
dt	$0,2$
Nombre de pas	3
C	30000
σ^2	$0,5$
Nombre d'itérations	29
Nombre final de SV	1642
Temps (heures)	14

TAB. 2.3 – Paramètres et résultats de la simulation pour le problème d'écosystème marin.

Effet de la pêche sur les poissons benthiques

La figure 2.9 présente des coupes du noyau de viabilité pour des valeurs de biomasses de détritus $=$ 2000 tonnes/km^2, phytoplancton $=$ 100 tonnes/km^2, zooplancton $=$ 90 tonnes/km^2 et pour différentes valeurs de biomasses de poisson pélagique.
Les différents niveaux de biomasses de poisson pélagique, benthique et de la pêche ont une influence sur la frontière du noyau de viabilité :

- pour les plus faibles valeurs de biomasse de poisson pélagique, la biomasse de poisson benthique ne doit pas être trop importante et les plus hauts niveaux de pêche sont à éviter ;
- de la même façon, lorsque la biomasse de poisson pélagique est importante, la biomasse de poisson benthique doit être élevée et certaines faibles valeurs de pêche sont à éviter pour garantir un système viable ;
- pour les valeurs moyennes de biomasse de poisson pélagique, il n'y a pas de restriction sur le niveau des pêches.

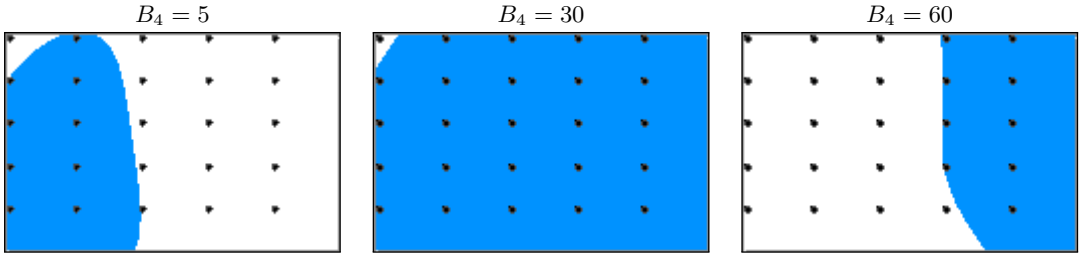


FIG. 2.9 – Approximation du noyau de viabilité pour différentes biomasses de poisson pélagique. L'axe horizontal représente la biomasse de poisson benthique (B_5), l'axe vertical la pêche Y, $B_1 = 2000$ tonnes/km^2, $B_2 = 100$ tonnes/km^2 et $B_3 = 90$ tonnes/km^2.

La figure 2.10 présente des coupes du noyau de viabilité pour des valeurs de biomasses de détritus $=$ 2000 tonnes/km^2, phytoplancton $=$ 400 tonnes/km^2, zooplancton $=$ 130 tonnes/km^2. On constate que l'approximation du noyau est plus petite que dans la figure 2.9 : une biomasse importante de poisson pélagique représente une situation non-viable. Comme précédemment, certains niveaux trop faibles ou trop forts de pêche doivent être évités. De façon générale, plus la biomasse de zooplancton est importante, plus l'approximation du noyau est petite et il n'y a pas de situation viable pour des biomasses de poisson pélagique égale à 60 tonnes/km^2.

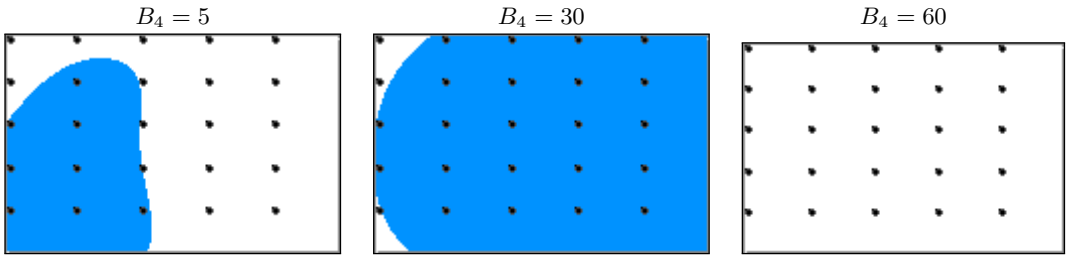


FIG. 2.10 – Approximation du noyau de viabilité pour différentes biomasses de poisson pélagique. L'axe horizontal représente la biomasse de poisson benthique (B_5), l'axe vertical la pêche Y, $B_1 = 2000$ tonnes/km^2, $B_2 = 400$ tonnes/km^2 et $B_3 = 130$ tonnes/km^2.

Effet de la pêche sur les poissons pélagiques

On étudie maintenant l'effet de la pêche sur les poissons pélagiques, en gardant les mêmes valeurs que précédemment pour les autres espèces.
La figure 2.11 présente des coupes du noyau de viabilité pour des valeurs de biomasses de détritus $= 2000$ tonnes/km^2, phytoplancton $= 100$ tonnes/km^2, zooplancton $= 90$ tonnes/km^2 et pour différentes valeurs de biomasses de poisson benthique. On remarque que les niveaux de pêche influent sur la frontière de l'approximation du noyau uniquement lorsque la biomasse de poisson benthique est faible : plus la biomasse de poisson pélagique est importante, plus la quantité de poissons pêchée peut être importante. Cependant, quelles que soient les biomasses de poisson benthique, les niveaux de pêche doivent être soigneusement contrôlés pour garantir un système viable. Pour les valeurs moyennes de poisson benthique, le système n'est pas viable pour les valeurs de pêche extrêmes. Pour les valeurs les plus importantes, la biomasse de poisson benthique ne doit pas être trop faible.

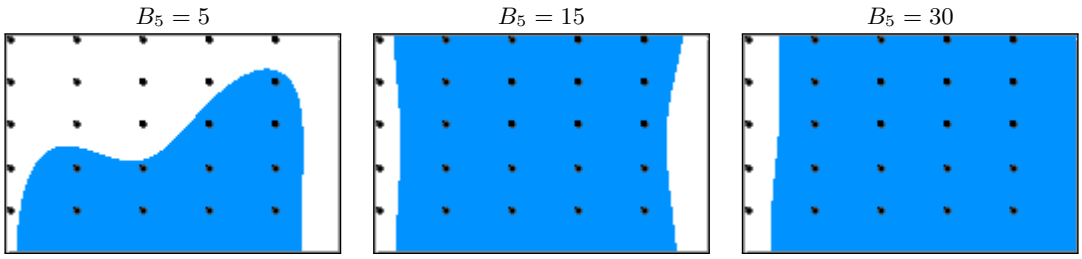


FIG. 2.11 – Approximation du noyau de viabilité pour différentes biomasses de poisson benthique. L'axe horizontal représente la biomasse de poisson pélagique (B_4), l'axe vertical la pêche Y, $B_1 = 2000$ tonnes/km^2, $B_2 = 100$ tonnes/km^2 et $B_3 = 90$ tonnes/km^2.

Pour des valeurs de biomasse de zooplancton plus grandes (figure 2.12), l'approximation du noyau est plus petite. Certains niveaux de biomasse de poisson benthique et de pêche sont nécessaires pour garantir l'existence d'une trajectoire viable :

- pour les faibles valeurs de biomasse de poisson benthique, les pêches doivent être soigneusement contrôlées ; les valeurs extrêmes représentent des situations non-viables ;
- pour les valeurs moyennes, le système n'est pas viable uniquement pour les plus grandes valeurs de biomasse de poisson pélagique ;
- les pêches ont une influence lorsque le système a des fortes valeurs de biomasses de poisson pélagique et benthique : un niveau minimal de pêche est nécessaire pour assurer la viabilité du système.

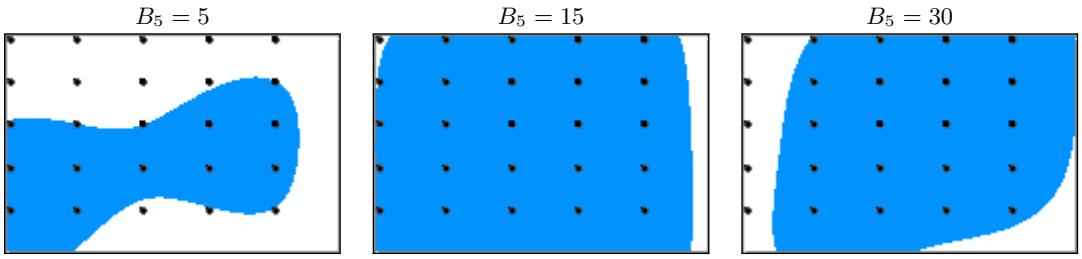


FIG. 2.12 – Approximation du noyau de viabilité pour différentes biomasses de poisson benthique. L'axe horizontal représente la biomasse de poisson pélagique (B_4), l'axe vertical la pêche Y, $B_1 = 2000$ tonnes/km^2, $B_2 = 400$ tonnes/km^2 et $B_3 = 130$ tonnes/km^2.

Deuxième partie

Approcher des bassins de capture et calculer des valeurs de résilience

Chapitre 3

Approcher un bassin de capture en utilisant une méthode d'apprentissage

Dans ce chapitre, nous proposons un algorithme d'approximation de bassins de capture et de fonction de temps minimal dans l'espace d'état initial en utilisant une méthode d'apprentissage, et nous déterminons deux variantes de l'algorithme : une première qui fournit une approximation qui inclut les bassins de capture, et une deuxième pour laquelle l'approximation obtenue est incluse dans les bassins de capture. Nous proposons également un contrôleur optimal, basé sur le résultat de l'algorithme.

Dans la section 3.1, nous définissons le bassin de capture d'un système et décrivons comment il peut être utilisé pour dériver des politiques de contrôle. Dans la section 3.2, nous adaptons tout d'abord l'algorithme proposé dans le chapitre 1 au problème d'atteinte d'une cible, en utilisant un système auxiliaire et en définissant une gestion plus efficace de la grille de points. Nous proposons ensuite un algorithme d'approximation dans l'espace d'état initial, et décrivons deux variantes : la première variante fournit une approximation par l'extérieur et la seconde variante une approximation par l'intérieur. Nous définissons les conditions mathématiques que l'algorithme d'apprentissage doit respecter pour garantir la convergence. Dans la section 3.3, nous définissons un contrôleur optimal. Pour terminer, dans la section 3.4, nous étendons les résultats au problème de minimisation d'une fonction de coût strictement décroissante.

Sommaire

3.1 Capturabilité

3.1.1 Définitions

La théorie de la viabilité traite également le problème d'atteinte d'une cible fermée $C \in K$, sans que le système ne quitte jamais l'espace des contraintes K. Le système dynamique est défini par :

$$x'(t) \in F(x(t)) = \{\varphi(x(t), u(t)), u(t) \in U(x(t))\}. \tag{3.1}$$

Les évolutions $x(t)$ *capturant la cible* sont les évolutions viables dans K jusqu'à ce qu'elles atteignent la cible C en temps fini :

$$\exists \tau \geq 0 \text{ tel que } \begin{cases} x(\tau) \in C \\ \forall t \in [0, \tau], x(t) \in K. \end{cases} \tag{3.2}$$

Le *bassin de capture* $Capt_F(K, C)$ [Aubin, 2001] du système pour F est l'ensemble de tous les états pour lesquels il existe au moins une évolution qui permette d'atteindre la cible sans quitter K (figure 3.1). Considérons la correspondance F_I, qui coïncide partout avec F sauf dans la cible :

$$F_I(x(t)) = \begin{cases} F(x(t)) & \text{si } x \notin C \\ \overline{Co}(F(x(t)) \cup \{0\}) & \text{si } x \in \partial C \\ \{0\} & \text{si } x \in int(C), \end{cases} \tag{3.3}$$

où $\overline{Co}$ est l'enveloppe convexe[1] de $(F(x(t)) \cup \{0\})$, ∂C est la frontière de la cible et $int(C)$ est l'intérieur de C. L'introduction de la correspondance F_I permet d'écrire la relation suivante :

$$Viab_{F_I}(K) = Capt_F(K, C) \cup Viab_F(K) \tag{3.4}$$

Lorsque K est un répulseur (i.e. $Viab_F(K) = \emptyset$), le bassin de capture coïncide avec le noyau de viabilité du système (3.3). Cependant, K n'est en général pas un répulseur et une autre approche doit être utilisée pour approcher le bassin de capture du système.

La *fonction de temps minimal* est la fonction qui associe à un état $x_0 \in K$ le temps minimal d'atteinte de la cible C :

$$\vartheta_C^K(x_0) = \inf \{\tau \geq 0 | \exists x(.) \text{ vérifiant } (3.1), x(0) = x_0, x(\tau) \in C \text{ et } \forall t \in [0, \tau], x(t) \in K\}. \tag{3.5}$$

Cette fonction est la fonction valeur obtenue lorsque l'on résout les équations Hamilton-Jacobi-Bellman (HJB) en programmation dynamique [Frankowska, 1989 ; Cardaliaguet *et al.*, 1997] (pour plus de détails sur la fonction valeur et son approximation dans la théorie du contrôle, le lecteur peut se référer à [Bellman, 1957] ou [Crandall et Lions, 1983]). Elle prend des valeurs dans $\mathbb{R}^+ \cup +\infty$, en particulier $\vartheta_C^K(x_0) = 0$ si $x_0 \in C$ et $\vartheta_C^K(x_0) = +\infty$ s'il n'existe aucune trajectoire qui permette à x_0 d'atteindre la cible sans quitter K. Le bassin de capture de C viable dans K peut être alors défini en utilisant la fonction de temps minimal :

$$Capt_F(K, C) = \left\{x | \vartheta_C^K(x) < +\infty\right\}. \tag{3.6}$$

On peut également définir le bassin de capture en temps fini T :

$$Capt_F(K, C, T) = \left\{x | \vartheta_C^K(x) \leq T\right\}. \tag{3.7}$$

Afin de résoudre un problème de capture de cible dans une perspective de viabilité, on considère le système dynamique auxiliaire suivant :

$$(x(t)', y'(t)) \in F_C(x(t), y(t)) = \begin{cases} F(x(t)) \times \{-1\} & \text{si } x \notin C \\ \overline{Co}((F(x(t)) \times \{-1\}) \cup (\{0\} \times \{0\})) & \text{si } x \in \partial C \\ \{0\} \times \{0\} & \text{si } x \in int(C). \end{cases} \tag{3.8}$$

[1] L'enveloppe convexe d'un ensemble de points $X \in \mathbb{R}^d$ est l'ensemble convexe minimal contenant X.

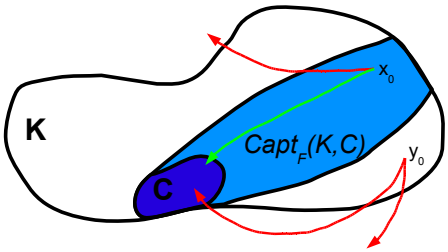


FIG. 3.1 – Exemple de bassin de capture (représenté en bleu). La cible C à atteindre est représentée en bleu foncé et l'espace des contraintes est délimité par la ligne continue. x_0 et y_0 sont deux états initiaux, et quelques évolutions possibles sont représentées par les flèches. x_0 est un point qui capture la cible car il existe au moins une évolution qui lui permet d'atteindre la cible sans quitter K (évolution capturant la cible en vert), tandis que y_0 est non viable : il n'existe aucune fonction de contrôle qui permette au système d'atteindre la cible tout en restant dans K (évolutions non viables en rouge).

La variable $y(t)$ représente le temps qui s'écoule et prend ses valeurs dans $\mathbb{R}^+$. [Frankowska, 1989 ; Aubin et Frankowska, 1996] prouvent que, avec $\mathcal{H} = K \times \mathbb{R}^+$, l'épigraphe[2] de la fonction valeur $\vartheta_C^K(.)$ est le noyau de viabilité de $\mathcal{H}$ pour l'inclusion différentielle F_C :

$$Viab_{F_C}(\mathcal{H}) = \varepsilon pi(\vartheta_C^K), \tag{3.9}$$

où $\varepsilon pi(\vartheta_C^K) \in \mathcal{X} \times \mathbb{R}^+$ est l'ensemble $\left\{(x,y) \in \mathcal{X} \times \mathbb{R}^+ | \vartheta_C^K(x) \leq y\right\}$.

3.1.2 Contrôleur de viabilité

Le bassin de capture d'un système est déterminant pour définir des politiques d'actions qui capturent la cible. A partir d'un point appartenant à $Capt(K,C)$, on sait qu'il existe une suite de contrôles qui va permettre au système d'atteindre la cible, tout en restant dans K. La question est : comment choisir ces contrôles ?

[Frankowska, 1989 ; Aubin et Frankowska, 1996] caractérisent l'épigraphe d'une fonction valeur V d'un problème de contrôle optimal comme le noyau de viabilité d'un système auxiliaire. Approcher un bassin de capture permet d'estimer la fonction valeur du problème discrétisé. La fonction valeur permet de déduire un contrôleur optimal : en tout état x, le contrôle optimal u^* est le contrôle qui permet d'atteindre le minimum de la fonction valeur :

$$u^* = \arg \min_{u \in U(x)} V(x). \tag{3.10}$$

Cette caractérisation du bassin de capture permet d'approcher la fonction de temps minimal et de contrôler ainsi le système afin qu'il atteigne la cible le plus rapidement possible.

En général, il n'existe pas de formule explicite pour déterminer le bassin de capture d'un système. Dans la section suivante, nous proposons deux algorithmes utilisant une méthode d'apprentissage qui permettent d'approcher le bassin de capture d'un système. Le premier algorithme utilise la correspondance (3.8), et le deuxième travaille directement dans l'espace d'état initial (système (3.1)).

[2] L'épigraphe de la fonction f est défini par $\varepsilon pi(f) = \{(x,y) \in I \times \mathbb{R} | f(x) \geq y\}$.

3.2 Algorithme d'approximation de bassins de capture en utilisant une méthode d'apprentissage

Dans cette section, nous proposons tout d'abord un algorithme d'approximation de bassins de capture (ou de fonction de temps minimal) en utilisant un système auxiliaire. Nous proposons ensuite deux variantes d'un algorithme d'approximation de bassins de capture dans l'espace d'état initial en utilisant une technique d'apprentissage : la première variante permet d'avoir une approximation qui inclut le bassin de capture (approximation par l'extérieur), et la deuxième variante une approximation incluse dans le bassin de capture (approximation par l'intérieur). Nous donnons les preuves de convergence de ces algorithmes vers les vrais bassins de capture. Nous définissons également un contrôleur optimal qui permet de construire une trajectoire qui atteint la cible en un temps minimal, tout en restant dans K.
Dans les paragraphes suivants, on note pour simplifier $Capt(K, C, n) = Capt(K, C, n.dt)$ le bassin de capture au temps $t = n.dt$.

3.2.1 Algorithme d'approximation de bassins de capture et de fonction de temps minimal en utilisant un système auxiliaire

Dans cette section, nous proposons un algorithme d'approximation de bassins de capture en utilisant un système auxiliaire, basé sur l'algorithme 1.1 défini dans le chapitre 1. L'algorithme proposé utilise certaines spécificités du problème afin d'économiser du temps de calcul et de l'espace mémoire.

Considérons la dynamique F_C (équation (3.8)). On associe à l'espace des contraintes $\mathcal{H}$ une grille $\mathcal{H}_h$ telle que :

$$\forall (x, y) \in \mathcal{H}, \exists (x_h, y_h) \in \mathcal{H}_h \text{ tel que } d((x, y), (x_h, y_h)) \leq \alpha(h), \tag{3.11}$$

avec

$$\alpha(h) \to 0 \text{ quand } h \to 0. \tag{3.12}$$

On définit la correspondance $G_C : \mathcal{X} \times \mathbb{R}^+ \to \mathcal{X} \times \mathbb{R}^+$ telle que $G_C = 1 + F_C.dt$, la dynamique discrète associée à F_C :

$$(x^{n+1}, y^{n+1}) \in G_C(x^n, y^n) \text{ pour tout } n \geq 0. \tag{3.13}$$

L'algorithme d'approximation de noyau de viabilité décrit dans le chapitre 1 définit la suite $(\mathcal{H}_h^n)_n$, avec $\mathcal{H}_h^0 = \mathcal{H}_h$, telle que :

$$\mathcal{H}_h^{n+1} = \{(x_h, y_h) \in \mathcal{H}_h^n \text{ tels que } d(G_C(x_h, y_h), L_h^n) \leq \mu_C \alpha(h)\}, \tag{3.14}$$

avec μ_C la constante de Lipschitz de la correspondance F_C et $L_h^n = L(\mathcal{H}_h^n)$, L étant la fonction de discrimination. L'algorithme s'arrête lorsque $\mathcal{H}_h^{n+1} = \mathcal{H}_h^n$ et L_h^n est alors une approximation du noyau de viabilité du système étendu 3.8, et donc de l'épigraphe de la fonction de temps minimal ϑ_C^K.
On remarque dans l'inclusion différentielle (3.8) que la dynamique sur la dimension temps est particulière : la fonction est monotone et la variable $\frac{y(t)}{dt}$ prend uniquement ses valeurs dans $\mathbb{N}^+$. Cette propriété nous permet d'utiliser une discrétisation particulière de l'espace sur cette dimension afin de gagner de la mémoire et du temps de calcul.

3.2.1.1 Discrétisation de la dimension auxiliaire

Introduisons la grille $K_{hi} = \{(x_h, i.dt)\}$ pour redéfinir la grille $\mathcal{H}_h$ (figure 3.2) :

$$\mathcal{H}_h = \bigcup_{i=0}^{\mathbb{N}^+} K_{hi} \tag{3.15}$$

En réécrivant l'algorithme décrit dans le chapitre 1, on peut définir la suite $(\mathcal{H}_h^n)_n$:

$$\mathcal{H}_h^{n+1} = \bigcup_{i=0}^{\mathbb{N}^+} K_{hi}^{n+1} = \bigcup_{i=0}^{\mathbb{N}^+} \{(x_h, y_h) \in K_{hi}^n \text{ tels que } d(G_C(x_h, y_h), L_h^n) \leq \mu_C \alpha(h)\}, \tag{3.16}$$

avec $K_{hi}^0 = K_{hi}$.

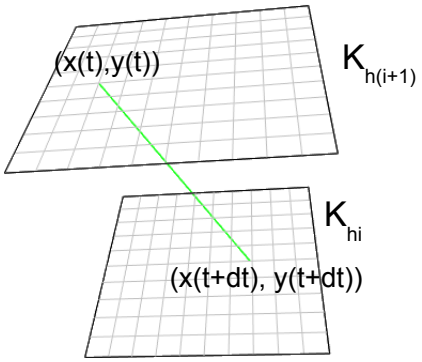


FIG. 3.2 – Grilles K_{hi} et $K_{h(i+1)}$. A partir d'un point $(x(t), y(t)) \in K_{h(i+1)}$, le point suivant $(x(t + dt), y(t + dt))$ est situé sur la grille $K_{h(i+1)}$ si $x(t) \in C$ et sur la grille K_{hi} sinon (cas représenté ici).

3.2.1.2 Construction incrémentale des grilles $(\mathcal{H}_h^n)_n$

A chaque itération n, on construit la fonction de discrimination L sur l'ensemble de la grille $\mathcal{H}_h$. On pose $L_h^0 = \mathcal{H}$. A l'initialisation, un point est viable à l'itération suivante si :

$$\begin{cases} (x_h, y_h) \in C & \text{si } y_h = 0, \\ d(G_C(x_h, y_h), L_h^0) \leq \mu_C \alpha(h) & \text{si } y_h > 0. \end{cases} \tag{3.17}$$

Aux itérations suivantes, avec la dynamique monotone sur la dimension temps, l'ensemble K_{hi}^{n+1} obtenu à l'étape $n + 1$ dépend uniquement de lui-même et de l'ensemble $K_{h(i-1)}^n$ défini à l'étape précédente :

$$\forall (x_h, y_h) \in K_{hi}^n, (x_h, y_h) \in K_{hi}^{n+1} \text{ si } \begin{cases} d(G_C(x_h, y_h), L_h^n \cap K_{hi}) \leq \mu_C \alpha(h) & \text{si } x_h \in C, \\ d(G_C(x_h, y_h), L_h^n \cap K_{h(i-1)}) \leq \mu_C \alpha(h) & \text{si } x_h \notin C. \end{cases} \tag{3.18}$$

Nous proposons de construire la grille $\mathcal{H}_h$ itérativement, en ajoutant peu à peu les sous-grilles K_{hi}, en fonction des résultats des itérations précédentes. On part de K_{h0} et on ajoute progressivement les sous-grilles K_{hi}, lorsque les différentes $K_{h(i-1)}$ ont été définies. Cette procédure présente un avantage majeur : le vecteur d'apprentissage pour le calcul des fonctions L_h^n n'est pas toujours constitué de l'ensemble des points de la grille, ce qui permet de diminuer le temps de calcul (le temps de calcul d'une fonction SVM est quadratique avec la taille du vecteur d'apprentissage).
On commence par travailler avec les points au temps $y_h = 0$. On pose $\mathcal{H}_{h0}^0 = K_{h0}$. Par définition, les points capturant au temps $y_h = 0$ sont uniquement ceux situés dans la cible. On peut donc écrire $\mathcal{H}_{h0}^1 = \{(x_h, y_h) \in \mathcal{H}_{h0}^0 \text{ tel que } x_h \in C\}$ et $\mathcal{H}_{h0}^2 = \{(x_h, y_h) \in \mathcal{H}_{h0}^1 \text{ tel que } x_h \in C\} = \mathcal{H}_{h0}^p, p \geq 1$. On note cet ensemble $\mathcal{H}_h^0$. On ajoute ensuite les points au temps $y_h = 1.dt$: $\mathcal{H}_{h1}^0 = \mathcal{H}_h^0 \cup K_{h1}$ et on définit

L^0_{h1}. On a donc $\mathcal{H}^1_{h1} = \left\{(x_h, y_h) \in \mathcal{H}^0_{h1} \text{ tel que } d((x_h, y_h), L(H^0_{h1})) \leq \mu_C \alpha(h)\right\}$. Or, si $x_h \in C$ alors $(x_h, y_h) \in \mathcal{H}^1_{h1}$ et si $x_h \notin C$, on teste si $d((x_h, y_h), L^0_h) \leq \mu_C \alpha(h)$. On a donc $\mathcal{H}^1_{h1} = \mathcal{H}^p_{h1}$, avec $p \geq 1$. A l'itération n, on connaît L^n_h et on veut calculer $\mathcal{H}^{n+1}_h$. On pose $\mathcal{H}^0_{h(n+1)} = \mathcal{H}^n_h \cup K_{h(n+1)}$. On peut écrire :

$$\mathcal{H}^1_{h(n+1)} = \left\{(x_h, y_h) \in \mathcal{H}^n_h \text{ tel que } d((x_h, y_h), L^0_{hn}) \leq \mu_C \alpha(h)\right\}. \tag{3.19}$$

Si $x_h \in C$ alors $(x_h, y_h) \in \mathcal{H}^1_{h(n+1)}$ et si $x_h \notin C$, on teste si $d((x_h, y_h), L_{hn^0}) \leq \mu_C \alpha(h)$. Or, $L^0_{hn} = L^n_h$. Donc, $\mathcal{H}^1_{h(n+1)} = \mathcal{H}^p_{h(n+1)}$, $p \geq 1$. Ainsi, connaissant L^n_h, une seule itération suffit pour déterminer L^{n+1}_h, avec $\mathcal{H}^{n+1}_h = \mathcal{H}^n_h \cup K_{hn}$.

Cette propriété peut également être utilisée pour déduire l'étiquette de certains points et ainsi économiser du temps de calcul. En effet, à chaque itération n, ce sont uniquement les points $(x_h, \tau) \in K_{hn}$ qui vont pouvoir changer d'étiquette. Pour les points tels que $y_h < n.dt$, il suffit de récupérer les résultats des itérations précédentes (avec la dynamique F_C, si un point atteint la cible sans sortir de K au temps $n.dt$, il l'atteint aussi au temps $(n+1).dt$).

3.2.1.3 Algorithme d'approximation de bassins de capture avec un système auxiliaire

En utilisant une discrétisation particulière et la dimension auxiliaire, et en construisant de façon incrémentale les différentes grilles $(\mathcal{H}^n_h)_n$, l'algorithme 1.1 peut être adapté à l'approximation de noyau de viabilité d'un système étendu (et de fonctions de temps minimal), afin de gagner de l'espace mémoire et du temps de calcul. L'algorithme 3.1 détaille la procédure.

Procédure 3.1 (Algorithme d'approximation de bassins de capture avec un système auxiliaire)
On définit itérativement les ensembles L^n_h tels que :

$$\begin{array}{l} \mathcal{H}^0_h = \{(x_h, y_h) \in K_{h0} \textit{ tels que } x_h \in C\} \\ L^n_h = L(\mathcal{H}^n_h) \\ \mathcal{H}^{n+1}_h = \mathcal{H}^n_h \bigcup \left\{(x_h, y_h) \in K_{h(n+1)} \textit{ tels que } d(G_C(x_h, y_h), L^n_h) \leq \mu_C \alpha(h)\right\} \end{array} \tag{3.20}$$

Si la procédure de discrimination respecte les conditions suivantes :

$$\forall x \in L^n_h,\, d(x, \mathcal{H}^n_h) \leq \lambda \alpha(h), \tag{3.21}$$

$$\forall x \in \mathcal{H} \backslash L^n_h,\, d(x, \mathcal{H}_h \backslash \mathcal{H}^n_h) \leq \alpha(h), \tag{3.22}$$

alors

$$\forall n,\, L^n_h \to Viab_{F_C}(\mathcal{H}) \textit{ lorsque } h \to 0. \tag{3.23}$$

3.2.2 Algorithme d'approximation de bassins de capture et de fonction de temps minimal dans l'espace d'état initial

3.2.2.1 Notations

De la même façon que pour l'algorithme d'approximation d'un noyau de viabilité en utilisant une méthode d'apprentissage supervisé, on discrétise le système dynamique dans le temps mais pas dans l'espace. Considérons la correspondance F :

$$F(x) = \left\{ \begin{array}{ll} \varphi(x, u) & \text{si } x \notin C \\ 0 & \text{si } x \in C \\ u \in U(x), & \end{array} \right. \tag{3.24}$$

et la correspondance $G = 1 + F.dt$ la correspondance discrète associée à F :

$$\begin{cases} x^{n+1} \in G(x^n) \\ x^0 = x_0. \end{cases} \tag{3.25}$$

On suppose que G est μ-Lipschitz. Soient K et C deux ensembles de $\mathcal{X}$, $C \subset K$, on recherche le bassin de capture du système. On discrétise K avec une grille K_h (équation (1.15)).

A chaque itération n, on définit un ensemble discret $C_h^n \subset C_h^{n-1} \subset K_h$, et un ensemble continu $L(C_h^n)$ qui est une généralisation de cet ensemble discret, et qui constitue l'approximation du bassin de capture au temps $n.dt$.

L'avantage de l'algorithme proposé ici est double :
- il permet de travailler dans un espace de dimension d (alors qu'on doit travailler en dimension $d+1$ si on veut utiliser l'algorithme présenté dans la section précédente) ;
- il est possible d'approcher le bassin de capture au temps $n.dt$ non seulement par l'extérieur, mais également par l'intérieur. L'approximation par l'intérieur permet de définir une procédure de contrôle qui garantit d'atteindre la cible. De plus, la comparaison des deux approximations (pour un exemple d'application donné) permet d'obtenir une estimation empirique de l'erreur d'approximation de l'algorithme.

3.2.2.2 Algorithme d'approximation par l'extérieur

Dans la première variante de l'algorithme, chaque approximation au temps $n.dt$ contient le vrai bassin de capture au temps $n.dt$ (comme pour l'algorithme d'approximation de noyau de viabilité). A chaque itération, on ajoute les points tels qu'il existe au moins un contrôle qui permette à la trajectoire de ne pas être « trop loin » de l'approximation à l'itération précédente.

A l'initialisation, on considère que seuls les points $x_h \in C$ capturent la cible. On ajoute ensuite progressivement les points pour lesquels il existe au moins une fonction de contrôle qui permette à la trajectoire de ne pas être « trop loin » de l'approximation à l'itération précédente. A chaque itération, on constitue une base d'apprentissage, composée des points x_h étiquetés $+1$ si le point capture la cible et -1 sinon. La fonction de discrimination l^{n+1} permet de définir le bassin de capture au temps $(n+1).dt$:

$$L(C_h^{n+1}) = \left\{ x \in K \text{ tels que } l^{n+1}(x) = +1 \right\}. \tag{3.26}$$

Procédure 3.2 (Algorithme d'approximation d'un bassin de capture par l'extérieur)
On définit itérativement les ensembles L_h^n tels que :

$$\begin{aligned} &L_h^0 = C \\ &C_h^{n+1} = \{x_h \in K_h \text{ tels que } d(G(x_h), L_h^n) \leq \mu\beta(h)\} \\ &L_h^{n+1} = L(C_h^{n+1}). \end{aligned} \tag{3.27}$$

Si la procédure de discrimination respecte les conditions suivantes :

$$\exists \lambda \geq 1 \text{ tel que } \forall x \in L_h^n,\ d(x, C_h^n) \leq \lambda\beta(h), \tag{3.28}$$

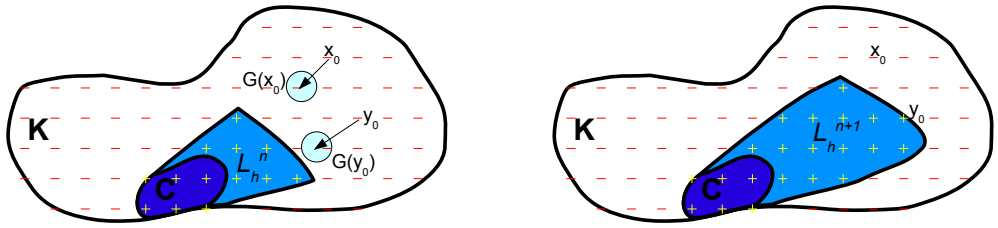
$$\forall x \in K \backslash L_h^n,\ d(x, K_h \backslash C_h^n) \leq \beta(h), \tag{3.29}$$

alors

$$\forall n, Capt_G(K, C, n) \subset L_h^n \text{ et} \tag{3.30}$$

$$L_h^n \to Capt_G(K, C, n) \text{ lorsque } h \to 0. \tag{3.31}$$

La figure 3.3 illustre le passage de l'itération n à l'itération $n+1$ de l'algorithme.



(a) (b)

FIG. 3.3 – Exemple de passage de l'itération n (figure (a)) à l'itération $n+1$ (figure (b)). Les approximations par l'extérieur du bassin de capture courant L_h^n et L_h^{n+1} sont représentées en bleu. L'espace des contraintes est délimité par la ligne continue et la cible C est représentée en bleu foncé. Les ensembles C_h^n et C_h^{n+1} sont représentés par les points « + » en jaune et les ensembles $K_h \backslash C_h^n$ et $K_h \backslash K_h^{n+1}$ par les points « - » en rouge. La grille K_h est composée des points C_h^n et $K_h \backslash C_h^n$ sur la figure (a) et des points C_h^{n+1} et $K_h \backslash C_h^{n+1}$ sur la figure (b). Les ensembles L_h^n et L_h^{n+1} sont des généralisations des ensembles C_h^n et C_h^{n+1}. A l'itération n, on teste pour tous les points $x_h \in K_h$ si $d(G(x_h), L_h^n) \leq \mu\beta(h)$. Si oui, alors $x_h \in C_h^{n+1}$ (point y_0 par exemple sur la figure), et sinon, $x_h \notin C_h^{n+1}$ (point x_0 par exemple).

3.2.2.3 Preuve de la convergence de l'algorithme d'approximation par l'extérieur

Les conditions que l'algorithme d'apprentissage doit respecter (équations (3.28) et (3.29)) sont les mêmes que celles requises pour l'algorithme d'approximation de noyau de viabilité, avec une condition plus stricte à l'extérieur de l'approximation. Notons que ces conditions peuvent être réécrites de la façon suivante : considérons $x \in K$,

$$\begin{array}{l} \text{si } \forall x_h \in K_h \cap B(x, \beta(h)), x_h \in C_h^n, \text{ alors } x \in L_h^n, \\ \text{si } \forall x_h \in K_h \cap B(x, \lambda\beta(h)), x_h \in K_h \backslash C_h^n, \text{ alors } x \in K \backslash L_h^n. \end{array} \tag{3.32}$$

La preuve de la convergence comporte trois étapes :

1. **Pour chaque itération n, le bassin de capture en temps $n.dt$ est inclus dans L_h^n.**
   Par définition, $Capt_G(K, C, 0) = C = L_h^0$. Supposons qu'à chaque étape n, $Capt_G(K, C, n) \subset L_h^n$. Considérons $x \in Capt_G(K, C, n+1)$.
   Par définition, $d(G(x), Capt_G(K, C, n)) = 0$, ce qui implique que pour tout point $x_h \in K_h \cap \boldsymbol{B}(x, \beta(h))$, $d(G(x_h), Capt_G(K, C, n)) \leq \mu\beta(h)$, car G est μ-Lipschitz. De plus, pour chaque $x_h \in K_h \cap \boldsymbol{B}(x, \beta(h))$, $d(G(x_h), L_h^n) \leq \mu\beta(h)$, car, par hypothèse, $Capt_G(K, C, n) \subset L_h^n$. On a donc $x_h \in C_h^{n+1}$. Par conséquent, $x \in L_h^{n+1}$ (à cause de la condition (3.32)).
   On peut donc conclure que $Capt_G(K, C, (n+1)) \subset L_h^{n+1}$.

2. **Pour chaque n, $L_h^n \to Capt_G(K, C, n)$ lorsque $h \to 0$.**
   Par définition, $Capt_G(K, C, 0) = C = L_h^0$. Supposons maintenant que, pour une valeur donnée de n, $L_h^n \to Capt_G(K, C, n)$ lorsque $h \to 0$.
   $Capt_G(K, C, n) \subset L_h^n$ signifie :

   $$\forall x \in K \text{ tel que } x \notin Capt_G(K, C, n), \exists h > 0 \text{ tel que } x \notin L_h^n.$$

   Considérons maintenant $x \in K$ tel que $x \notin Capt_G(K, C, (n+1))$.
   Ceci implique que $d(G(x), Capt_G(K, C, n)) > 0$. On peut choisir une valeur de h telle que $d(G(x), Capt_G(K, C, n)) > (1+\mu)\lambda\beta(h)$. Dans ce cas, pour chaque $x_h \in K_h \cap \boldsymbol{B}(x, \lambda\beta(h))$, $d(G(x_h), Capt_G(K, C, n)) > \lambda\beta(h)$, car G est μ-Lipschitz.
   Considérons maintenant n'importe quel $y_h \in K_h$ tel qu'il existe $x_h \in K_h \cap \boldsymbol{B}(x, \lambda\beta(h))$ avec

$d(y_h, G(x_h)) \leq \lambda\beta(h)$. On a $d(y_h, Capt_G(K, C, n)) > 0$, à cause de l'inégalité triangulaire. Avec l'hypothèse d'induction, et parce que l'ensemble des y_h est de taille finie, on peut trouver h tel que pour chaque y_h, $d(y_h, L_h^n) > 0$. Ceci implique que, pour chaque $y \in \cup_{x_h} G(x_h)$, tous les points de la grille plus proches de $\lambda\beta(h)$ de y, se situent à l'extérieur de L_h^n, et donc que $y \notin L_h^n$ (à cause de la condition 3.32). Ce qui implique que tous les points de la grille plus proches de $\lambda\beta(h)$ par rapport à x, sont situés à l'extérieur de L_h^{n+1}, et que donc $x \notin L_h^{n+1}$.
On a donc $L_h^{n+1} \rightarrow Capt_G(K, C, n+1)$ lorsque $h \rightarrow 0$.

3. **Conclusion.**
$Capt_G(K, C, n) \subset L_h^n$ et $L_h^n \rightarrow Capt_G(K, C, p)$ alors L_h^n est une approximation par l'extérieur du bassin de capture en temps fini $n.dt$, approximation qui tend vers le vrai bassin de capture lorsque la résolution h de la grille tend vers 0.

3.2.2.4 Algorithme d'approximation par l'intérieur

Dans la seconde variante de l'algorithme, le bassin de capture au temps $n.dt$ inclut son approximation. A chaque itération, l'algorithme ajoute les points tels qu'il existe un contrôle qui permette de rentrer « franchement » à l'intérieur de l'approximation précédente. La convergence de l'approximation par l'intérieur requiert une condition sur la dynamique : un point x à l'intérieur du bassin de capture $n+1$ doit être défini tel qu'il existe $y \in G(x)$ qui appartient à l'intérieur du bassin de capture n. De plus, les conditions sur la procédure de discrimination sont plus strictes à l'intérieur de l'approximation. On définit itérativement les ensembles L_h^n :

Procédure 3.3 (Algorithme d'approximation d'un bassin de capture par l'intérieur)
On définit itérativement les ensembles L_h^n tels que :

$$\begin{array}{l} L_h^0 = C, \\ C_h^{n+1} = \{x_h \in K_h \text{ tels que } \exists x \in G(x_h), d(x, K \backslash L_h^n) > \mu\beta(h)\}, \\ L_h^{n+1} = L(C_h^{n+1}) \end{array} \tag{3.33}$$

Si la procédure de discrimination respecte les conditions suivantes :

$$\forall x \in L_h^n, d(x, C_h^n) \leq \beta(h) \tag{3.34}$$

$$\exists \lambda \geq 1 \text{ tel que } \forall x \in K \backslash L_h^n, d(x, K_h \backslash C_h^n) \leq \lambda\beta(h), \tag{3.35}$$

et si la dynamique est définie telle que :

$$\forall x \in int(Capt_G(K, C, n+1)), \exists y \in G(x) \text{ tels que } d(y, K \backslash Capt_G(K, C, n)) > 0, \tag{3.36}$$

alors

$$\forall n, L_h^n \subset Capt_G(K, C, n), \tag{3.37}$$

$$\forall n, L_h^n \rightarrow Capt_G(K, C, n) \text{ lorsque } h \rightarrow 0. \tag{3.38}$$

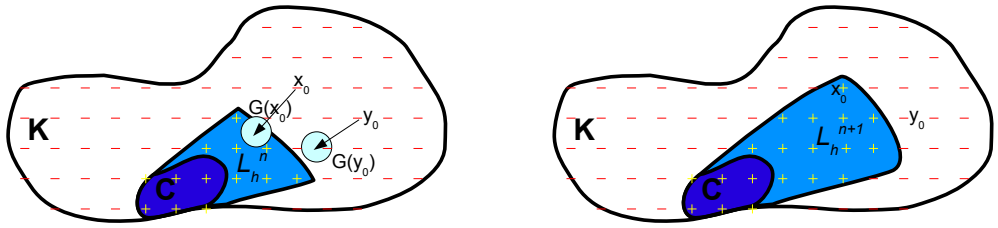
La figure 3.4 illustre le passage de l'itération n à l'itération $n+1$.

3.2.2.5 Preuve de la convergence de l'algorithme d'approximation par l'intérieur

Comme pour la première variante de l'algorithme, les conditions (3.34) et (3.35) peuvent être réécrites :

$$\begin{array}{l} \text{si } \forall x_h \in K_h \cap B(x, \lambda\beta(h)), x_h \in C_h^n, \text{ alors } x \in L_h^n, \\ \text{si } \forall x_h \in K_h \cap B(x, \beta(h)), x_h \in K_h \backslash C_h^n, \text{ alors } x \in K \backslash L_h^n. \end{array} \tag{3.39}$$

La preuve de la convergence comporte trois étapes :



(a) (b)

FIG. 3.4 – Exemple de passage de l'itération n (figure (a)) à l'itération $n+1$ (figure (b)). Les approximations du bassin de capture courant L_h^n et L_h^{n+1} sont représentées en bleu. L'espace des contraintes est délimité par la ligne continue et la cible C est représentée en bleu foncé. Les ensembles C_h^n et C_h^{n+1} sont représentés par les points « + » en jaune et les ensembles $K_h \backslash C_h^n$ et $K_h \backslash C_h^{n+1}$ par les points « - » en rouge. Les ensembles L_h^n et L_h^{n+1} sont des généralisations des ensembles C_h^n et C_h^{n+1}. A l'itération n, on teste pour tous les points $x_h \in K_h$ si $\exists x \in G(x_h)$ tel que $d(x, K \backslash L_h^n) > \mu\beta(h)$. Si oui, alors $x_h \in C_h^{n+1}$ (point x_0 par exemple sur la figure), et sinon, $x_h \notin C_h^{n+1}$ (point y_0 par exemple).

1. **Pour chaque** n**,** $L_h^n \subset Capt_G(K, C, n)$**.**
   Par définition de l'algorithme, $L_h^0 = C = Capt_G(K, C, 0)$.
   Supposons que $L_h^n \subset Capt_G(K, C, n)$ et considérons que $x \in L_h^{n+1}$.
   Grâce à la condition (3.34), on peut écrire :

$$\exists x_h \in C_h^{n+1} \text{ tel que } \|x - x_h\| \leq \beta(h). \tag{3.40}$$

   Par construction de l'algorithme :

$$\exists y \in G(x_h), d(y, K \backslash L_h^n) > \mu\beta(h). \tag{3.41}$$

   Comme G est μ-Lipschitz :

$$G(x) \subset (G(x_h) + \mu\beta(h)\boldsymbol{B}(0,1)). \tag{3.42}$$

   On a donc :

$$\exists y_1 \in G(x) \text{ tel que } d(y_1, K \backslash L_h^n) > 0 \Rightarrow y_1 \in L_h^n. \tag{3.43}$$

   Comme $L_h^n \subset Capt_G(K, C, n)$, $y_1 \in Capt_G(K, C, n)$. On a donc $x \in Capt_G(K, C, n+1)$.
   Par conséquent, $L_h^{n+1} \subset Capt_G(K, C, n+1)$.

2. **Pour chaque** n**,** $L_h^n \rightarrow Capt_G(K, C, n)$**.**
   Par construction de l'algorithme, $C = L_h^0 \rightarrow Capt_G(K, C, 0)$.
   Supposons que $L_h^n \rightarrow Capt_G(K, C, n)$ lorsque $h \rightarrow 0$. Avec cette hypothèse, et comme $L_h^n \subset Capt_G(K, C, n)$, on peut écrire :

$$\forall x \in int(Capt_G(K, C, n)), \exists h > 0 \text{ tel que } x \in int(L_h^n). \tag{3.44}$$

   Considérons $x \in int(Capt_G(K, C, n+1))$. On peut choisir h tel que $d(x, K \backslash Capt_G(K, C, n+1)) > \lambda\beta(h)$. Avec un tel choix, pour chaque $x_h \in K_h \cap \boldsymbol{B}(x, \lambda\beta(h))$, $d(x_h, K \backslash Capt_G(K, C, n+1)) > 0$, et il existe $y \in G(x_h) \in K$ tel que $d(y, K \backslash Capt_G(K, C, n)) > 0$.
   Avec l'hypothèse d'induction, et parce que $K_h \cap \boldsymbol{B}(x, \lambda\beta(h))$ est fini, il existe h tel que pour tout $x_h \in K_h \cap \boldsymbol{B}(x, \beta(h))$, $\exists y \in G(x_h)$ tel que $d(y, K \backslash L_h^n) > \mu h$. Dans ce cas, chaque $x_h \in C_h^{n+1}$, donc $d(x, K \backslash L_h^{n+1}) > 0$ (à cause de la condition (3.39)).
   On a donc $L_h^{n+1} \rightarrow Capt_G(K, C, n+1)$ lorsque $h \rightarrow 0$.

3. **Conclusion.**
   $L_h^n \subset Capt_G(K, C, n)$ et $L_h^n \to Capt_G(K, C, n)$ donc L_h^n est une approximation par l'intérieur du bassin de capture en temps fini $n.dt$, approximation qui tend vers le vrai bassin de capture lorsque la résolution de la grille tend vers 0.

3.3 Contrôleur optimal

Le but du contrôleur optimal est de choisir une fonction de contrôle qui permette au système d'atteindre la cible en un temps minimal, sans violer les contraintes de viabilité. L'idée est de choisir le contrôle qui pilote le système afin qu'il traverse les différents ensembles L_h^n dans un ordre décroissant, en suivant la direction de la plus grande pente.

Afin d'utiliser les ensembles L_h^n, on suppose que l'on peut calculer la distance algébrique d'un point x à la frontière ∂L_h^n de L_h^n. On suppose que cette distance est négative à l'extérieur de l'ensemble et positive à l'intérieur. Notons $d(x, \partial L_h^n)$ cette distance.

Procédure 3.4 (Algorithme de contrôle optimal)
Considérons $x \in L_h^T$. Notons $n(x)$ défini comme suit :

$$n(x) = \arg\max_n x \in L_h^n. \tag{3.45}$$

et $u^*(x)$ est tel que :

$$u^*(x) = \arg\max_{u \in U(x)} d(x + \varphi(x, u)dt, \partial L_h^{n(x)-1}). \tag{3.46}$$

La fonction de contrôle $u^*(x)$ converge vers une politique de contrôle qui minimise le temps maximal d'atteinte de la cible, lorsque h et dt tendent vers 0.

La preuve que le système atteindra la cible sans jamais quitter K est directe : lorsque h et dt tendent vers 0, les frontières ∂L_h^n tendent vers les contours de la fonction valeur associée à la solution HJB du problème et la procédure choisit le contrôle qui maximise cette fonction valeur.

3.4 Minimisation d'une fonction de coût

Les algorithmes donnés dans ce chapitre peuvent être étendus aux problèmes de minimisation d'une fonction positive non nulle de coût [Cardaliaguet *et al.*, 1997] :

$$\int_0^{\vartheta_C^K(x_0)} \ell(x(t), u(t))dt, \tag{3.47}$$

avec $x(0) = x_0$ et $x(t) \in K, \forall t \in \left[0, \vartheta_C^K(x_0)\right]$. Dans ce cas, on considère le système dynamique étendu F_E :

$$(x(t), c(t))' \in F_E(x(t), c(t)) = \begin{cases} F(x(t)) \times \{-\ell(x(t), u(t))\} & \text{si } x \notin C \\ 0 & \text{si } x \in C. \end{cases} \tag{3.48}$$

avec c une variable représentant le coût. Dans ce cas, comme dans le cas du bassin de capture, on peut écrire :

$$Viab_{F_E}(\mathcal{H}) = \varepsilon pi(\vartheta_C^K), \tag{3.49}$$

avec $\mathcal{H} = K \times \mathbb{R}^+$. On peut donc utiliser les algorithmes classiques de viabilité pour approcher $\varepsilon pi(\vartheta_C^K)$. Afin de pouvoir utiliser les algorithmes présentés dans la section 3.4, on transforme le système (3.49)

afin d'obtenir une variable de coût qui prenne uniquement ses valeurs dans $\mathbb{R}^+$.
On définit dc la variation de coût que l'on veut observer à chaque itération $\ell(x,u)dt = dc$ et on note, pour $\ell(x,u) > 0$, $dt^* = \frac{dc}{\ell(x,u)}$. Dans ce cas, on définit la correspondance $G : \mathcal{X} \to \mathcal{X}$, $G = 1 + F_E.dc$, le système dynamique discret associé à F_E :

– si $x(t) \notin C$:

$$(x(t+dc), c(t+dc)) = \begin{cases} x(t) + (\frac{\varphi(x(t),u(t))}{\ell(x,u)})dc = x(t) + \varphi(x(t),u(t))dt^* \\ c(t) - 1.dc \end{cases} \tag{3.50}$$

– si $x(t) \in C$:

$$(x(t+dc), c(t+dc)) = \begin{cases} x(t) \\ c(t) \end{cases} \tag{3.51}$$

On se ramène ainsi à un cas classique d'approximation d'un bassin de capture, excepté qu'on fixe à chaque itération la variation de coût dc au lieu de la variation de temps dt.

Dans le cas de problèmes de minimisation d'une fonction de coût positive et non nulle (la quantité $\ell(x,u)$ doit toujours être supérieure à 0), on peut utiliser les algorithmes présentés dans la section 3.2, soit en utilisant un système dynamique auxiliaire, soit directement dans l'espace d'état initial.

Chapitre 4

Approcher un bassin de capture en utilisant des SVMs

Dans ce chapitre, nous montrons comment les SVMs peuvent être utilisées pour approcher des bassins de capture et quelles sont les conditions qu'elles doivent remplir pour respecter les conditions d'application du théorème. Nous proposons ensuite un contrôleur optimal qui permet de définir une trajectoire minimisant la fonction valeur.

Dans la section 4.1, nous utilisons les SVMs pour approcher un bassin de capture en utilisant un système auxiliaire. Dans la section 4.2, nous considérons les SVMs comme méthode d'apprentissage pour approcher des bassins de capture dans l'espace d'état initial, et nous étudions les conditions d'application. Dans la section 4.3, nous proposons une procédure de contrôle optimal utilisant les SVMs. Nous illustrons les différents algorithmes sur des exemples simples d'application dans la section 4.4.

Sommaire

4.1 Algorithme d'approximation de bassins de capture en utilisant un système auxiliaire avec des SVMs

4.1.1 Respect des conditions d'application de l'algorithme

Approcher un bassin de capture revient à approcher le noyau de viabilité d'un système étendu. L'algorithme d'approximation de bassins de capture en utilisant un système auxiliaire est basé sur l'algorithme d'approximation de noyau de viabilité décrit dans le chapitre 1. L'algorithme utilise seulement certaines spécificités du problème afin d'économiser du temps de calcul et de l'espace mémoire. Le respect des conditions d'application de l'algorithme, avec $\lambda = 1$, est discuté dans le chapitre 2. On définit les approximations L_h^n comme suit :

$$L_h^n = \{(x,y) \in \mathcal{H}^n \text{ tels que } f_n(x,y) \geq 0\}, \tag{4.1}$$

avec $\mathcal{H}^n = \bigcup_{i=0}^{n} \{(x, i.dt)\}$.

4.1.2 Méthode d'optimisation pour trouver un contrôle qui permet de capturer la cible

Utiliser des SVMs pour approcher un noyau de viabilité fournit une expression analytique de la frontière f_n, ce qui permet d'utiliser une méthode d'optimisation pour trouver un contrôle qui permette de rester dans le noyau de viabilité à l'instant suivant.

Considérons tout d'abord une optimisation sur un pas de temps. La procédure est semblable à celle décrite dans le chapitre 2. Notons le couple $(x_s(x,y,u_1), y_s(x,y,u_1))$ le successeur de (x,y), avec :

$$x_s(x,y,u_1) = \begin{cases} x + \varphi(x,u_1)dt & \text{si } x \notin C \\ x & \text{si } x \in C, \end{cases} \tag{4.2}$$

et

$$y_s(x,y,u_1) = \begin{cases} y - dt & \text{si } x \notin C \\ y & \text{si } x \in C. \end{cases} \tag{4.3}$$

Le contrôle optimal u_1^* est celui qui résout le problème suivant (le choix du contrôle optimal ne dépend pas explicitement du temps) :

$$u_1^* = \arg \max_{u_1 \in U(x)} f_n(x_s(x, y-dt, u_1), y - dt) \tag{4.4}$$

Comme décrit dans le chapitre 2, il est possible de déterminer une séquence de contrôles optimaux sur j pas de temps avec un temps de calcul raisonnable, afin d'améliorer la qualité de l'approximation. Définissons le couple $(x_s(x,y,u_1,...,u_j), y_s(x,y,u_1,...,u_j))$, le couple issu de (x,y) atteint après j pas de temps :

$$x_s(x,y,u_1,...,u_j) = \begin{cases} x_s(x,y,u_1,...,u_{j-1}) + \varphi(x_s(x,y,u_1,...,u_{j-1}), u_j)dt \\ \qquad \text{si } x_s(x,y,u_1,...,u_{j-1}) \notin C \\ x_s(x,y,u_1,...,u_{j-1}) \\ \qquad \text{si } x_s(x,y,u_1,...,u_{j-1}) \in C, \end{cases} \tag{4.5}$$

et

$$y_s(x,y,u_1,...,u_j) = \begin{cases} y_s(x,y,u_1,...,u_{j-1}) - dt & \text{si } x_s(x,y,u_1,...,u_{j-1}) \notin C \\ y_s(x,y,u_1,...,u_{j-1}) & \text{si } x_s(x,y,u_1,...,u_{j-1}) \in C. \end{cases} \tag{4.6}$$

A partir d'un point initial (x, y) et si le point $x_s(x, y-jdt, u_1, ..., u_j), y-jdt$ n'appartient pas au noyau de viabilité courant, on réalise une descente de gradient, jusqu'à trouver la suite de contrôles $(u_j)_j$ qui maximise la fonction $f_n(x_s(x, y - jdt, u_1, ..., u_j), y - jdt)$:

$$(u_1^*, ..., u_j^*) = \arg \max_{(u_1,...,u_j) \in U(x)} f_n(x_s(x, y - jdt, u_1, ..., u_j), y - jdt), \tag{4.7}$$

et on note $(x_j, y_j)^*$ le point atteint après j pas de temps $(x_s(x, y, u_1, ..., u_j), y_s(x, y, u_1, ..., u_j))$.

Posons $n_T = \frac{T}{dt}$, le nombre d'itérations nécessaires pour obtenir le bassin de capture au temps T. Lorsque $(n_T \mod j) \neq 0$, le nombre de pas à la dernière incrémentation de la grille doit être adapté afin de respecter la contrainte sur la dimension temps $y \in [0; T]$.

4.1.3 Algorithme général d'approximation de bassins de capture en utilisant un système auxiliaire avec les SVMs

Dans le chapitre 2, nous avons montré que les SVMs sont une méthode de discrimination adaptée au problème d'approximation de noyau de viabilité. On applique les SVMs sur un ensemble d'apprentissage $\mathcal{S}$, qui contient l'ensemble des points de la grille considérée, auxquels on associe l'étiquette $+1$ si le point appartient à l'approximation courante du bassin de capture et l'étiquette -1 sinon. Nous utilisons également la règle suivante pour déterminer si un point appartient à l'approximation courante du bassin de capture (itération $n + 1$) :

$$(x_h, y_h) \in \mathcal{H}_h^{n+1} \text{ si } f_n((x_h, y_h)^*) \geq -1. \tag{4.8}$$

Cette règle permet de simplifier les calculs (comme discuté dans le chapitre 1).

L'algorithme 4.1 détaille la procédure d'approximation d'un noyau de viabilité d'un système auxiliaire, en considérant une optimisation sur j pas de temps. On fixe un temps maximal de capture, T. On suppose ici que $(n_T \mod j) = 0$ afin de simplifier la présentation de l'algorithme. L'approximation du noyau de viabilité du système étendu $L_h^{n_T}$ en temps fini T est donnée par :

$$L_h^{n_T} = \{(x, y) \in \mathcal{H} \text{ tel que } f_{n_T}(x, y) \geq -1\}. \tag{4.9}$$

4.1.4 Contrôleur

Un bassin de capture est l'ensemble des états initiaux à partir desquels il existe au moins une évolution qui respecte à chaque instant les contraintes de viabilité et qui atteint la cible en un temps *prescrit* ou *minimal*.
De la même façon que pour l'approximation d'un noyau de viabilité classique, on définit un contrôleur qui construit une trajectoire restant indéfiniment dans l'espace des contraintes. Dans le chapitre 1, nous avons introduit le contrôleur lourd dont le principe est de changer le contrôle uniquement lorsque le système approche de la frontière du noyau de viabilité, c'est-à-dire lorsque la viabilité est en péril. Ce contrôleur peut être adapté dans le cas de problèmes de bassin de capture. Dans ce cas, on résout un problème d'atteinte de la cible en un temps *prescrit*.
De plus, les algorithmes de viabilité permettent d'obtenir les même résultats que la programmation dynamique [Frankowska, 1989], avec des contraintes sur l'espace d'état. La recherche de solutions optimales revient à rechercher des évolutions viables d'un système dynamique étendu. A partir de l'approximation du bassin de capture, on peut ainsi définir un contrôleur en temps *minimal*, qui donne une approximation du temps minimal de capture et détermine une trajectoire qui atteint la cible.
Nous détaillons tout d'abord la procédure de contrôleur lourd dans le cas de bassins de capture, puis nous proposons une procédure de contrôle en temps minimal.

Algorithme 4.1 Algorithme général d'approximation d'un bassin de capture avec un système auxiliaire

$\mathcal{S} \leftarrow \emptyset$
$n = 0$

// Initialisation
Pour tout $(x_h, y_h) \in K_{h0}$ **Faire**
  Si $x_h \in C$ **Alors**
    $\mathcal{S} \leftarrow \mathcal{S} \bigcup ((x_h, y_h), +1)$
  Sinon
    $\mathcal{S} \leftarrow \mathcal{S} \bigcup ((x_h, y_h), -1)$
  Fin si
Fin pour

// Itérations suivantes, pour j pas de temps
Faire
  Calculer $f_n(x, y)$ à partir de $\mathcal{S}$
  Pour tout $i = 1$ à j **Faire**
    Pour tout $(x_h, y_h) \in K_{h(n+i)}$ **Faire**
      Si $f_n(((x_h)_i), (y_h)_i)^*) \geq -1$ et $((x_h)_i, (y_h)_i)^* \in \mathcal{H}$ **Alors**
        $\mathcal{S} \leftarrow \mathcal{S} \bigcup ((x_h, y_h), +1)$
      Sinon
        $\mathcal{S} \leftarrow \mathcal{S} \bigcup ((x_h, y_h), -1)$
      Fin si
    Fin pour
  Fin pour
  $n = n + j$
Jusqu'à $n = n_T$
Calculer $f_n(x, y)$ à partir de $\mathcal{S}$
Renvoyer $f_n(x, y)$

4.1.4.1 Contrôleur lourd

Partant d'un point initial (x_0, y_0), on veut définir une trajectoire qui reste indéfiniment dans le bassin de capture, ce qui signifie que l'on cherche une suite de contrôles qui vont permettre au système d'atteindre la cible en un temps $y \leq y_0$, tout en restant toujours dans K.
L'algorithme d'approximation de noyau de viabilité en utilisant des SVMs donne une approximation contenue dans le vrai noyau. On définit la procédure suivante :

Procédure 4.1 (Contrôleur lourd de viabilité en utilisant une fonction SVM)
Pour Δ un réel positif donné, on définit :

$$A_\Delta = \{(x, y) \text{ tel que } f(x, y) \geq \Delta\}, \tag{4.10}$$

avec $A_\Delta \subset Capt_{G_C}(K, C)$. A partir d'un état initial $(x_0, y_0) \in A_\Delta$ et d'un contrôle $u_0 \in U(x)$ choisi aléatoirement, la procédure s'arrête si $x_n \in C$ ou, si $x_n \notin C$, associe un contrôle u_{n+1} à l'itération $(n+1)$:
– si $(x_n + \varphi(x_n, u_n)dt, y_n - dt) \in A_\Delta$, on garde le même contrôle $u_{n+1} = u_n$;
– sinon, $u_{n+1} = \arg\max_{u \in U(x)} f_n((x_n + \varphi(x_n, u)dt, y_n - dt))$.

De la même façon que pour le contrôleur lourd défini dans le chapitre 2, on peut définir des contrôleurs plus ou moins prudents, en anticipant sur plusieurs pas de temps (voir l'algorithme 2.2).

Il n'existe aucune garantie que l'ensemble A_Δ soit inclus dans $Capt_{G_C}(K, C)$ et que la procédure garde le système dans $\mathcal{H}$, tout en atteignant la cible en un temps $y \leq y_0$.

4.1.4.2 Contrôleur en temps minimal

Partant d'un état initial x_0, on veut piloter le système afin qu'il atteigne la cible en un temps minimal. La première étape est donc de déterminer le temps minimal nécessaire pour atteindre la cible. Notons $\widehat{\vartheta}_C^K(x_0)$ l'approximation du temps minimal de capture du point x_0 :

$$\widehat{\vartheta}_C^K(x_0) = \inf \{y \geq 0 | (x_0, y) \in A_\Delta\}. \tag{4.11}$$

On utilise ensuite la procédure décrite dans la section précédente afin de déterminer la séquence de contrôle $(u_n)_n$ telle que, partant du point $(x_0, \widehat{\vartheta}_C^K(x_0))$, on atteigne la cible sans quitter l'espace des contraintes $\mathcal{H}$. Cependant, comme l'approximation des bassins de capture est plus grande que les vrais résultats, il n'y a aucune garantie que la procédure atteigne la cible.

4.2 Algorithme d'approximation de bassins de capture dans l'espace d'état initial avec des SVMs

Dans le chapitre 1, nous avons proposé un algorithme qui approche un bassin de capture dans l'espace d'état initial en utilisant une méthode d'apprentissage. Nous proposons dans cette section d'utiliser les SVMs comme méthode d'apprentissage. Nous étendons ensuite la procédure au cas où l'on travaille sur plusieurs pas de temps. Finalement, nous décrivons l'algorithme général d'approximation, en utilisant les SVMs.

4.2.1 Respect des conditions d'application de l'algorithme

Les algorithmes 3.2 et 3.3 convergent vers le bassin de capture si la procédure d'apprentissage respecte les conditions suivantes :

– pour l'algorithme par l'extérieur :

$$\begin{array}{l} \exists \lambda \geq 1 \text{ tel que } \forall x \in L_h^n,\, d(x, C_h^n) \leq \lambda \beta(h), \\ \forall x \in K \backslash L_h^n,\, d(x, K_h \backslash C_h^n) \leq \beta(h). \end{array} \tag{4.12}$$

– pour l'algorithme par l'intérieur :

$$\begin{array}{l} \forall x \in L_h^n,\, d(x, C_h^n) \leq \beta(h), \\ \exists \lambda \geq 1 \text{ tel que } \forall x \in K \backslash L(C_h^n),\, d(x, K_h \backslash C_h^n) \leq \lambda \beta(h). \end{array} \tag{4.13}$$

Afin de respecter ces conditions, la procédure de classification ne doit pas faire d'erreur de classification et ne pas avoir d'irrégularités supérieures à $\beta(h)$. Nous n'avons aucune démonstration rigoureuse que les SVMs respectent ces conditions mais il existe des arguments qui laissent penser que ce sera généralement le cas, en choisissant correctement les paramètres de la fonction (paramètres C et σ dans les équations (2.9) et (2.11)). En posant $\lambda = 1$, les conditions (4.12) et (4.13) sont identiques et sont les mêmes que dans l'algorithme général présenté dans le chapitre 1. Le respect des conditions du théorème, avec $\lambda = 1$, est discuté dans le chapitre 2.

A chaque itération de l'algorithme, on construit un vecteur d'apprentissage $\mathcal{S}$, constitué de l'ensemble des points x_h de K_h, associés à l'étiquette $+1$ si $x_h \in C_h^n$ et -1 sinon. On calcule une fonction SVM à partir de $\mathcal{S}$ pour obtenir la fonction de classification f^n et on définit l'ensemble L_h^n comme suit :

$$L_h^n = \{x \in K \text{ tel que } f^n(x) \geq 0\}. \tag{4.14}$$

4.2.2 Méthode d'optimisation pour trouver un contrôle qui permet de capturer la cible

4.2.2.1 Optimisation sur un pas de temps

De la même façon que dans le chapitre 2, utiliser des SVMs pour définir l'approximation courante d'un bassin de capture permet d'utiliser une méthode d'optimisation pour chercher un contrôle viable. On évite ainsi l'augmentation exponentielle du temps de calcul lorsque la dimension de l'espace des contrôles augmente. A la différence de l'algorithme décrit dans le chapitre 2 où l'on cherche s'il existe au moins un contrôle qui va permettre de rester à l'intérieur de l'approximation courante du noyau, on recherche ici s'il existe au moins un contrôle qui va permettre à un point d'atteindre l'approximation du bassin de capture au temps précédent.
Par construction, on a à l'itération $n+1$, $C_h^{n+1} \subset C_h^n$ (les points qui peuvent atteindre la cible avant le temps $n.dt$ sans quitter K vont pouvoir également l'atteindre avant le temps $(n+1).dt$). Ainsi, il suffit de tester uniquement les points $x_h \in K_h \backslash C_h^n$ pour définir l'ensemble C_h^{n+1} : maximiser $f^n(x_h)$ fournit un contrôle qui permet à la trajectoire $x_h(.)$ d'atteindre L_h^n, si un tel contrôle existe. A partir d'un point x, on réalise une descente de gradient jusqu'à trouver la suite de contrôle $(u_j)_j$ qui maximise la fonction $f_n(x)$:

$$u^* = \arg \max_{u \in U(x)} f_n(x + \varphi(x, u)dt), \tag{4.15}$$

et on note x^* le point atteint en utilisant le contrôle u^*.
A la première itération, la fonction f_0 n'est pas définie analytiquement. En pratique, on pose :

$$C_h^0 = \{x_h \in K_h \text{ tel que } x_h \in C\}, \tag{4.16}$$

et on calcule la première fonction SVM à partir de l'ensemble d'apprentissage $\mathcal{S}$, constitué des points de K_h avec une étiquette $+1$ si $x_h \in C_h^0$ et une étiquette -1 sinon.

4.2.2.2 Optimisation sur j pas de temps

De la même façon que décrit dans le chapitre 2, il est possible de déterminer une séquence de j pas de temps à chaque itération. Dans ce cas, à l'itération n, L_h^n est l'approximation (par l'intérieur ou par l'extérieur) de $Capt_G(K, C, n.j.dt)$. On définit itérativement la trajectoire partant du point x :

$$\begin{cases} t(x, u_1) = x + \varphi(x, u_1)dt, \\ t(x, u_1, ..., u_j) = t(x, u_1, ..., u_{j-1}) + \varphi(t(x, u_1, ..., u_{j-1}), u_j)dt. \end{cases} \tag{4.17}$$

De la même façon que pour l'optimisation à un pas de temps, on utilise une descente de gradient pour maximiser la fonction $f_{n.j}(t(x, u_1, ..., u_j))$, ce qui fournit un contrôle, s'il existe, qui permet à la trajectoire d'atteindre L_h^n.

Posons $n_T = \frac{T}{dt}$, le nombre d'itérations nécessaires pour obtenir le bassin de capture au temps T. Lorsque $(n_T \mod j) \neq 0$, le nombre de pas à la dernière incrémentation de la grille doit être adapté afin de respecter la contrainte sur la dimension temps $y \in [0; T]$.

4.2.3 Simplification de l'algorithme

Dans le chapitre 2, nous avons remarqué que le calcul de la distance $\mu\beta(h)$ dans les algorithmes 3.2 et 3.3 n'est pas directe à calculer. De plus, cette opération tend à produire des approximations prudentes et nous avons constaté, pour certains exemples au moins, qu'elle augmente l'effet diffusif. On peut utiliser une règle plus simple pour définir les ensembles L_h^n :

– pour l'approximation par l'extérieur

$$C_h^{n+1} = \{x_h \in K_h \text{ tel que } f^n(x_h^*) \geq -\delta \text{ et } (x_h^*) \in K\}, \tag{4.18}$$

– pour l'approximation par l'intérieur

$$C_h^{n+1} = \{x_h \in K_h \text{ tel que } f^n(x_h^*) > \delta \text{ et } (x_h^*) \in K\}. \tag{4.19}$$

Dans les équations (4.18) et (4.19), on choisit $\delta = 1$ car cela limite la définition de l'approximation à une des marges de la SVM. Pour l'approximation par l'intérieur, les points qui sont vecteurs de support avec une étiquette -1 sont situés à l'intérieur de l'approximation et proches de la frontière de la SVM. Pour l'approximation par l'extérieur, l'approximation n'inclut pas les points qui sont vecteurs de support avec une étiquette $+1$. Cette procédure tend à satisfaire une condition plus stricte que (4.12) et (4.13), mais qui ne la garantit pas.

4.2.4 Erreur empirique d'approximation

Nous avons vu dans le chapitre 3 que l'aire entre les deux approximations (par l'extérieur et l'intérieur) permet d'obtenir, pour un exemple donné, une estimation informatique de l'erreur d'approximation. Cependant, l'aire entre ces deux ensembles n'est pas directe à calculer. On choisit une méthode plus simple et facile à mettre en oeuvre : on sait que l'approximation par l'intérieur (notée I_h^p) est incluse dans l'approximation par l'extérieur (notée E_h^p), et $I_h^p \subset Capt_F(K, C) \subset E_h^p$. Ainsi, sur un exemple donné, on dénombre l'ensemble des points $x_h \in K_h$ tels que $x_h \in E_h^p$ et $x_h \notin I_h^p$, ce qui donne une approximation grossière de l'aire entre les deux ensembles. Afin de supprimer l'effet taille de la grille, on divise ensuite le total obtenu par le nombre de points contenus dans l'approximation par l'extérieur, afin d'obtenir une estimation empirique de l'erreur d'approximation ϵ sur un exemple donné :

$$\epsilon = \frac{Card(\{x_h \in E_h^p \text{ et } x_h \notin I_h^p\})}{Card(\{x_h \in E_h^p\})}. \tag{4.20}$$

4.2.5 Algorithme général d'approximation de bassins de capture avec des SVMs

L'algorithme 4.2 décrit l'algorithme général d'approximation d'un bassin de capture en temps fini T. La procédure est détaillée ici dans le cas d'une approximation par l'extérieur mais le cas par l'intérieur peut en être facilement déduit : au lieu de tester si $f_n(x_h) \geq -1$ (ou $f_n(x_h^*) \geq -1$), on teste si $f_n(x_h) > 1$ (ou $f_n(x_h^*) > 1$). On se fixe un temps maximal de capture $T \in \mathbb{R}^+$, et on suppose que $(n_T \mod j) = 0$ afin de simplifier la présentation de l'algorithme.

Algorithme 4.2 Algorithme général d'approximation par l'extérieur d'un bassin de capture dans l'espace d'état initial

$\mathcal{S} \leftarrow \emptyset$
$n = 0$

// Initialisation
Pour tout $x_h \in K_h$ **Faire**
  Si $x_h \in C$ **Alors**
    $\mathcal{S} \leftarrow \mathcal{S} \bigcup (x_h, +1)$
  Sinon
    $\mathcal{S} \leftarrow \mathcal{S} \bigcup (x_h, -1)$
  Fin si
Fin pour

// Itérations suivantes, pour j pas de temps
Faire
  Calculer $f_n(x)$ à partir de $\mathcal{S}$
  Pour tout $x_h \in K_h$ **Faire**
    Si $f_n(x_h) \geq -1$ **Alors**
      $\mathcal{S} \leftarrow \mathcal{S} \bigcup (x_h, +1)$
    Sinon
      Si $(f_n(x_h^*) \geq -1)$ et $(x_h^* \in K)$ **Alors**
        $\mathcal{S} \leftarrow \mathcal{S} \bigcup (x_h, +1)$
      Sinon
        $\mathcal{S} \leftarrow \mathcal{S} \bigcup (x_h, -1)$
      Fin si
    Fin si
  Fin pour
  Calculer $f_n(x)$ à partir de $\mathcal{S}$
  $\mathcal{S} \rightarrow \emptyset$
  $n = n + j$
Jusqu'à $n = n_T$
Renvoyer $f_n(x)$

4.3 Approximation du contrôle optimal en utilisant les SVMs

Dans le chapitre 1, nous avons vu que la fonction de temps minimal ϑ_C^K peut être approchée par les fonctions L_h^n obtenues grâce à l'algorithme d'approximation par l'intérieur, et qui sont les fonctions valeurs associées au problème discrétisé. Dans cette section, nous détaillons la procédure qui permet d'approcher un contrôle optimal à partir des approximations L_h^n obtenues avec les SVMs.

On part d'un point x_0 tel que $\vartheta_C^K(x_0) < \infty$ pour lequel il existe un seul contrôle optimal $u(.)$. On note $(u_n^*)_n$ le contrôle optimal discret associé à $\vartheta_C^K(x_0)$. La procédure 3.4 définit le contrôle u_i^* qui

maximise la distance entre le point suivant et l'approximation du bassin de capture au temps $n(i)-1$:

$$u_i^* = \arg\max_{u\in U(x_i)} a(x_i+\varphi(x_i,u)dt, \partial L_h^{n(i)-1}), \tag{4.21}$$

où $a(x,\partial L_h^n)$ représente la distance algébrique entre le point x et la frontière de L_h^n. Cependant, cette distance n'est pas directe à calculer lorsque L_h^n est définie par une fonction SVM. On choisit d'utiliser une méthode plus rapide et plus simple à mettre en oeuvre pour calculer $(u_i^*)_i$.

Procédure 4.2 (Contrôle optimal en utilisant des SVMs)
On pose $x_0 \in L_h^T$, le point initial. Pour chaque point x_i de la trajectoire, on calcule $n(i)$, la valeur maximale de n telle que $x_i \in L_h^n$. Si $n(i)>0$, on définit u_i^* comme suit :

$$u_i^* = \arg\max_{u\in U(x_i)} f^{n(i)-1}(x_i+\varphi(x_i,u)dt). \tag{4.22}$$

Ainsi, on choisit la séquence de contrôle qui permet de piloter le système afin qu'il traverse les différentes frontières de L_h^n, en suivant la plus grande pente par rapport à f^n.

Si l'approximation a été réalisée en utilisant plusieurs pas de temps, la procédure doit être légèrement adaptée. A la première itération, on recherche le nombre de pas de temps $j_0 \leq j$ nécessaires pour que le point $t(x_0,u_1^*,...,u_{j0}^*)$ atteigne la première frontière $L_h^{n(i)-1}$:

$$\begin{array}{l} \arg\max_{u_1,...,u_{j_0}\in U(x)} f^{n(i)-1}(t(x_0,u_1,...,u_{j_0})) \in L_h^{n(i)-1} \text{ et} \\ \arg\max_{u_1,...,u_{j_0-1}\in U(x)} f^{n(i)-1}(t(x_0,u_1,...,u_{j_0-1})) \notin L_h^{n(i)-1}. \end{array} \tag{4.23}$$

Ensuite, on reprend l'algorithme 4.2, en remplaçant $x_i+\varphi(x_i,u)dt$ par $t(x_i,u_1,...,u_j)$.

4.4 Exemples d'application des algorithmes d'approximation de bassins de capture

Dans cette section, nous illustrons les algorithmes décrits dans ce chapitre sur 3 exemples d'application : le problème de Zermelo, d'une petite cible et de la voiture sur la colline.

4.4.1 Problème de Zermelo

4.4.1.1 Définition du problème

Le problème étudié ici est dérivé du fameux problème de Zermelo. Le but est de piloter un navire dans une rivière afin qu'il atteigne une île en un temps minimal, sans s'échouer sur les bancs de sable. La description du problème est tirée de [Cardaliaguet *et al.*, 1997] et [Bokanowski *et al.*, 2006].
Le problème peut être défini comme suit : l'état du système (x,y) représente la position du navire dans la rivière, où le courant, qui décroît lorsqu'on s'approche des bords de la rivière, est donné par $f(x,y)$:

$$f(x,y) = \left\{1-ay^2\right\} \times \{0\} \text{ (a = 0,1).} \tag{4.24}$$

L'ensemble des contraintes K représente la rivière navigable et C l'île à atteindre. A chaque instant, le capitaine peut contrôler l'accélération du navire u et sa direction θ. Le système en temps discret, avec un intervalle de temps dt, peut être défini comme suit :

$$\left\{\begin{array}{l} x(t+dt) = x(t) + (1-ay(t)^2 + u\cos\theta)dt \\ y(t+dt) = y(t) + (u\sin\theta)dt. \end{array}\right. \tag{4.25}$$

Posons $K = [-6; 2] \times [-2; 2]$ et $C = \boldsymbol{B}(0; 0, 44)$, où $\boldsymbol{B}$ est la boule unitaire dans $\mathbb{R}^2$. Les contrôles doivent appartenir à un intervalle donné : $u \in [0; 0, 44]$ et $\theta \in [0; 2\pi]$. Le navire doit atteindre la cible avant le temps $T = 7$.

Dans un premier temps, nous allons rechercher l'ensemble des états initiaux tel que le système atteigne la cible sans quitter K, puis partant d'une position donnée, nous définirons un contrôle optimal.

4.4.1.2 Résolution du problème en utilisant un système auxiliaire

On définit le système auxiliaire en temps discret suivant :

$$(x(t+dt), y(t+dt), \tau(t+dt)) = \begin{cases} (x(t+dt), y(t+dt)) \times \{-1\} \text{ si } (x(t), y(t)) \notin C \\ (x(t), y(t)) \times \{0\} \text{ sinon.} \end{cases} \tag{4.26}$$

où τ représente le temps qui s'écoule à chaque itération et $(x(t+dt), y(t+dt))$ est défini par l'équation (4.25). L'espace des contraintes en dimension 3 est donné par $\mathcal{H} = K \times [0; T]$. Pour cet exemple simple, il est possible de dériver le vrai bassin de capture en temps fini T du système, afin de le comparer à l'approximation donnée par l'algorithme d'approximation.

La figure 4.1 présente un exemple de déroulement de l'algorithme, avec l'ajout successif des sous-grilles K_{hi}. Les paramètres utilisés pour cette simulation sont précisés dans le tableau 4.1 (simulation 1). La figure 4.2 présente un exemple d'approximation du noyau de viabilité du système étendu, et donc de l'épigraphe de la fonction de temps minimal (simulation 2 dans le tableau 4.1).

	simulation 1	**simulation 2**	**simulation 3**	**simulation 4**
Dimension	3	3	2	2
Pas de la grille h	0, 1	0, 015	0, 05	0, 015
Nombre total de points	1815	105861	441	5041
dt	0, 5	0, 35	0, 1	0, 1
Nombre de pas	2	2	7	7
C	30000	30000	30000	30000
σ^2	0, 1	0, 05	0, 05	0, 05
Nombre d'itérations	8	10	10	10
Nombre final de SV	72	2938	13 (extérieur) 17 (intérieur)	20 (extérieur) 22 (intérieur)
Temps	40 sec	1 j 28 min	1 min (extérieur) 1 min (intérieur)	19 min (extérieur) 30 min (intérieur)

TAB. 4.1 – Paramètres et résultats des simulations pour le problème de Zermelo.

4.4.1.3 Résolution du problème dans l'espace d'état initial

La figure 4.3 compare les résultats obtenus avec les deux variantes de l'algorithme dans l'espace d'état initial, pour deux tailles de grille différentes. Le tableau 4.1 détaille les paramètres des simulations (simulation 3 pour le cas avec une grille de 21 points par dimension et simulation 4 pour 71 points par dimension). Afin de quantifier l'erreur empirique de l'approximation sur cet exemple, on calcule la différence entre le nombre de points capturant la cible dans les deux résultats, relativement au nombre de points contenu dans l'approximation par l'extérieur. Pour l'approximation utilisant 21 points par dimension, la différence est de $\epsilon = 20\%$, tandis qu'elle est uniquement de $\epsilon = 6\%$ avec la grille contenant 71 points par dimension.

En comparant ces résultats avec l'approximation obtenue en utilisant un système auxiliaire, on remarque que les approximations dans l'espace d'état initial sont meilleures. Cependant, la discrétisation

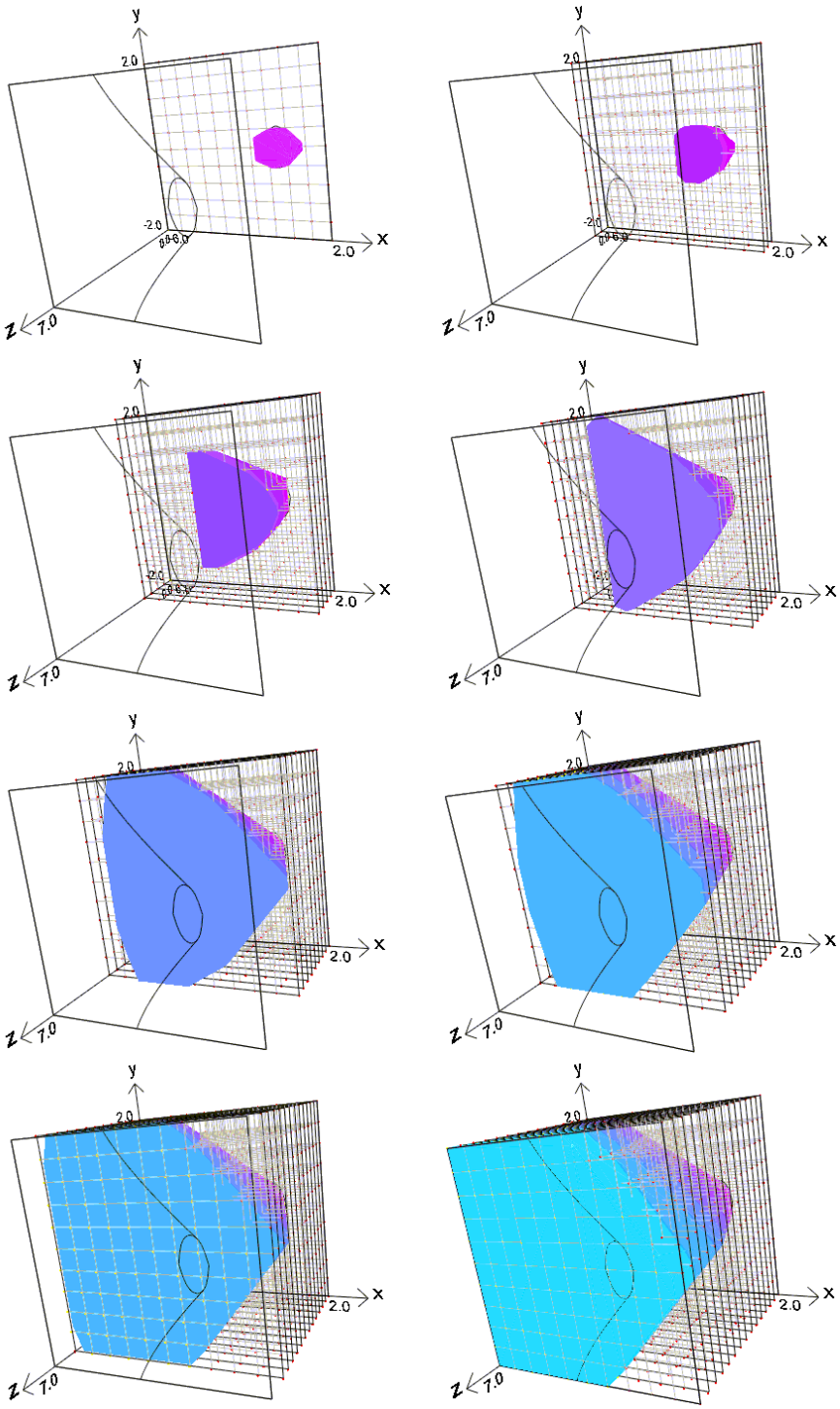


FIG. 4.1 – Exemple de l'approximation progressive du bassin de capture pour le problème Zermelo en dimension 3. L'axe horizontal représente la position $x(t)$, l'axe vertical $y(t)$ et l'axe z le temps $\tau(t)$. Le noyau de viabilité est représenté en dégradé rose-bleu. La ligne épaisse noire délimite la frontière du vrai bassin de capture au temps $T = 7$. Les grilles en gris clair représentent les différentes sous-grilles K_{hi}.

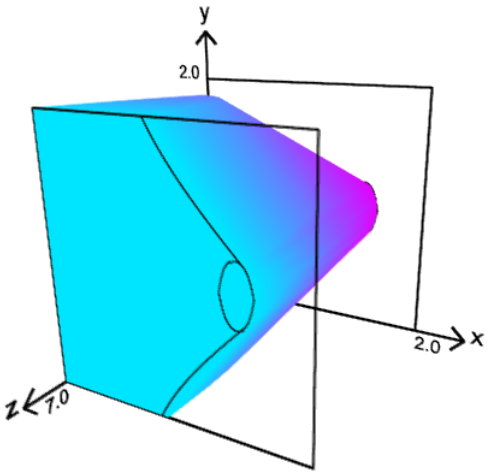
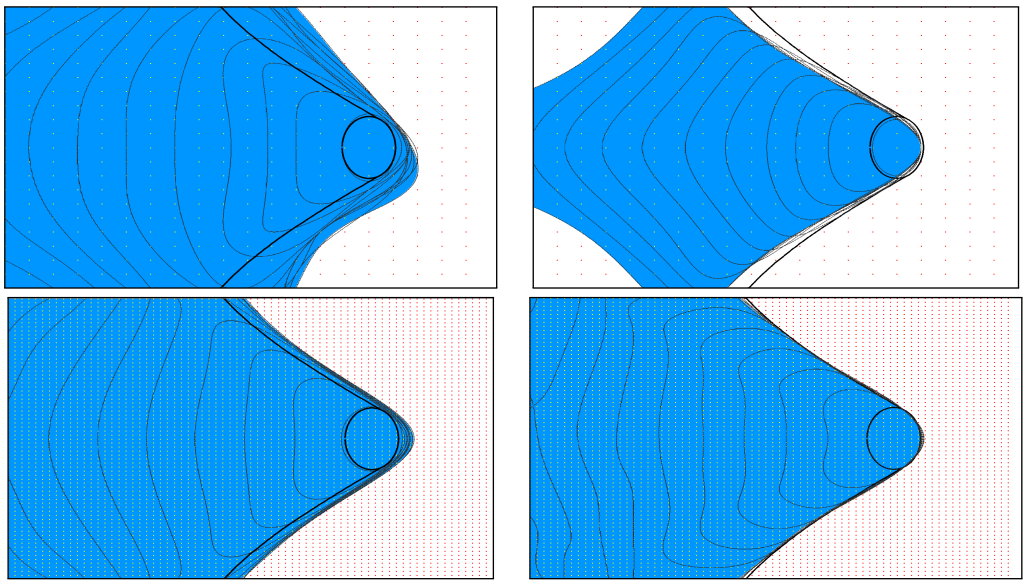


FIG. 4.2 – Exemple d'approximation du bassin de capture pour le problème Zermelo en dimension 3. L'axe horizontal représente la position $x(t)$, l'axe vertical $y(t)$ et l'axe z le temps $\tau(t)$. Le noyau de viabilité du système étendu (4.26) est représenté en dégradé rose-bleu.



(a) (b)

FIG. 4.3 – Approximation (a) par l'extérieur (b) par l'intérieur du bassin de capture du problème Zermelo. L'axe horizontal représente la position $x(t)$ et l'axe vertical $y(t)$. K est le rectangle qui limite les points de la grille. Le bassin de capture est représenté en bleu. La ligne épaisse noire délimite la frontière du vrai noyau. Les différentes lignes de niveaux représentent les contours de la fonction valeur approchée. En haut, la grille contient 21 points par dimension et en bas, 71 points.

du temps dt n'est pas la même pour les deux approximations : la discrétisation est plus grossière avec un système auxiliaire, car le nombre de points de la grille augmente avec cette valeur de dt. Nous avons donc choisi une valeur de dt plus élevée dans ce cas et les deux résultats ne sont donc pas tout à fait comparables.
On peut également comparer ces résultats avec ceux obtenus dans [Bokanowski *et al.*, 2006], qui ont

approché le bassin de capture du système de Zermelo en utilisant les mêmes paramètres, et comparé leurs résultats avec ceux obtenus par l'algorithme de Saint-Pierre [Saint-Pierre, 1994]. Avec une grille de 20 points par dimension, leur approximation est plus proche que celle obtenue avec l'algorithme utilisé ici. Ils ont également testé leur algorithme sur une grille contenant 100 points par dimension : les approximations données par l'algorithme Ultra-Bee et celles de la figure 4.3 (avec 71 points par dimension) sont sensiblement les mêmes, et bien meilleures que celles obtenues avec l'algorithme de Saint-Pierre. L'avantage principal de l'algorithme Ultra-Bee pour approcher des noyaux de viabilité et des bassins de capture est qu'il utilise un schéma non diffusif et donne donc des approximations précises. L'algorithme décrit dans la section 4.2 montre environ les mêmes performances sur cet exemple, avec une grille d'environ 70 points par dimension.

4.4.1.4 Contrôle optimal

La figure 4.4 représente un exemple de trajectoire optimale, à partir du point $x_0 = (-5,2;1,4)$, calculée à partir de l'approximation par l'intérieur (qui seule garantit d'atteindre la cible). La trajectoire permet au navire d'atteindre l'île, en un temps égal à 6,5.

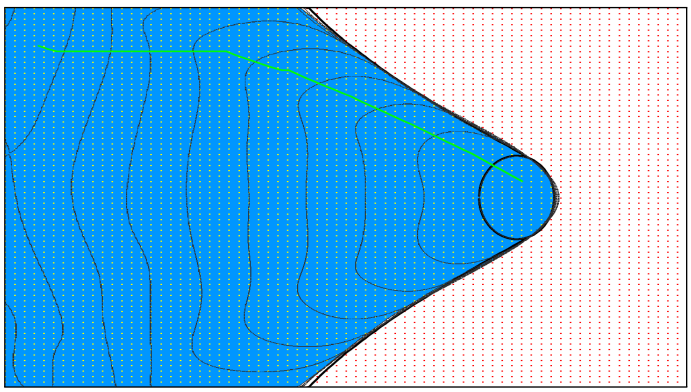


FIG. 4.4 – Exemple de trajectoire (en vert) en utilisant le contrôleur optimal pour le problème Zermelo. L'approximation est la même que dans la figure 4.3 (b), avec 71 points par dimension.

4.4.2 Petite cible

Dans un système en deux dimensions (x, y), on considère le problème d'atteinte d'une petite cible $C = [0, 0]$. Le système doit rester dans un espace $K = [-1; 1] \times [-1; 1]$. La dynamique en temps discret est décrite par l'équation (4.27) :

$$\begin{cases} x(t+dt) = x(t) + y(t)dt \\ y(t+dt) = y(t) + u(t)dt, \end{cases} \tag{4.27}$$

avec le contrôle $u(t) \in [-1; 1]$.

On approche le bassin de capture en temps fini T du système, avec $T = 1$. Il est possible de dériver la solution, afin de pouvoir comparer les résultats théoriques et ceux obtenus avec les différentes approximations.

4.4.2.1 Discrétisation

Dans ce problème, la cible est le point $C = (0, 0)$. Pour les deux algorithmes d'approximation du bassin de capture (avec un système auxiliaire ou dans l'espace d'état initial), à la première itération, le

bassin de capture au temps $\tau = 0$ contient uniquement le point cible $x_h = (0, 0)$. Les SVMs fournissent ensuite une approximation du bassin de capture au temps $\tau = 0$. Cependant, si le point cible n'appartient pas à K_h ou $\mathcal{H}_h$, le vecteur d'apprentissage initial est constitué uniquement de points d'étiquette -1. Or, pour obtenir une fonction SVM, il est nécessaire d'avoir au moins un point de chaque étiquette. Il est donc nécessaire de choisir une discrétisation qui définit une grille contenant le point cible.

4.4.2.2 Résolution du problème en utilisant un système auxiliaire

On définit le système auxiliaire suivant :

$$\begin{cases} x(t+dt) = x(t) + y(t)dt \\ y(t+dt) = y(t) + u(t)dt \\ \tau(t) = \tau(t) - dt \text{ si } (x(t), y(t)) \in C, \tau(t) \text{ sinon.} \end{cases} \tag{4.28}$$

Approcher le bassin de capture du système revient à approcher le noyau de viabilité du système (4.28). La figure 4.5 présente un exemple d'approximation du noyau de viabilité. Les paramètres utilisés pour cette simulation sont donnés dans le tableau 4.2 (simulation 1).

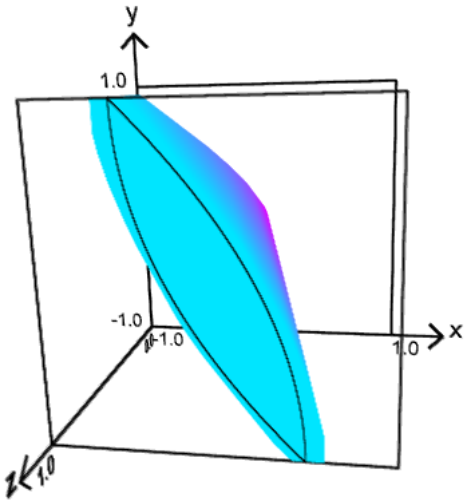


FIG. 4.5 – Exemple d'approximation du bassin de capture pour le problème de la petite cible en dimension 3. L'axe horizontal représente la dimension $x(t)$, l'axe vertical $y(t)$ et l'axe z le temps $\tau(t)$. Le noyau de viabilité est représenté en dégradé rose-bleu. La ligne noire délimite la frontière du vrai bassin de capture au temps $\tau = 1$.

4.4.2.3 Résolution du problème dans l'espace d'état initial

Pour le problème de la petite cible, il n'est pas possible d'approcher le bassin de capture en utilisant l'algorithme par l'intérieur : en effet, dans l'équation (4.19), on définit l'ensemble des points capturant la cible à l'itération suivante en considérant la frontière définie par $f(x) > 1$. Or, à la première itération, le vecteur d'apprentissage contient uniquement un point capturant la cible (la cible), qui sera forcément vecteur de support de la première SVM (la définition d'une fonction SVM contient au moins un point positif et un point négatif), et on aura $f_0(C) = +1$. La fonction $f_0(x) > 1$ ne sera donc pas définie sur l'ensemble K, et on ne pourra pas approcher le bassin de capture en utilisant l'approximation par l'intérieur.

	simulation 1	**simulation 2**
Dimension	3	2
Pas de la grille h Nombre total de points	$0,02$ 54621	$0,02$ 2601
dt	$0,05$	$0,025$
Nombre de pas	2	8
C	30000	30000
σ^2	$0,05$	$0,05$
Nombre d'itérations	10	6
Nombre final de SV	210	26
Temps	15 min	1 min

TAB. 4.2 – Paramètres et résultats des simulations pour le problème de la petite cible.

En ce qui concerne l'approximation par l'extérieur, la frontière est définie par $f(x) \geq -1$ (équation (4.18)). L'algorithme fonctionne donc normalement.

La figure 4.6 présente un exemple d'approximation des bassins de capture du problème dans l'espace d'état initial. Les paramètres utilisés pour cette simulation sont résumés dans le tableau 4.2 (simulation 2). En comparant cette approximation avec celle obtenue en utilisant un système auxiliaire, les résultats sont sensiblement les mêmes. Si on compare ces résultats avec ceux donnés dans [Bokanowski *et al.*, 2006], on constate que, avec le même nombre de points par dimension, les approximations données ici et celles obtenues en utilisant le schéma Ultra-bee sont sensiblement les mêmes. Par contre, ces approximations sont beaucoup plus fines que celle obtenue en utilisant l'algorithme de Saint-Pierre.

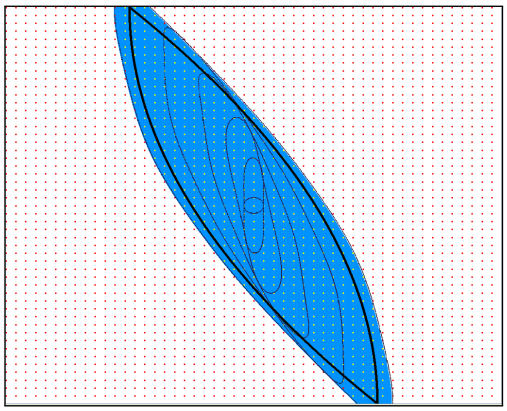


FIG. 4.6 – Approximation par l'extérieur du bassin de capture pour le problème de la petite cible. L'axe horizontal représente la dimension x et l'axe vertical la dimension y. Le bassin de capture est représenté en bleu. La grille contient 51 points par dimension. Le vrai bassin de capture est représenté par les lignes épaisses.

4.4.3 Voiture sur la colline

4.4.3.1 Définition du problème

On considère ici le fameux problème de la voiture sur la colline (en anglais « car on the hill »). La description du problème est tirée de [Moore et Atkeson, 1995].

Le système est défini en deux dimensions : la position de la voiture x et sa vitesse x'. La voiture peut être contrôlée à l'aide d'une variable de contrôle continue en une dimension : l'accélération a. Le but est de garder la voiture dans un ensemble de contraintes données, tout en atteignant la cible aussi vite que possible (figure 4.7). Le but est d'atteindre le haut de la colline, avec une vitesse faible : $C \in$

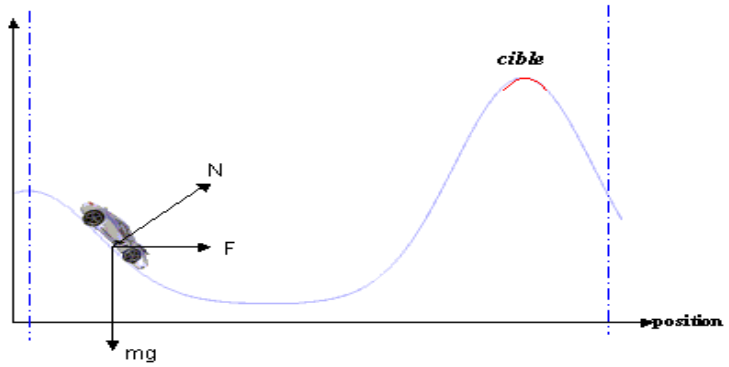


FIG. 4.7 – Représentation du système de la voiture sur la colline.

$[0,5;0,7] \times [-0,1;0,1]$. La voiture doit rester dans l'ensemble de contraintes $K = [-1;1] \times [-2;2]$. L'accélération est limitée $a \in [-4;4]$.
L'évolution de la vitesse x' est fonction de la position x :

$$x'' = \frac{a}{\sqrt{1+(H'(x))^2}} - \frac{gH'(x)}{1+H'(x)^2}, \tag{4.29}$$

avec $g = 9,81$ et :

$$H'(x) = \begin{cases} x^2 + x & \text{si } x < 0 \\ x/\sqrt{1+5x^2} & \text{si } x \geq 0. \end{cases} \tag{4.30}$$

On considère le système dynamique en temps discret suivant :

$$\begin{cases} x(t+dt) = x(t) + x'(t)dt \\ x'(t+dt) = x'(t) + x''(t)dt. \end{cases} \tag{4.31}$$

4.4.3.2 Résolution du problème dans l'espace d'état initial

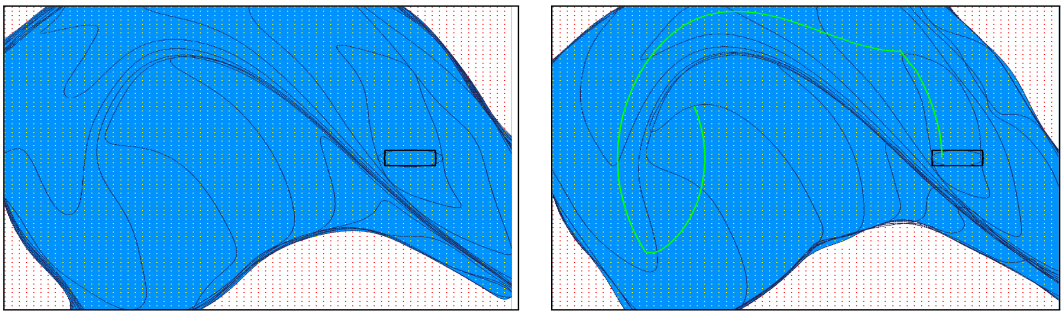
La figure 4.8 présente l'approximation du bassin de capture du système et des contours de la fonction valeur, en utilisant les deux variantes de l'algorithme d'approximation dans l'espace d'état initial. Les paramètres utilisés pour ces simulations sont donnés dans le tableau 4.3 (simulation 1). La différence entre l'approximation par l'extérieur et celle par l'intérieur est inférieure à $\epsilon = 7\%$.

4.4.3.3 Contrôleur optimal

Si la vitesse est trop faible lorsque la voiture est dans la côte de la colline, même l'accélération maximale ne suffit pas pour remonter la pente. La voiture doit redescendre pour accumuler assez de vitesse et remonter ensuite jusqu'en haut de la colline. La figure 4.8(b) présente une trajectoire qui permet d'atteindre la cible en un temps minimal, à partir du point $x_0 = (-0,1;0,4)$.

	simulation 1
Dimension	2
Pas de la grille h Nombre total de points	$0,015$ 5041
dt	$0,025$
Nombre de pas	8
C	30000
σ^2	$0,05$
Nombre d'itérations	17
Nombre final de SV	41 (extérieur) 43 (intérieur)
Temps	10 min (extérieur) 22 min (intérieur)

TAB. 4.3 – Paramètres et résultats des simulations pour le problème de la voiture sur la colline.



(a) (b)

FIG. 4.8 – Approximation (a) par l'extérieur (b) par l'intérieur du bassin de capture du problème de la voiture sur la colline. L'axe horizontal représente la position $x(t)$ et l'axe vertical la vitesse $x'(t)$. K est le rectangle qui limite les points de la grille. Le bassin de capture est représenté en bleu. Les différentes lignes de niveaux représentent les contours de la fonction valeur approchée. La grille contient 71 points par dimension.

Chapitre 5

Calculer la résilience d'un système en utilisant des SVMs

Dans ce chapitre, nous proposons un algorithme de calcul des valeurs de résilience, algorithme basé sur l'algorithme d'approximation de bassins de capture dans l'espace d'état initial. L'algorithme est ensuite illustré sur un modèle d'eutrophisation des lacs.

Dans la section 5.1, nous considérons le calcul des valeurs de résilience comme un cas particulier de minimisation d'une fonction de coût. Après avoir défini la résilience d'un système, nous proposons un algorithme d'approximation des valeurs de résilience, basé sur l'algorithme d'approximation des bassins de capture dans l'espace d'état initial. Nous introduisons également rapidement un algorithme simple qui permet de calculer le noyau de viabilité et les valeurs de résilience en présence d'incertitudes. Pour terminer, nous illustrons cet algorithme sur un problème en 3 dimensions d'eutrophisation des lacs dans la section 5.2

Sommaire

5.1 Minimisation d'une fonction de coût : application au problème du calcul de la résilience

Dans cette section, nous présentons un cas particulier de minimisation d'une fonction de coût : le cas du calcul de la résilience.

5.1.1 Définition de la résilience dans le formalisme de la viabilité

En général, la résilience est définie comme la capacité d'un système à conserver certaines de ses propriétés d'intérêt malgré des chocs ou perturbations. En écologie, le terme résilience est employé lorsque l'on parle de la durabilité d'un écosystème. Holling [Holling, 1973] a mis en avant deux aspects de la durabilité, et proposé deux définitions : l'« engineering resilience » et l'« ecological resilience » (termes introduits dans [Holling, 1996]). L'« engineering resilience » mesure le temps de retour d'un système à un équilibre global, après avoir subi une perturbation [Pimm, 1984]. L'« ecological resilience » mesure l'intensité maximale que le système peut absorber sans changer de comportement. Gunderson and Holling [Gunderson et Holling, 2002] ont proposé d'utiliser le concept de résilience pour définir des politiques d'action durables. Selon eux, gérer un système écologique et économique en accord avec le principe du développement durable est agir dans le but de préserver sa résilience.

Martin [Martin, 2004 ; Martin, 2005] a proposé une définition de la résilience proche de l'interprétation de Holling de l'« ecological resilience ». Elle est basée sur la théorie de la viabilité, avec les méthodes pour calculer des valeurs de résilience et définir des politiques d'action durables. La résilience est alors définie comme l'inverse du coût de restauration d'une certaine propriété d'intérêt perdue après une perturbation. La définition du coût a une part politique, dépendante du point de vue du manager. Il est très souvent défini comme un coût cumulé le long d'une évolution du système durant une certaine période : par exemple, le temps passé par le système privé de sa propriété d'intérêt ou le déficit accumulé tout au long de cette période. Lorsque la propriété d'intérêt est maintenue, le coût est nul et la résilience infinie ; lorsqu'il n'existe aucune politique d'action qui permet de restaurer le système, le coût est infini et la résilience nulle ; lorsque la propriété d'intérêt est perdue mais peut être restaurée, la valeur de la résilience et le coût sont finis et peuvent être calculés. Cette définition inclut également la détermination de politiques d'action qui vont permettre de restaurer la propriété perdue, en fonction des valeurs de la résilience.

Selon Martin [Martin, 2004], et en suivant les recommandations de [Ludwig *et al.*, 1997] et [Carpenter *et al.*, 2001], les concepts de la résilience basés sur la théorie de la viabilité dépendent :

– de la dynamique du système et des contrôles

$$x(t)' \in F(x(t)) = \{\varphi(x(t), u(t)) | u(t) \in U(x(t))\}\,; \tag{5.1}$$

– de la propriété d'intérêt, définie comme l'espace des contraintes : le système doit rester dans l'espace des contraintes K ;
– du temps maximal considéré $T \in \mathbb{R}^+$;
– des fonctions de coût $\lambda(x, u)$, qui servent à évaluer la résilience, et qui associent à une trajectoire $x(.)$ un coût non nul lorsque $x(t) \notin K$ et nul lorsque $x(t) \in K$;
– des perturbations D qui associent à un état x un ensemble $D(x)$ l'ensemble des états atteints après une perturbation.

La première question est, étant donné un ensemble des contraintes K et un état initial x_0, peut-on définir une politique d'action qui permette de conserver la propriété d'intérêt indéfiniment ? Ce qui peut être réécrit dans le formalisme de la viabilité par : le point initial x_0 est-il viable ? Pour répondre à cette question, on approche le noyau de viabilité du système (5.1) dans K et on regarde si $x_0 \in L_h^p$, avec L_h^p l'approximation du noyau de viabilité :

– si $x_0 \in Viab(K)$, cela signifie qu'il existe une politique d'action qui permette de conserver la propriété d'intérêt : le coût est alors nul et la résilience infinie ;

– si $x_0 \notin Viab(K)$, le système est condamné à perdre sa propriété d'intérêt. On veut alors déterminer une politique d'action qui permet de restaurer, si possible, la propriété d'intérêt avant le temps T, tout en minimisant le coût de restauration.

Une situation de crise correspond à une situation où le système sort de l'espace des contraintes K. Ces contraintes peuvent représenter une situation de « bonne santé », et le système ne s'effondre pas forcément lorsqu'il quitte K. Le temps de crise mesure le temps pendant lequel le système est à l'extérieur de K, que ce soit de manière réversible ou non. [Doyen et Saint-Pierre, 1997] introduisent la fonction de temps de crise minimal, qui associe à chaque état du système le temps minimal passé dans une situation de crise. Dans le cas de la résilience, on adapte la notion de temps de crise : on ne mesure plus le temps passé en dehors de l'espace des contraintes mais un coût lié à la trajectoire lorsqu'elle est en dehors de K.
Notons $C_K^T(x)$ la fonction qui associe à la trajectoire issue de x son coût minimal :

$$C_K^T(x) = \inf \int_0^\infty \lambda(x(.), u(.))dt. \tag{5.2}$$

Martin [Martin, 2004] montre que l'épigraphe de la fonction C_K^T est le noyau de viabilité d'un système auxiliaire F_E, défini dans un nouvel espace des contraintes $\mathcal{H} = \mathcal{X} \times \mathbb{R}^+$:

$$(x(t), c(t))' \in F_E(x(t), c(t)) = \begin{cases} F(x(t)) \times \{-\lambda(x(t), u(t))\} & \text{si } x \notin K \\ F(x(t)) \times \{0\} & \text{si } x \in K. \end{cases} \tag{5.3}$$

Ainsi, on peut calculer les fonctions de coût minimal en utilisant un algorithme de calcul de noyau de viabilité. A partir de là, on distingue deux cas :

– si le coût minimal d'une trajectoire partant du point x est inférieur à ∞, il existe au moins une politique d'action qui permet au système d'être restauré, et la résilience est non nulle ;
– si le coût est infini, le système ne peut être restauré et la résilience est nulle.

Dans le premier cas, on peut donc calculer les valeurs de la résilience $R_{K,D}^T$. Sa valeur dépend de la fonction de coût minimal et de l'ensemble des perturbations envisagées. Elle est définie comme l'inverse du coût maximal de restauration parmi tous les états possibles partant de x après une perturbation :

$$R_{K,D}^T = \frac{1}{\max_{y \in D(x)} C_K^T(y)}. \tag{5.4}$$

5.1.2 Algorithme de calcul de la résilience dans l'espace d'état initial

Le problème du calcul de valeur de résilience revient à calculer le noyau de viabilité du système (5.3) dans l'espace des contraintes $\mathcal{H} = X \times \mathbb{R}^+$. Le calcul du noyau nécessite donc l'addition d'une dimension supplémentaire, représentant le coût de restauration. Cependant, les algorithmes de viabilité souffrent de la « malédiction de la dimensionnalité » : les coûts de calcul augmentent exponentiellement avec la dimension de l'espace d'état. Dans cette partie, nous proposons de modifier la dynamique et d'intégrer une fonction de coût respectant certaines conditions afin de pouvoir adapter l'algorithme d'approximation de bassins de capture dans l'espace d'état initial, et ainsi gagner une dimension pour les calculs.

On définit une nouvelle fonction de coût $\ell(x, u)$ qui pénalise le temps passé par la trajectoire issue de x en dehors du noyau de viabilité du système :

$$\ell(x, u) = \begin{cases} \lambda(x, u) + g(x) & \text{si } x \notin Viab(K) \\ 0 & \text{si } x \in Viab(K), \end{cases} \tag{5.5}$$

avec $g(x) > 0$ le coût lorsque $x \notin Viab(K)$ et $\lambda(x, u) = 0$ si $x \in K$. On pose $(x(t), c(t))' = 0$ si $x \in Viab(K)$ car lorsqu'un point $x \in Viab(K)$, il existe au moins une trajectoire qui va lui permettre de rester dans l'ensemble des contraintes : on pourra donc dans ce cas définir une politique d'action qui respecte la propriété d'intérêt, et le coût de restauration associé sera donc nul. Par rapport à la

définition de [Martin, 2004], on ne pénalise ainsi pas uniquement le temps passé en dehors de l'espace des contraintes K (avec la fonction de coût $\lambda(x,u)$), mais également le temps passé en dehors du noyau de viabilité du système (avec la fonction de coût $g(x)$). Avec $\ell(x,u) > 0$ si $x \notin Viab(K)$, on peut donc réécrire le système (5.3) :

$$(x(t),c(t))' \in F_E(x(t),c(t)) = \begin{cases} F(x(t)) \times \{-\ell(x(t),u(t))\} & \text{si } x \notin Viab(K) \\ 0 & \text{si } x \in Viab(K), \end{cases} \tag{5.6}$$

ce qui permet de se ramener au problème de minimisation d'une fonction de coût défini dans la section 3.4 avec un système dynamique (3.48), $Viab(K)$ étant la cible C à atteindre. L'introduction de la fonction $g(x)$ est indispensable pour pouvoir appliquer l'algorithme d'approximation d'un bassin de capture du système (3.48) défini dans la section 3.4. En effet, avec $\ell(x,u) > 0$ si $x \notin Viab(K)$, la dynamique sur la dimension coût est strictement décroissante en dehors du noyau de viabilité (considéré ici comme la cible à atteindre) et on peut utiliser l'algorithme d'approximation dans l'espace d'état initial.

Cette procédure permet, lorsque la fonction de coût pénalise le temps passé en dehors du noyau de viabilité de K (avec la fonction $g(x)$), d'approcher les valeurs de résilience sans ajouter une dimension auxiliaire au système. Si on souhaite pénaliser uniquement le temps passé en dehors de K, on peut toujours utiliser l'algorithme d'approximation de noyau de viabilité défini dans la section 2, en considérant la dynamique étendue (5.3).

5.1.3 Extension aux jeux dynamiques

Dans cette sous-section, nous introduisons un algorithme simple pour calculer des noyaux discriminants. Cet algorithme est seulement un premier pas dans cette direction et reste insuffisant, mais il nous permettra dans la section suivante d'introduire des incertitudes dans le modèle.

Lorsque l'évolution du système est influencée par des perturbations liées à notre ignorance de certains comportements, on définit un jeu dynamique :

$$\begin{cases} x(t)' = \varphi(x(t),u(t),v(t)) \\ u(t) \in U(x(t)) \\ v(t) \in V(x(t)), \end{cases} \tag{5.7}$$

où les différentes perturbations v qui peuvent intervenir appartiennent à un certain ensemble $V(x(t))$. Dans ce cas, on recherche l'ensemble des états viables du système, quelle que soit la valeur de la perturbation. Un état x est viable si, pour chaque valeur $v \in V(x(t))$ et pour tout $x \in K$, il existe au moins une évolution $x(.)$ viable dans K. L'ensemble de tous les états viables est alors appelé noyau discriminant.

Dans ce chapitre, nous proposons un premier pas pour approcher un noyau discriminant, en modifiant légèrement l'algorithme d'approximation de noyau de viabilité. La procédure est simple : on discrétise l'ensemble des incertitudes et on teste toutes les valeurs discrètes pour chaque état. Si le point est non-viable pour au moins une valeur de l'intervalle, le point est non-viable. De la même façon, pour l'algorithme d'approximation de bassins de capture, si un point ne capture pas la cible pour au moins une valeur de l'intervalle, le point ne capture pas la cible.

5.2 Calcul des valeurs de résilience dans un modèle d'eutrophisation des lacs

Le problème d'eutrophisation des lacs est un exemple connu de modification soudaine du système, d'un état oligotrophique (eau claire) à un état eutrophique (eau trouble) [Scheffer, 1998 ; Carpenter *et al.*, 1999b]. Ce phénomène est très répandu et se développe dans les eaux de surface [Smith, 1998] ;

il est dû à la décharge excessive de nutriments dans l'eau, décharge liée aux activités agricoles. L'eutrophisation des lacs a des conséquences néfastes sur l'environnement, telles que la diminution de la biodiversité aquatique [Jeppesen *et al.*, 1998], turbidité et faible valeur économique. Le problème est donc de garder le lac dans un état oligotrophique (faibles apports de nutriments, eau claire et forte valeur économique), tout en permettant aux agriculteurs d'utiliser des nutriments pour assurer la rentabilité de leur activité.

Le phosphate est le nutriment le plus critique pour l'eutrophisation des lacs [Carpenter *et al.*, 1999b ; Carpenter *et al.*, 1999a]. L'excédent de phosphate est importé des fermes sous la forme d'engrais et de complément alimentaire pour les animaux. La plupart du phosphate s'accumule dans le sol [Bennett *et al.*, 2001], et peut être transporté dans les eaux par les pluies. La propriété d'intérêt d'un lac est donc l'oligotrophie selon le point de vue de la population qui profite de ses services. Cependant, les agriculteurs doivent être inclus dans le système, selon le point de vue qu'il doivent conserver des activités rentables, ce qui est lié à l'apport de phosphate. L'objectif est donc double : le lac doit rester dans un état oligotrophique, et la rentabilité des activités agricoles assurée.

Les questions que se pose le gestionnaire du lac sont : est-il possible de garder le lac dans un état oligotrophique tout en assurant la rentabilité des activités agricoles ? Si oui, quelles sont les politiques d'actions à mener ? Si non, existe t-il une politique qui permettra de retrouver la propriété d'intérêt ? A quel prix ?

5.2.1 Modèle simple d'eutrophisation des lacs

Le modèle comprend un lac et les activités agricoles qui apportent du phosphate. La propriété d'intérêt évaluée par la résilience est l'état oligotrophique et la pérennité des activités agricoles.

5.2.1.1 La dynamique et les contrôles

Le modèle combine un modèle d'écosystème pour la dynamique des phosphates et un modèle contrôlé pour la dynamique des apports de nutriments. La dynamique des apports de phosphates dans l'eau et les sédiments, proposée par [Dent *et al.*, 2002], est la suivante :

$$\begin{cases} \frac{dM}{dt} = -kM(t) + sP(t) - rM(t)f(P(t)) \\ \frac{dP}{dt} = -(s+h)P(t) + L(t) + rM(t)f(P(t)), \end{cases} \tag{5.8}$$

avec $f(P(t)) = \dfrac{P(t)^q}{P(t)^q + m^q}$. Le système est défini par trois variables d'état : la variable M représente la masse de phosphate dans les sédiments du lac, P la masse de phosphate dans l'eau du lac (toutes deux décrites en unité de temps et en $g.m^{-2}$) et L le flux de phosphate ($g.m^{-2}.y^{-1}$). Les coefficients s et h sont des taux constants respectivement pour la sédimentation et l'écoulement des eaux. k représente la proportion de vase qui se transforme à chaque pas de temps en dépôt souterrain permanent. r correspond au taux maximal de recyclage des phosphates. La fonction $f(P)$ représente le passage d'un recyclage faible à un fort recyclage. Le paramètre q contrôle la pente de la courbe de recyclage et le niveau de phosphate P pour lequel le taux de recyclage est à la moitié de son taux maximal est noté m. La figure 5.1 (tirée de [Ludwig *et al.*, 2003]) résume le modèle.

Un modèle plus simple, où l'influence de la masse de phosphate dans les sédiments M sur la dynamique de la masse de phosphate dans l'eau du lac P n'est pas prise en compte, est souvent utilisé dans la littérature [Ludwig *et al.*, 2003]. [Martin, 2004] a utilisé ce modèle pour définir les valeurs de résilience. En comparant les résultats obtenus dans cette étude avec ceux de [Martin, 2004], nous montrerons que la part de phosphates dans les sédiments a un impact significatif sur les valeurs de la résilience.

Afin de montrer l'influence de la variable M, on conserve la dynamique sur les flux de phosphates proposée par [Martin, 2004]. Le gestionnaire du lac peut agir directement sur la variation des flux de

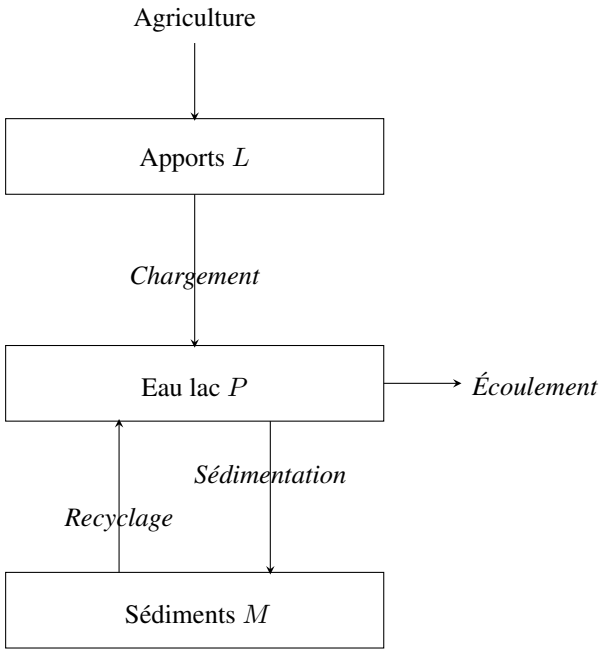


FIG. 5.1 – Modèle simplifié du lac en trois dimensions (L, P, M).

phosphates, à travers l'utilisation d'une variable de contrôle u bornée, car les modifications prennent du temps :

$$\frac{dL}{dt} = u \, , \, u \in [-VL_{max}; VL_{max}] \, . \tag{5.9}$$

5.2.1.2 La propriété d'intérêt comme espace des contraintes

On suppose qu'un lac oligotrophique passe à un état eutrophique lorsque la quantité de phosphates dans l'eau dépasse un certain seuil P_{max}. En conséquence, l'objectif est atteint lorsque la variable $P \geq 0$ satisfait la condition suivante :

$$P \in [0; P_{max}] \, . \tag{5.10}$$

On suppose également que les profits des agriculteurs dépendent linéairement des apports en phosphates. La rentabilité de leur activité est donc assurée lorsque les apports en phosphates sont supérieurs à une certaine valeur L_{min} et inférieure à la valeur maximale légale L_{max} :

$$L \in [L_{min}; L_{max}] \, . \tag{5.11}$$

Les équations (5.8), (5.9), (5.10), (5.11) peuvent être réécrites synthétiquement en utilisant le formalisme de la viabilité, avec $x(t) = (L(t), P(t), M(t))$:

$$x'(t) = \begin{pmatrix} L'(t) \\ P'(t) \\ M'(t) \end{pmatrix} = \begin{pmatrix} u(t) \\ -(s+h)P(t) + L(t) + rM(t)f(P(t)) \\ -kM(t) + sP(t) - rM(t)f(P(t)) \end{pmatrix} , \tag{5.12}$$

avec $K = [L_{min}; L_{max}] \times [0; P_{max}] \times [0; +\infty]$.

5.2.1.3 Les fonctions de coût

Les fonctions de coût sont utilisées pour évaluer la résilience de la propriété d'intérêt, définie par l'ensemble des contraintes K, qui assure la rentabilité de l'activité des agriculteurs, tout en gardant le lac oligotrophique. Ces fonctions doivent donc satisfaire deux conditions : *(i)* le coût d'une trajectoire telle que la propriété d'intérêt est maintenue est nul *(ii)* le coût d'une trajectoire telle que la propriété d'intérêt ne peut pas être restaurée est infini. De plus, la trajectoire, partant d'un état initial x_0, de coût minimal définit la meilleure politique d'action à mener pour conserver la propriété d'intérêt ou pour la restaurer avant le temps T. En pratique, les façons d'évaluer le coût d'une évolution $x(.) = (L(.), P(.), M(.))$, satisfaisant l'équation (5.12), sont nombreuses et dépendent de la situation. On considère dans cet exemple une fonction de coût $\lambda(x, u)$, pénalisant le temps passé hors de K, qui peut être décomposée en deux :

- le coût écologique : pénalise le temps passé hors de l'état oligotrophique,
- le coût économique : mesure la période de non-rentabilité des activités agricoles, pondérée par la norme des profits négatifs.

L'équation (5.13) définit la fonction $\lambda(x, u)$:

$$\lambda(x,u) = \min_{x(.)} \left(c_1 \int (L_{min} - L)\chi_{L \leq L_{min}}(x(\tau)d\tau) + c_2 \int \chi_{P \geq P_{max}}(x(\tau)d\tau) \right), \tag{5.13}$$

avec $\chi_{L \leq L_{min}} = 1$ si $L \leq L_{min}$, $\chi_{P \geq P_{max}} = 1$ si $P \geq P_{max}$, et 0 sinon.
On introduit également une deuxième fonction de coût $g(x)$, correspondant à un coût de management, qui pénalise le temps passé en dehors du noyau de viabilité du système : les politiques d'action doivent être scrupuleusement suivies afin de restaurer, si possible, la propriété d'intérêt avec un coût minimal :

$$g(x) = \min_{x(.)} \left(c_3 \int \chi_{x \notin Viab(K)}(x(\tau)d\tau) \right), \tag{5.14}$$

avec $\chi_{x \notin Viab(K)} = 1$ si $x \notin Viab(K)$ et 0 sinon. L'introduction de cette fonction est indispensable pour pouvoir utiliser l'algorithme décrit dans la section 4 : la fonction de coût doit être strictement décroissante en dehors de la cible à atteindre (le noyau de viabilité ici).
La fonction $\ell(x, u)$, qui associe à un état x le coût minimal, est définie par l'équation (5.15) :

$$\ell(x,u) = \lambda(x,u) + g(x). \tag{5.15}$$

5.2.1.4 Les perturbations

Dans cet exemple, on considère des perturbations D_α, qui correspondent à une augmentation soudaine de la concentration de phosphates dans le lac, avec α l'intensité maximale de la perturbation. Si une perturbation se produit lorsque le système est à un état x, il passera à un état y appartenant à l'ensemble $D_\alpha(x)$:

$$D_\alpha(x) = [x; x + (0; \alpha, 0)]. \tag{5.16}$$

5.2.1.5 Les valeurs de résilience

La valeur de la résilience d'un système à un état donné x pour sa propriété d'intérêt, en prenant en compte les perturbations attendues, est l'inverse du coût de restauration pour tous les sauts décrits par D. Dans cet exemple simple de perturbations, l'augmentation soudaine de P avec une intensité maximale α produit le coût maximal de restauration. On a donc :

$$R^T_{K,D}(x) = \frac{1}{\ell(x + (0; \alpha, 0), u)}. \tag{5.17}$$

5.2.2 Résultats

Le calcul des valeurs de résilience nécessite deux étapes préliminaires : approcher le noyau de viabilité puis les coûts de restauration de la propriété d'intérêt.

5.2.2.1 Le noyau de viabilité

Le but est de déterminer quels sont les niveaux de phosphates dans l'eau (P), dans les sédiments (M) et des apports (L) qui sont compatibles avec l'objectif de maintenir la propriété d'intérêt. Pour cela, on calcule le noyau de viabilité du système (5.12), en utilisant l'algorithme décrit dans le chapitre 1. Les paramètres du modèle choisis pour les simulations sont décrits dans le tableau 5.1 et sont tirés de [Janssen et Carpenter, 1999] et [Martin, 2004].

Paramètres	**Valeurs**
L_{min}	$0,1$
L_{max}	1
P_{max}	$0,5$ - $1,2$
VL_{max}	$0,1$
s	$0,5$
h	$0,3$
k	$0,01$
q	8
r	1
m	$[0,8;1,2]$

TAB. 5.1 – Paramètres du modèle d'eutrophisation des lacs.

Le paramètre m est estimé à partir des données issues de chaque lac, sa valeur ne peut être connue avec certitude. C'est pour cette raison que l'on inclut les incertitudes sur ce paramètre $m \in [0,8;1,2]$. Pour calculer le noyau de viabilité du système (ou noyau discriminant), on utilise alors l'algorithme défini dans la sous-section 5.1.3.

La figure 5.2 représente le noyau de viabilité du système pour $P_{max} = 0,5$, pour une valeur de $m = 1$. Le tableau 5.2 présente les paramètres de simulation (colonne noyau 1). On a choisi ici de définir un pas de discrétisation différent pour chaque dimension, afin de mieux rendre compte de la dynamique lente de la variable M.

	noyau 1	**noyau 2**	**coût**
Dimension	3	3	3
Nombre total de points	129381 ($21 \times 61 \times 101$)	129381	129381
dt	$0,05$	$0,05$	-
dc	$-$	$-$	$0,1$
Nombre de pas	6	6	6
C	30000	30000	30000
σ^2	$0,1$	$0,1$	$0,1$
Nombre d'itérations	18	39	52
Nombre final de SV	97	133	188
Temps	20 minutes	1 heure	$\approx$ 2 jours

TAB. 5.2 – Paramètres et résultats des simulations pour le problème d'eutrophisation des lacs.

Pour tous les points $x \in Viab(K)$, il est possible de définir une politique de contrôle qui permette de conserver la propriété d'intérêt : le coût de restauration est nul et la résilience infinie. Pour les points $x \notin Viab(K)$, le système est voué à sortir de K, quelle que soit la politique choisie. La figure montre en trait pointillé un exemple de trajectoire partant d'un point non-viable et satisfaisant $u = VL_{max}$: même en diminuant au maximum l'apport de phosphate, la propriété d'intérêt ne peut être conservée et le lac est condamné à devenir eutrophique.

Les projections sur le plan $(L - P)$ varient très peu avec M. En comparant ce résultat avec ceux de [Martin, 2004], on constate que les approximations sont très proches. Avec de tels paramètres, la concentration de phosphates dans les sédiments n'affecte pas la forme du noyau. De plus, l'approximation du noyau ne change pas lorsqu'on inclut les incertitudes sur le paramètre m.

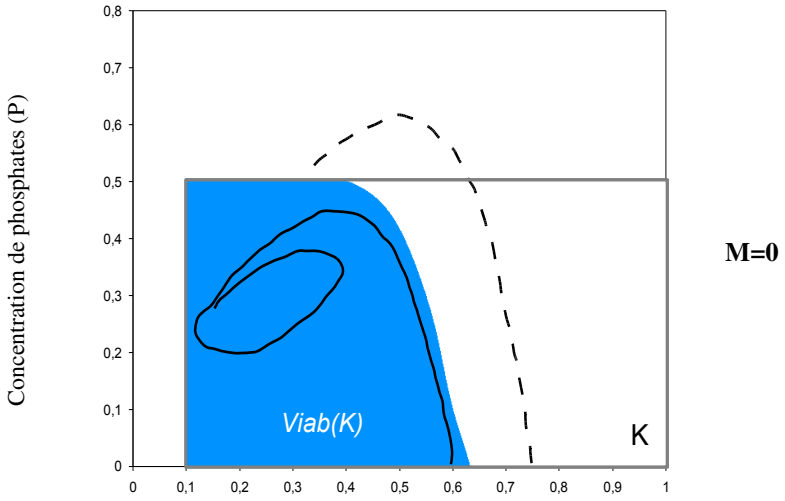


FIG. 5.2 – Projection du noyau de viabilité pour $P_{max} = 0,5$. Le noyau est représenté en bleu et l'espace des contraintes K par le rectangle gris. La trajectoire en pointillé part d'un point non-viable et la trajectoire en trait plein d'un point viable.

On approche maintenant le noyau de viabilité pour $P_{max} = 1,2$ (figure 5.3). Les paramètres sont donnés dans le tableau 5.1 (colonne noyau 2). Les résultats mettent en évidence le rôle crucial de la dynamique lente du modèle : plus la masse de phosphates dans le sol est importante, plus le noyau de viabilité est petit.

Dans toutes les configurations, il y a des situations pour lesquelles la propriété d'intérêt ne peut être maintenue. Si on prend en compte les incertitudes sur le paramètre m, le noyau de viabilité change, tout spécialement pour les plus fortes valeurs de M. Pour $M < 3$, le noyau de viabilité est plus petit que celui approché sans prendre en compte les incertitudes et l'objectif peut être maintenu dans moins de cas. Pour $M \geq 3$, les incertitudes n'ont pas d'influence sur le résultat.

La prochaine étape est de déterminer si le système (en incluant les incertitudes) peut être restauré, et si oui, à quel prix.

5.2.2.2 Les coûts de restauration

Le noyau de viabilité du système correspond au niveau 0 de la fonction de coût. A partir d'un état non-viable, le système est condamné à quitter l'ensemble des contraintes K. Cependant, dans certains

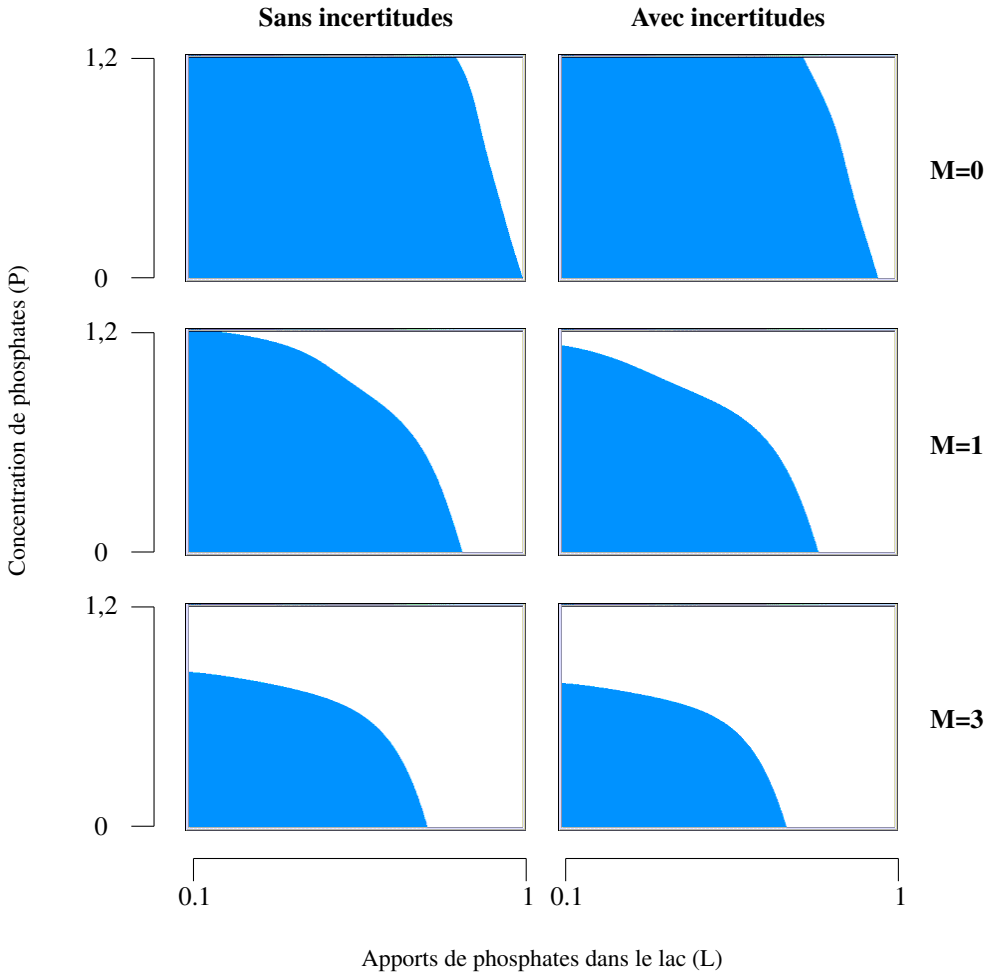


FIG. 5.3 – Projection du noyau de viabilité pour $P_{max} = 1, 2$.

cas, il est possible de restaurer la propriété d'intérêt.

On calcule ici les valeurs non-nulles de la fonction de coût en utilisant l'algorithme décrit dans la section 5.1. Les paramètres de la simulation sont donnés dans le tableau 5.1 (colonne coût). La figure 5.4 représente la projection sur le plan $(L - P)$ des coûts de restauration pour $P_{max} = 1, 2$, en incluant les incertitudes sur le paramètre m. Ici, on choisit de donner un coût écologique plus fort que le coût économique ($c_1 = 3$ et $c_2 = 10$), et on considère un coût de management $c_3 = 1$. On constate que les valeurs de coût prennent des valeurs finies pour tous les états représentés : quel que soit l'état initial du lac, il est possible de restaurer sa propriété d'intérêt.

5.2.2.3 Les valeurs de la résilience

Les valeurs de la résilience sont l'inverse du coût de restauration de la propriété d'intérêt, perdue à cause de perturbations exogènes. On suppose que, quel que soit l'état x du système, la perturbation maximale est causée par une augmentation de la valeur de P, d'une magnitude de $\alpha = 0, 5$. On évalue la résilience pour chacune des fonctions de coût, avec $P_{max} = 1, 2$, et en prenant en compte les incertitudes sur le paramètre m (figure 5.5).

Le graphique discrimine deux zones :

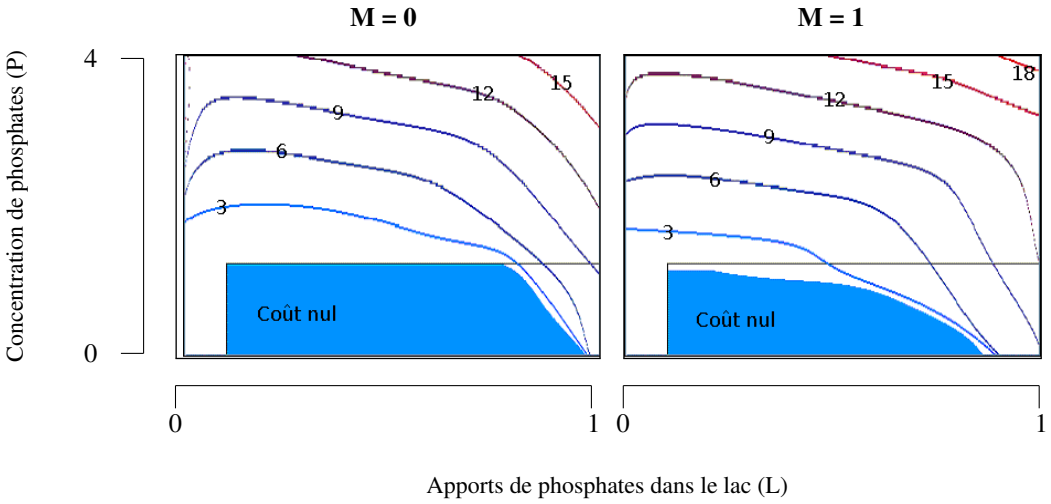


FIG. 5.4 – Projection de l'approximation des valeurs de coût pour $P_{max} = 1,2$.

- en bleu, les états pour lesquels la résilience est infinie : même avec une perturbation de magnitude $\alpha = 0,5$ anticipée, le lac reste dans un état oligotrophique et la rentabilité économique des activités agricoles est assurée ;
- entre les courbes de niveaux, le lac va devenir eutrophique dans le cas de perturbations, mais la propriété d'intérêt va pouvoir être restaurée, avec un coût donné par les lignes de niveaux.

Dans cet exemple, on n'observe aucune zone de résilience nulle. Avec les paramètres $(s + h)$ choisis, le lac est hystérésique : l'eutrophisation peut être renversée en diminuant sérieusement les apports en phosphates [Carpenter *et al.*, 1999b].

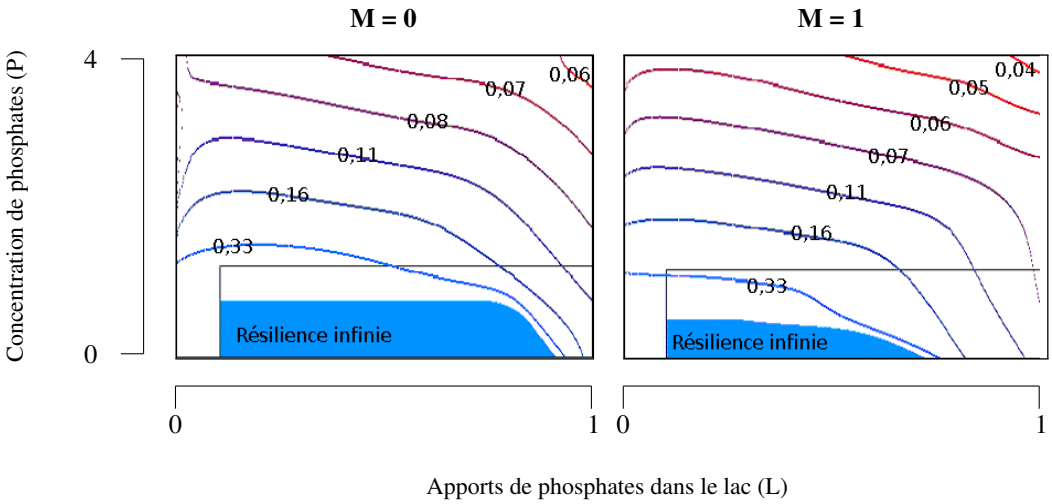


FIG. 5.5 – Projection de l'approximation des valeurs de résilience pour $P_{max} = 1,2$.

5.2.2.4 Déterminer des politiques d'action

Dans cet exemple, les actions à mener pour restaurer la propriété d'intérêt avec un coût minimal sont simples : il suffit de diminuer les apports en phosphates L jusqu'à retrouver un lac oligotrophique, puis

d'augmenter L jusqu'à obtenir des profits acceptables. Pour déterminer les politiques d'action à mener pour restaurer la propriété d'intérêt avec un coût minimal, il suffit d'utiliser la procédure de contrôle optimal définie dans le chapitre 4.

Troisième partie

Intégrer une procédure d'apprentissage actif

Chapitre 6

Apprentissage actif de noyau de viabilité et de bassins de capture

Les algorithmes décrits dans ce manuscrit sont basés sur la discrétisation de l'espace d'état et souffrent donc de la malédiction de la dimensionnalité. Dans ce chapitre, nous proposons d'introduire des méthodes d'apprentissage actif dans les algorithmes d'approximation de noyau de viabilité et de bassin de capture utilisant les SVMs.

Nous définissons tout d'abord la malédiction de la dimensionnalité dans la section 6.1. Dans la section 6.2, nous introduisons l'apprentissage actif en général puis dans le cas des SVMs. Nous proposons ensuite un nouvel algorithme d'apprentissage actif de noyau de viabilité et de bassin de capture dans la section 6.3. Nous terminons avec quelques exemples d'application dans la section 6.4, notamment avec un exemple de contrôle d'un vélo dans un circuit, système défini en 6 dimensions.

Sommaire

6.1 Réduction de la taille de la base d'apprentissage

Les algorithmes proposés dans cette thèse souffrent de la malédiction de la dimensionnalité : la taille de la grille augmente exponentiellement avec la dimension de l'espace d'état. Ces algorithms ne sont applicables que pour des systèmes de faible dimension (5 ou 6), en utilisant une grille avec peu de points par dimension. Un problème crucial est donc d'étendre ces méthodes afin de pouvoir les appliquer à des systèmes en plus grande dimension.

Le problème de la dimensionnalité est général dans les problèmes de contrôle optimal mais est encore plus critique dans notre approche car le temps de calcul des SVMs est quadratique avec la taille de la base d'apprentissage. Cependant, les SVMs ne souffrent pas de la malédiction de la dimensionnalité (voir section 2.1.5). De plus, on utilise un algorithme de type SMO pour calculer les fonctions SVMs qui ne nécessitent pas le stockage de la matrice $N \times N$. La malédiction de la dimensionnalité au niveau des SVMs est donc réduite à un problème de temps de calcul et de stockage de la base d'apprentissage.

On rappelle que la fonction SVM est définie à l'aide d'un nombre réduit de points : les vecteurs de support, qui contiennent toute l'information nécessaire. Les autres exemples n'entrent pas dans la définition (ils ont leur coefficient $\alpha_i = 0$). En utilisant uniquement les SVs pour construire la fonction de décision, les résultats sont les mêmes que ceux obtenus en utilisant la base d'apprentissage complète ([Scholkopf et Smola, 2002], section 7.8). Ainsi, même si la base d'apprentissage contient un très grand nombre d'exemples, seulement une partie d'entre eux est pertinente.
Il y a deux types de SVs :

- ceux qui se situent sur la marge ($0 < \alpha_i < C$) ;
- ceux qui sont situés à l'intérieur des marges ou mal-classés par la SVM ($\alpha_i = C$).

Il n'y a pas de « méthode » qui permette de borner supérieurement le nombre de SVs total pour définir un hyperplan séparateur (au minimum, 2 points sont nécessaires). Le nombre de SVs dépend des données et du paramètre de régularisation C, et croît linéairement avec le nombre d'exemples [Scholkopf *et al.*, 2000]. Cependant, dans les exemples d'application décrits dans les chapitres 2 et 4, leur nombre est généralement très petit par rapport à la taille de la grille. De plus, on cherche à ce que la SVM ne fasse pas d'erreur (respect des conditions d'application des algorithmes avec $\lambda = 1$) : la fonction est donc uniquement définie avec les SVs tels que $0 < \alpha_i \leq C$.

Dans ce chapitre, nous proposons de construire progressivement la base d'apprentissage des SVMs, en essayant de la maintenir la plus réduite possible. Cela implique une procédure qui sélectionne les points à ajouter à la base d'apprentissage, afin d'en utiliser uniquement un nombre qui soit proche du nombre de vecteurs de support. Le but n'est pas de casser la malédiction de la dimensionnalité mais de pouvoir travailler avec des systèmes en plus grande dimension, en ayant des résultats précis. Afin de réduire très fortement la taille de la base d'apprentissage tout en conservant une approximation précise du noyau de viabilité ou du bassin de capture, nous combinons une méthode d'apprentissage actif et une grille virtuelle multi-résolution.

6.2 Apprentissage actif

6.2.1 Présentation générale

L'apprentissage actif [Cohn *et al.*, 1995 ; Tong, 2001] regroupe l'ensemble des méthodes de sélection de points utilisées pour construire l'ensemble d'apprentissage de manière itérative. Les points sélectionnés ne sont pas connus en avance mais sont basés sur le résultat des sélections précédentes. Le processus de construction de la base d'apprentissage en utilisant les résultats obtenus par les requêtes précédentes est appelé apprentissage actif. Toutes les stratégies de sélection ont en commun de chercher à utiliser le moins de points possibles, en sélectionnant les instances les plus « informatives ». L'apprentissage actif s'oppose à la notion d'apprentissage passif, où la base d'apprentissage est composée d'instances choisies au hasard, sans tenir compte des résultats des itérations précédentes. Le but général

est d'économiser de l'espace mémoire, des ressources de calcul et le coût d'étiquetage des instances non indispensables. En effet, dans certaines applications comme la classification de documents, le coût d'étiquetage des données est très important [Tong et Koller, 2000].

Dans le processus d'apprentissage actif, on acquiert des informations sur le « monde » en formulant des requêtes et en recevant des réponses. On construit ensuite un classifieur (figure 6.1). A partir d'un ensemble de points $\mathcal{U}$ non étiquetés, le processus d'apprentissage actif ℓ a trois composants (f, q, S) :

– $\mathcal{S}$: base d'apprentissage, composée d'un ensemble de points et de leur étiquette ;
– q : à partir de la base d'apprentissage $\mathcal{S}$, détermine quelle instance $x \in \mathcal{U}$ à étiqueter ;
– $f : \mathcal{X} \to [-1; +1]$, calculé sur la base d'apprentissage $\mathcal{S}$ (dans le cas de la discrimination binaire).

La procédure d'apprentissage actif renvoie un classifieur f après un nombre fixé de requêtes [Tong, 2001].

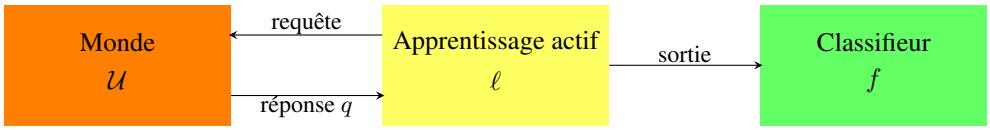


FIG. 6.1 – Schéma général de l'apprentissage actif [Tong, 2001].

Le problème principal est de choisir les instances les plus informatives à étiqueter.

Les SVMs ont reçu beaucoup d'attention dans le contexte de l'apprentissage actif [Tong et Chang, 2001 ; Schohn et Cohn, 2000 ; Campbell *et al.*, 2000] car ils discriminent les données en utilisant les instances les plus informatives (les vecteurs de support). Ce sont donc des candidats naturels dans le cadre de l'apprentissage actif. Dans le cadre des SVMs, le processus d'apprentissage actif essaie de minimiser le nombre de points non-SVs étiquetés. Par exemple, [Tong et Chang, 2001] ont proposé une heuristique pour sélectionner les instances, basée sur leur proximité à la frontière de décision, ce qui permet de réduire l'espace des versions (« version space » en anglais). La même procédure a été développée par [Schohn et Cohn, 2000], avec une justification plus géométrique, qui montrent que le processus diminue le temps de calcul.

6.2.2 Apprentissage actif et contrôleur de viabilité

Pour notre problème d'apprentissage d'un contrôleur de viabilité, l'étiquetage des données est coûteux car il requiert l'optimisation d'une fonction de contrôle. Mais le principal problème est de limiter le nombre de points utilisés pour calculer la fonction SVM, nombre qui augmente avec la dimension et la qualité de l'approximation. On choisit de combiner une procédure d'apprentissage actif avec une grille multi-résolution, afin de diminuer de façon drastique la taille de la base d'apprentissage, tout en gardant une approximation fine du noyau. On ajoute donc une boucle supplémentaire dans l'algorithme initial : on ajoute peu à peu les points sélectionnés dans la base d'apprentissage, jusqu'à obtenir une fonction SVM qui ne fait aucune erreur sur la grille complète. Un compromis entre le nombre d'apprentissages nécessaires et la taille de la base d'apprentissage doit être trouvé : dans notre problème, on recherche des couples de points d'étiquettes opposées les plus proches, car de tels points permettent de contraindre la fonction SVM. Comparé à l'ajout de points uniques, on obtient des bases d'apprentissage de plus grande taille, mais l'approximation d'une fonction SVM ne faisant aucune erreur nécessite moins d'itérations.

La figure 6.2 décrit l'algorithme d'approximation du noyau en utilisant une procédure d'apprentissage actif.

La principale contribution de ce chapitre est la procédure d'apprentissage actif pour mettre à jour la base d'apprentissage. L'idée principale est d'ajouter un nombre limité de couples de points qui contraignent f_n^k à chaque mise à jour de la base d'apprentissage.

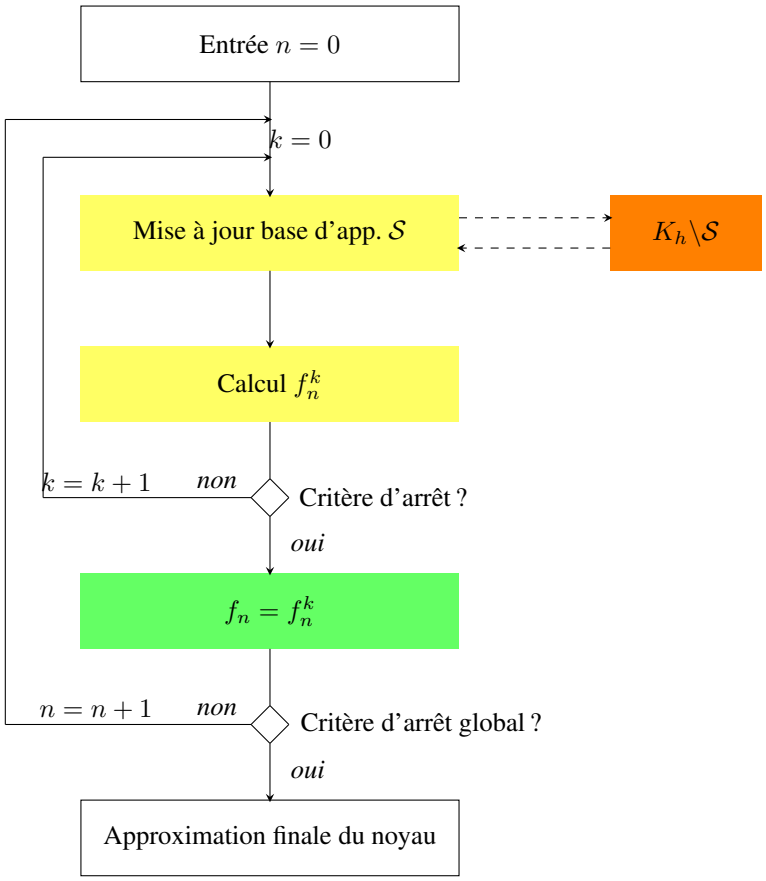


FIG. 6.2 – Schéma général de l'algorithme d'approximation de noyau de viabilité en utilisant une procédure d'apprentissage actif.

6.3 Apprentissage actif de noyau de viabilité et de bassins de capture

6.3.1 Principe général

Dans le problème d'approximation de noyaux de viabilité ou de bassins de capture, on choisit de construire des fonctions SVMs qui ne font aucune erreur pour respecter les conditions d'application du théorème (on considère ici $\lambda = 1$). Il est donc indispensable de définir une procédure d'apprentissage actif qui fournit une fonction SVM qui classifie correctement tous les points de la grille. De plus, les seuls points vecteurs de support seront donc tels que $0 < \alpha_i < C$, i.e. les points situés sur la marge : il suffit donc de mettre à jour la base d'apprentissage en choisissant parmi les points proches de la fonction SVM recherchée.

On définit une procédure qui se concentre uniquement sur la frontière : si tous les points autour de la frontière de décision sont bien classés, alors tous les autres points de la grille le seront également. Nous avons développé un premier algorithme, qui sélectionne tous les exemples avec une étiquette -1 de la grille et qui sont à une distance inférieure à $2 \times \beta(h)$ de la frontière de l'approximation suivante du noyau, et leur associe un point d'étiquette $+1$, situé à une distance inférieure à $2 \times \beta(h)$. Ainsi, la base d'apprentissage comprend tous les points autour de la frontière de L_h^{n+1}, à une résolution $2 \times \beta(h)$.

Cette méthode permet d'obtenir une base d'apprentissage dont la taille correspond à un problème en dimension $d-1$. Nous ne détaillerons pas cette procédure dans ce chapitre ; pour plus d'informations, le lecteur pourra se référer à [Chapel et Deffuant, 2006].

Dans cette section, nous détaillons une procédure d'apprentissage actif pour mettre à jour la base d'apprentissage. Le principe général est d'ajouter un nombre minimal de couples de points d'étiquettes opposées, qui contraignent au mieux la fonction SVM à chaque mise à jour de la base d'apprentissage. De plus, nous définissons également une grille adaptative, qui permet d'obtenir une approximation plus fine du noyau de viabilité. Afin de décrire la procédure plus en détail, nous introduisons quelques notations.

6.3.2 Discrétisation adaptative

6.3.2.1 Construction de la grille

On utilise une grille virtuelle, sur laquelle nous pouvons définir plusieurs résolutions. [Grüne, 1997 ; Munos et Moore, 2002] ont utilisé de la même façon une grille à résolution variable dans le cadre de l'apprentissage par renforcement et de la programmation dynamique. Cette discrétisation adaptative leur permet d'obtenir des résultats lorsque le pas de la discrétisation augmente, spécialement lorsque les systèmes étudiés sont en grande dimension. Ils ont défini les régions critiques sur lesquelles la grille doit être développée.

La différence majeure avec la méthode présentée ici est que notre grille multi-résolution est « virtuelle ». Elle n'est jamais instanciée explicitement et est utilisée pour sélectionner les points les plus pertinents à ajouter à la base d'apprentissage.

Supposons que l'espace d'état est l'ensemble $[0,1]^d$. On pose h la résolution de la grille telle que $h = \frac{1}{m-1}$, où $m > 1$ est le nombre de points par dimension à la résolution h. Les coordonnées x_i d'un point x de la grille à la résolution $\frac{h}{2^j}$, où j est un entier, sont données par les entiers q_i, avec $0 \leq q_i \leq m \times 2^j$ tels que :

$$x_i = \frac{q_i h}{2^j} \text{ pour } i = 1, ..., d. \tag{6.1}$$

La grille de résolution $\frac{h}{2^j}$ contient donc $((m \times 2^j)+1)$ points. Pour simplifier dans la suite, on appellera la résolution $\frac{h}{2^j}$ la résolution j, et on note $\mathcal{G}_j$ la grille correspondante. Pour chaque résolution j, on a :

$$\forall x \in K, \exists y \in \mathcal{G}_j \text{ tel que } d(x,y) \leq \frac{\beta(h)}{2^j}. \tag{6.2}$$

Considérant $x \in K$, on note $\mathcal{G}_j(x)$ le point $y \in \mathcal{G}_j$ qui est le plus proche de x. On pose j_M la résolution la plus fine de la grille.

On organise la grille multi-résolution comme un arbre. Les points de la grille grossière (résolution 0) constituent la racine de l'arbre. Pour $j < j_M$, on définit les feuilles d'un point $x \in \mathcal{G}_j$ comme l'ensemble $C(x)$ de points appartenant à $\mathcal{G}_{j+1}$:

$$C(x) = \left\{ y = x + \epsilon_i \frac{h}{2^{j+1}}, \text{ tel que } y \in K, \text{ avec } \epsilon_i \in \{0,1\} \text{ et } i = 1, ..., d \right\}. \tag{6.3}$$

Chaque point a donc 2^d feuilles, sauf pour certains points situés sur le bord de K (pour lesquels le nombre de feuilles est plus petit car certaines n'appartiennent pas à K). On peut noter que $x \in C(x)$. L'ensemble des feuilles nous permet de définir récursivement l'arbre de niveau j pour un point x :

$$T_0(x) = \{x\} \; ; T_{j+1}(x) = \bigcup_{y \in T_j(x)} C(y). \tag{6.4}$$

A partir des points $x \in \mathcal{G}_0$, la question est : où augmenter la résolution de la discrétisation ? On définit deux critères de développement d'une branche, en partant de la racine pour aller vers les feuilles. Pour chaque niveau j, on développe uniquement les branches qui respectent ces critères. La figure 6.3 présente un exemple de développement de l'arbre à partir d'un point $x \in \mathcal{G}_0$.

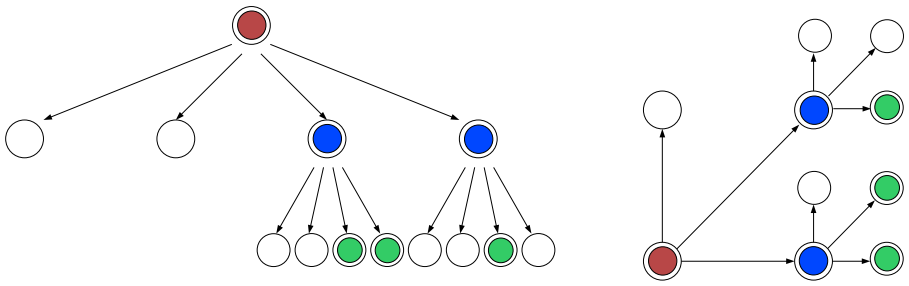


FIG. 6.3 – Exemple d'arbre de profondeur 2 en dimension 2 et discrétisation correspondante. A partir de la racine (en rouge), on obtient quatre feuilles, dont deux seulement (en bleu) respectent un certain critère. On développe uniquement ces branches, pour obtenir 3 points au niveau 2 qui respectent un certain critère (en vert).

6.3.2.2 Critères de développement de l'arbre

Il est suffisant de considérer uniquement les points les plus proches de la frontière de la fonction SVM recherchée, seuls points susceptibles d'être vecteurs de support, et qui suffisent complètement pour définir la fonction SVM sur tout l'espace : si tous les points les plus proches de la frontière de la SVM à la résolution j_M sont bien classés, tous les autres points de la grille le seront aussi.

Un premier critère de développement de l'arbre est donc la proximité avec la fonction SVM. Cependant, la distance entre un point $x \in \mathcal{G}_j$ et la frontière d'une fonction SVM f n'est pas directe à calculer. Pour obtenir une approximation de la distance, on utilise le gradient de la fonction f. Si le point x est proche de la surface de décision f, il n'existe aucun point $y \in \mathcal{G}_j$, choisi tel que $d(x,y) \leq \frac{2\beta(h)}{2^j}$, tel que la frontière passe entre ces deux points. Définissons le point p :

$$p = x + l(x,f)\frac{\beta(h)}{2^j}\frac{\nabla f(x)}{\|\nabla f(x)\|}, \tag{6.5}$$

et la fonction $l(x,f)$:

$$\begin{array}{l}\text{(approximation de noyau de viabilité ou de bassin de capture par l'extérieur)}\\ \quad l(x,f) = +1 \text{ si } f(x) \geq -1, -1 \text{ sinon,}\\ \text{(approximation de noyau de viabilité ou de bassin de capture par l'extérieur)}\\ \quad l(x,f) = +1 \text{ si } f(x) > 1, -1 \text{ sinon.}\end{array} \tag{6.6}$$

Le point x est donc proche de la frontière à la résolution j si les points x et p sont classés différemment par la fonction f :

$$l(x,f) \neq l(p,f). \tag{6.7}$$

En effet, si les deux points ont des étiquettes différentes, cela signifie qu'il existe au moins un point y tel que la frontière passe entre les points x et y.

Pour définir le second critère, on introduit le point x^* comme le point obtenu en partant de x et en appliquant la meilleure fonction de contrôle $(u_i^*)_i$ (avec i le nombre de pas de temps), en fonction d'une SVM donnée f. Un point est dit bien classé par la fonction f_n^k lorsque :

$$l(x, f_n^k) = l(x^*, f_{n-1}). \tag{6.8}$$

On choisit de développer l'arbre jusqu'a ce que l'on trouve un point mal-classé par f_n^k.

On considère tout d'abord l'ensemble des points $x \in \mathcal{G}_0$, et on teste s'ils sont proches à la résolution 0 de la frontière de la fonction SVM courante f_n^k. Si oui, et si le point est bien classé par f_n^k, on développe l'arbre sur chacun des niveaux $T_0(x), T_1(x), ..., T_{j_M}(x)$, en considérant uniquement les points proches de la frontière à la résolution j, jusqu'à ce qu'on trouve un point mal classé par f_n^k ou que l'arbre soit complètement développé.
L'algorithme 6.1 résume la procédure de développement de l'arbre, en partant d'un point $x \in \mathcal{G}_0$.

Algorithme 6.1 developperArbre(Point x, Fonction f)

Pour j = 0 à j_M **Faire**
  Pour $y \in T_j(x)$ **Faire**
    $p = x + l(x, f)\frac{\beta(h)}{2^j}\frac{\nabla f(x)}{\|\nabla f(x)\|}$
    Si $l(y, f) \neq l(p, f)$ **Alors**
      Si $l(y, f_n^k) \neq l(y^*, f_{n-1})$ **Alors**
        Renvoyer y
      Fin si
    Fin si
  Fin pour
Fin pour
Renvoyer null

6.3.3 Mise à jour de la base d'apprentissage

La définition de la grille adaptative nous permet de mettre en place une procédure de mise à jour de la base d'apprentissage, en limitant le nombre de points ajoutés à chaque itération du processus d'apprentissage actif.
Considérons que l'on connaît la fonction f_n^0 (nous verrons plus loin comment la construire). A chaque itération $k > 0$ de l'apprentissage actif, on teste si tous les points $\in \mathcal{G}_{j_M}$ les plus proches (à la résolution j_M) sont bien classés par la fonction f_n^k :

- si oui, la fonction ne fait aucune erreur sur $\mathcal{G}_{j_M}$ et on a $f_n = f_n^k$;
- si non, on met à jour la base d'apprentissage et on calcule une nouvelle fonction f_n^{k+1}.

Le but de l'apprentissage actif est d'ajouter uniquement à la base d'apprentissage un minimum de points les plus pertinents. Nous avons vu précédemment qu'il s'agit des points les plus proches de la fonction f_n recherchée. Mais le nombre de points satisfaisant ce critère explose aussi avec la dimension du problème. On choisit de sélectionner un sous-ensemble de ces points, de sorte qu'ils soient bien répartis sur K. Pour cela, nous allons nous appuyer sur la définition de la grille adaptative : on ajoute un couple de points d'étiquettes opposées par point de la grille $\mathcal{G}_0$ dès que l'on trouve un point $\in T_j(x)$ qui est mal-classé par f_n^k. A chaque itération de l'algorithme d'apprentissage actif, on ajoute ainsi au maximum $(2 \times m)$ points à la base d'apprentissage.
L'algorithme 6.2 résume la procédure de mise à jour de la base d'apprentissage.
La procédure $coupleLabelOppose(x)$ est présentée dans le paragraphe suivant.

Algorithme 6.2 majVectApp()

Pour $x \in \mathcal{G}_0$ **Faire**
  $y \leftarrow$ developperArbre(x, f_n^k)
  Si $y \neq$ null **Alors**
    $\mathcal{S} \leftarrow \mathcal{S} \cup$ coupleLabelOppose(y)
  Fin si
Fin pour

6.3.4 Couple de points d'étiquettes opposées

On considère un point mal-classé $x \in \mathcal{G}_j$ par la fonction f_n^k. On recherche un couple de points x_1 et x_2, voisins sur la grille de résolution la plus fine, tels que $l(x_1^*, f_{n-1}) \neq l(x_2^*, f_{n-1})$, et qui sont aussi proches que possible de x. On navigue sur la grille $\mathcal{G}_{j_M}$ en suivant la direction du gradient $\nabla f_n^k(x)$. La procédure comprend deux étapes :

- on recherche le point x_2 tel que $d(x, x_2) = h$ et $l(x^*, f_{n-1}) \neq l(x_2^*, f_{n-1})$. Cette étape permet de naviguer plus rapidement dans la grille ;
- on divise récursivement le segment en deux parties, en conservant la partie qui comprend la frontière, jusqu'à ce que la taille du segment soit à la plus fine résolution.

L'algorithme 6.3 résume la procédure et la figure 6.4 présente un exemple de points testés et ajoutés à la base d'apprentissage.

L'avantage de cette procédure est qu'elle permet de contraindre au maximum la fonction f_n recherchée, et maximise ainsi l'efficacité de l'apprentissage. En comparaison avec une procédure qui ajouterait uniquement au maximum un point par point mal-classé de la grille grossière, on obtient une base d'apprentissage de plus grande taille mais cela permet d'avoir un compromis entre le nombre d'itérations et la taille de la base d'apprentissage. De plus, en s'appuyant sur la grille $\mathcal{G}_0$, on ajoute un nombre restreint de points, qui sont bien répartis dans l'espace des contraintes.

Algorithme 6.3 coupleLabelOpposé(Point x)

$x_2 \leftarrow x_1$
Faire
  $x_2 = \mathcal{G}_{j_M}\left(x_2 + l(x_1^*, f_{n-1}) \frac{\nabla f_n^k(x_1)}{\|\nabla f_n^k(x_1)\|} h\right)$
Jusqu'à $l(x_1^*, f_{n-1}) \neq l(x_2^*, f_{n-1})$
Tant que $d(x_1, x_2) > \frac{h}{2^{j_M}}$ **Faire**
  $y = \frac{x_1 + x_2}{2}$
  Si $l(x_1^*, f_{n-1}) = l(x_2^*, f_{n-1})$ **Alors**
    $x_2 \leftarrow y$
  Sinon
    $x_1 \leftarrow y$
  Fin si
Fin tant que
Renvoyer $\{(x_1, l(x_1^*, f_{n-1})), (x_2, l(x_2^*, f_{n-1}))\}$

6.3.5 Base d'apprentissage initiale et critère d'arrêt global

A la première itération de la boucle de l'apprentissage actif, on construit une première base d'apprentissage en partant des vecteurs de support de la fonction f_{n-1}. La première base d'apprentissage est alors constituée d'un ensemble de points d'étiquettes opposées sur la grille $\mathcal{G}_{j_M}$, obtenus en appliquant l'algorithme 6.3 sur les SVs de f_{n-1}. La taille de la première base d'apprentissage est donc limitée à ($2\times$ nombre de SVs de f_{n-1}).

A l'initialisation de l'algorithme global, lorsque aucune fonction SVM n'est encore définie, on utilise le bord de K (noté ∂K) pour constituer la base d'apprentissage initial. Définissons la grille constituée des points situés sur ∂K :

$$\mathcal{G}_0' = \{x \in \mathcal{G}_0 \text{ tels que } x \in \partial K\}\,. \tag{6.9}$$

Afin de limiter la taille de la base d'apprentissage initiale, on travaille uniquement avec les points qui sont non-viables à l'itération suivante. Introduisons la fonction $l(x, K)$ qui associe à un point x une étiquette $+1$ si x est viable à l'itération suivante et -1 sinon :

$$l(x, K) = +1 \text{ si } x \in K,\ -1 \text{ sinon.} \tag{6.10}$$

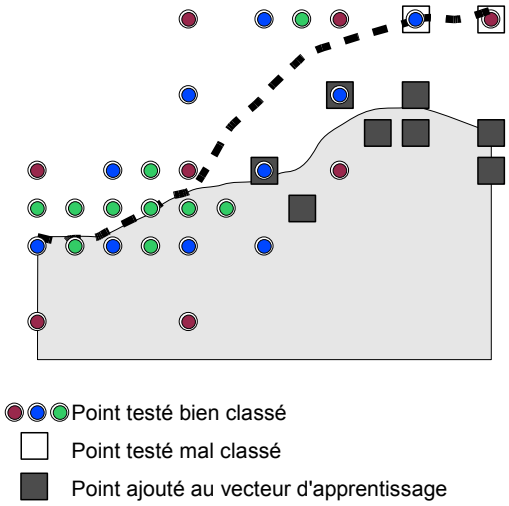


FIG. 6.4 – Mise à jour de la base d'apprentissage. La zone en gris est la zone à approcher. La ligne en pointillé est la surface de séparation de la SVM courante f_n^k. On remarque que pour la partie de la surface pour laquelle il n'y a aucune erreur, la grille est développée jusqu'au niveau le plus fin autour de la frontière. Au contraire, dans la zone où il y a des erreurs, l'arbre n'est pas complètement développé. Les paires de points ajoutées sont représentées par des carrés. Dans cet exemple, 4 paires de points sont ajoutées.

Pour chaque point $x \in \mathcal{G}'_0$ tel que $l(x^*, K) = -1$, on recherche un couple de points d'étiquettes opposées situés sur la grille à la résolution la plus fine. A partir de la méthode *coupleLabelOpposé(Point x)* (algorithme 6.3), on parcourt la grille à diverses résolutions, en choisissant la direction du centre de K au lieu du gradient de f et en remplaçant la fonction $l(x, f)$ par $l(x, K)$. Ainsi, la taille maximale de la base d'apprentissage initiale est de $2 \times Card(\mathcal{G}'_0)$.

Le critère d'arrêt global est rencontré lorsque tous les SVs de f_{n-1} restent à la frontière (chaque SV est le premier point de la paire retournée par l'algorithme 6.3) et qu'on développe complètement la grille $\mathcal{G}_{j_M}$ sans rencontrer de points mal-classés.

6.3.6 Algorithme général d'approximation de noyau de viabilité en utilisant l'apprentissage actif

L'algorithme 6.4 détaille la procédure globale d'approximation d'un noyau de viabilité ou d'un bassin de capture en utilisant la méthode d'apprentissage actif détaillée dans ce chapitre.

6.4 Exemples d'application

6.4.1 Un problème en dimension 2 : population

On considère un système dynamique simple de croissance de population dans un espace limité. La description complète du problème est donnée dans le chapitre 2.

La figure 6.5 présente un exemple d'approximation du noyau de viabilité et d'une trajectoire. Les points situés sur la figure sont ceux qui ont été utilisés pour calculer la dernière fonction SVM. Le

Algorithme 6.4 Algorithme d'approximation d'un noyau de viabilité ou d'un bassin de capture en utilisant des techniques d'apprentissage actif

// Initialisation
$n = 0, k = 0$
$\mathcal{S} \leftarrow \emptyset$

// Boucle initiale
Pour tout $x \in \mathcal{G}'_0$ **Faire**
  Si $l(x^*, K) = -1$ **Alors**
    $\mathcal{S} \leftarrow \mathcal{S} \cup$ *coupleLabelOpposéInit(x)*
  Fin si
Fin pour
Définir f_0^0 à partir de $\mathcal{S}$
Faire
  $k = k + 1$
  $c = Card(\mathcal{S})$
  $majVectApp()$
  Définir f_0^k à partir de $\mathcal{S}$
Jusqu'à $Card(\mathcal{S}) = c$
$f_0 \leftarrow f_0^k$
$n = n + 1, k = 0$
$\mathcal{S} \leftarrow \emptyset$

// Boucles suivantes
Faire
  Pour tout $x \in SV(f_{n-1})$ **Faire**
    $\mathcal{S} \leftarrow \mathcal{S} \cup$ *coupleLabelOpposé(x)*
  Fin pour
  Faire
    $k = k + 1$
    $c = Card(\mathcal{S})$
    $majVectApp()$
    Définir f_n^k à partir de $\mathcal{S}$
  Jusqu'à $Card(\mathcal{S}) = c$
  $f_n \leftarrow f_n^k$
Jusqu'à ce que le critère d'arrêt global soit respecté

tableau 6.1 donne les principales caractéristiques de l'approximation. La trajectoire comprend 80 pas de temps, en anticipant sur 5 pas de temps, et avec une distance de sécurité $\Delta = 8$.

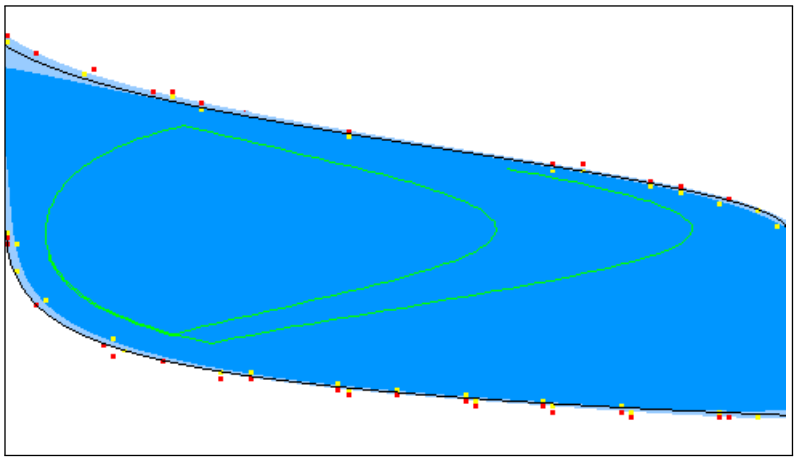


FIG. 6.5 – Approximation finale (en bleu) du noyau de viabilité pour le problème population, avec une grille à la résolution la plus fine de 6561 points (81 points par dimension) et un exemple de trajectoire lourde. L'optimisation est faite sur 3 pas de temps, $dt = 0,012$. Les lignes continues délimitent les frontières du vrai noyau.

La figure 6.6 montre un exemple de déroulement d'une boucle de l'apprentissage actif, en représentant les différentes fonctions f_n^k, les points de la grille adaptative sélectionnés et ajoutés dans la base d'apprentissage. On part des vecteurs de support de la fonction f_{n-1} pour construire f_n^0 et on met progressivement à jour le vecteur $\mathcal{S}$, jusqu'à ce que la fonction f_n^k ne fasse plus d'erreurs sur la grille à la résolution la plus fine. On constate que parfois, la taille de la base d'apprentissage à l'itération $k+1$ est inférieure à $(Card(\mathcal{S}) + 2\times$ nombre de mal-classés à l'itération $k)$. Cela est dû au fait que, dans la procédure de recherche du couple opposé, les points que l'on veut ajouter à la base d'apprentissage appartiennent déjà à $\mathcal{S}$ ou qu'ils sont en dehors de K.

Cet exemple montre certains avantages de l'algorithme d'approximation utilisant une technique d'apprentissage actif, sur une grille adaptative : bonne qualité de l'approximation, facilité de définir des politiques d'actions et taille de la base d'apprentissage limité. Au maximum, la base d'apprentissage a une taille inférieure à 2% de la taille de la grille $\mathcal{G}_{J_M}$, et moins de 12% de ces points ont été étiquetés au maximum pour une itération. Dans cet exemple, les dynamiques sont simples et le nombre de vecteurs de support de l'approximation finale est faible, comme le temps pour évaluer une action (pour le calcul de la trajectoire).

6.4.2 Un problème de capturabilité en dimension 2 : la voiture sur la colline

On considère maintenant le problème de la voiture sur la colline. La description complète du problème est donnée dans le chapitre 4.

La figure 6.7 présente un exemple d'approximation par l'extérieur des contours de la fonction valeur du problème. Les points situés sur la figure sont ceux qui ont été utilisés pour calculer la dernière fonction SVM. Le tableau 6.1 donne les principales caractéristiques de l'approximation. On constate que, comme pour le problème population, le nombre maximal de points utilisé pour approcher les contours est très faible (environ $4,5\%$ de la taille totale de la grille).

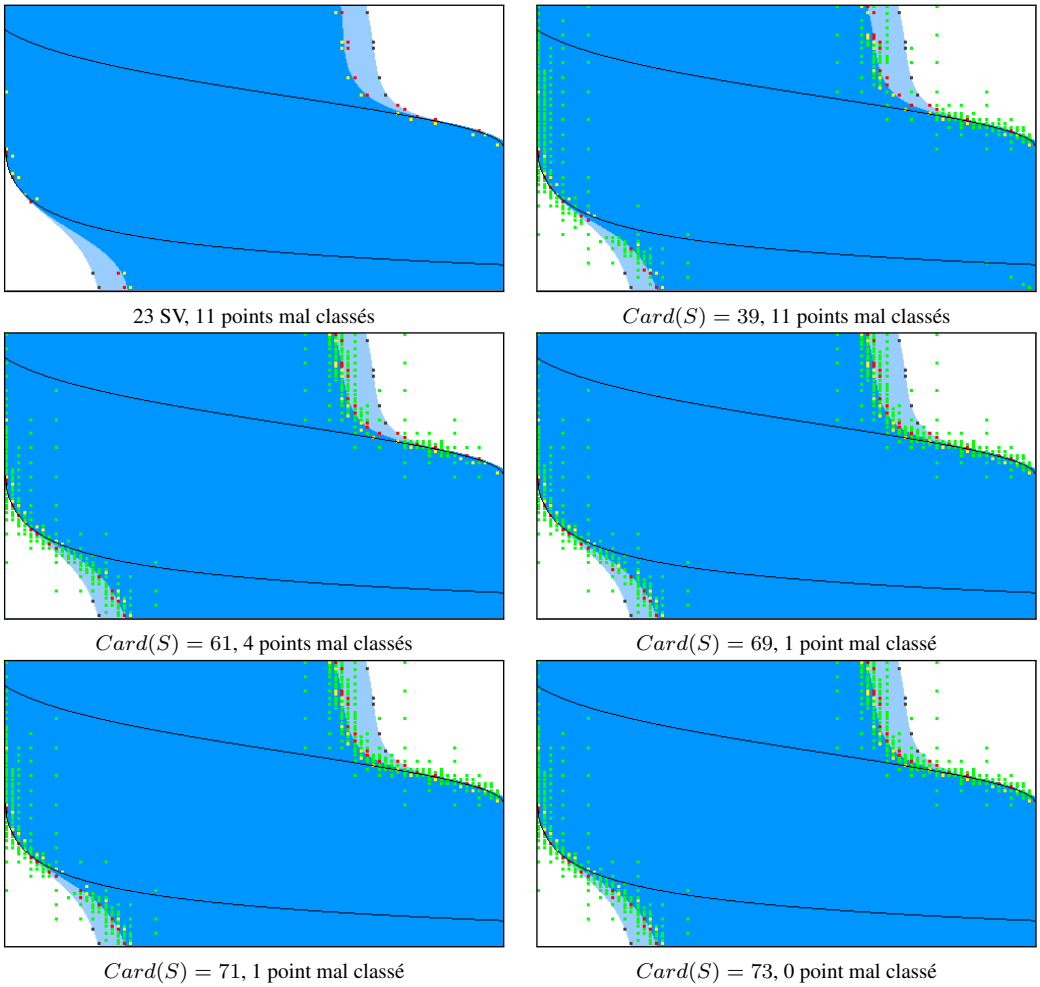


FIG. 6.6 – Exemple de déroulement d'une itération de l'apprentissage actif pour le problème population, en utilisant une grille à 4 niveaux de développement. L'approximation donnée par f_n^k est représentée en bleu foncé et celle définie par f_{n-1} est l'union des deux zones bleues. Les vecteurs de support de la fonction f_{n-1} sont représentés en noir, la base d'apprentissage est composée des points en rouge (étiquette $+1$) et en jaune (étiquette -1), les points de la grille adaptative qui respectent les deux critères de développement sont en vert.

6.4.3 Un problème en dimension 6 : le vélo dans un circuit

Certains chercheurs ont testé leur algorithme sur le problème d'équilibrer un vélo, tout en le conduisant jusqu'à une cible [Randlov et Alstrom, 1998 ; Lagoudakis et Parr, 2003]. Ils ont résolu le problème en utilisant des techniques d'apprentissage par renforcement. Les équations modélisent un vélo, avec 6 variables d'état :

- l'angle du déplacement du guidon par rapport à la verticale $\theta \in [-\frac{\pi}{2}; \frac{\pi}{2}]$ radians ;
- la vitesse de l'angle de déplacement du guidon $\dot{\theta} \in [-6; +6]$ radians/seconde ;
- l'angle du vélo par rapport à la verticale $\omega \in [-\frac{\pi}{15}; \frac{\pi}{15}]$. Si l'angle n'est pas compris dans l'intervalle alors le cycliste chute ;
- la vitesse de l'angle par rapport à la verticale $\dot{\omega} \in [-0,75; 0,75]$ radians ;
- la position de la roue avant $x \in [-2; +2]$ mètres, intervalle qui correspond à la largeur du circuit ;
- l'angle d'inclinaison du centre de la masse du vélo $\phi \in [-\frac{\pi}{2}; \frac{\pi}{2}]$ radians.

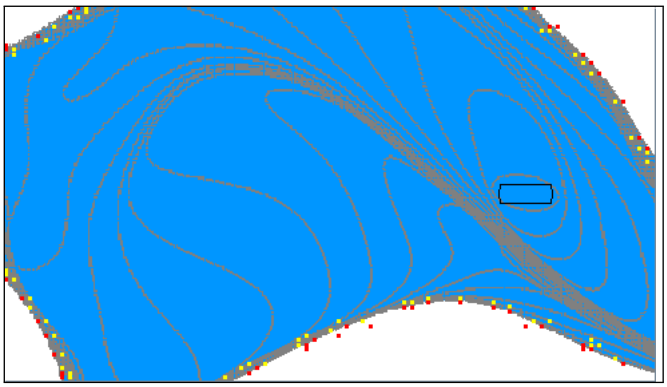


FIG. 6.7 – Approximation finale (en bleu) du bassin de capture et des contours de la fonction valeur (en gris) pour le problème de la voiture sur la colline, avec une grille à la résolution la plus fine de 6561 points (81 points par dimension). L'optimisation est faite avec 8 pas de temps, $dt = 0,015$.

Problème	Population	Voiture	Vélo
Nombre de points de la grille grossière	121	121	15625
Nombre de points de la grille la plus fine	6561	6561	531441
Profondeur maximale de l'arbre	4	4	2
Nombre d'apprentissages moyen par itération	6	8	21
Nombre de SVs de l'approximation finale	28	32	3914
Taille maximale de la base d'apprentissage	124	294	34028
Nombre maximal de points testés	761	1024	258900
Temps	1 min	30 min	$\approx$ 3 j

TAB. 6.1 – Paramètres et résultats des simulations pour les problèmes population, voiture sur la colline et vélo.

Le vélo peut être contrôlé en modifiant la poussée sur le guidon ($T \in [-2; 2]$ newtons) et en déplaçant le centre de sa masse ($d \in [-2; 2]$ cm). On considère ici le même système (pour la description de la dynamique, le lecteur peut se référer à [Randlov et Alstrom, 1998]), mais au lieu d'apprendre à atteindre une cible, on suppose ici que l'on veut conduire le vélo dans un circuit. Tout au long du parcours, la vitesse est constante. Notons deux autres différences dans la définition du problème : on considère ici les contrôles comme continus, et nous n'avons pas introduit de bruit sur la variable de contrôle d (bruit qui sert à simuler un équilibre imparfait).

6.4.3.1 Équilibrer le vélo

Le premier problème est d'équilibrer le vélo afin que le cycliste ne tombe pas et reste dans un certain chemin ($x \in [-2; +2]$ mètres). Cet objectif étant modélisé par l'ensemble des contraintes K, on calcule alors le noyau de viabilité du système. Le tableau 6.1 donne les détails sur la procédure utilisée. On remarque que la taille maximale de la base d'apprentissage est relativement faible par rapport à la taille de la grille à la plus fine résolution ($6,5\%$ des points ont été utilisés au maximum pour construire une fonction SVM). Ce ratio n'est pas aussi bon que celui obtenu avec le problème population : la principale raison est que la grille adaptative est de profondeur 2. L'algorithme présenté ici permet de diminuer de façon drastique la taille de la base d'apprentissage, mais le temps pour calculer la solution est toujours exponentiel avec la dimension, car on vérifie tous les points autour de la frontière des fonctions SVMs, et les dynamiques sont complexes. Le temps d'apprentissage des SVMs devient alors négligeable par rapport au temps de construction des arbres. Plusieurs jours de simulation ont été nécessaires pour obtenir une approximation du noyau.

6.4.3.2 Contrôler le vélo dans un circuit

Le vélo est ensuite contrôlé grâce à la SVM dans un circuit en dimension 2. Le circuit comprend deux parties : lignes droites et virages. Le contrôleur voit le circuit dans les coordonnées relatives du vélo, en considérant la direction tangente du circuit par rapport à la position du vélo. Afin de contrôler le vélo dans les virages, il suffit de mettre à jour l'angle du centre de la masse ϕ en fonction de la pente du virage. Ainsi, le contrôleur est opérationnel sur n'importe quel circuit, à partir du moment où les virages ne sont pas trop serrés.

La figure 6.8 représente un exemple de trajectoire du vélo, à partir d'un état initial en dimension 6 $x_0 = (0, 0, 0, 0, 0, 0)$ et d'un contrôle initial $u_0 = (0, 0)$. On montre la trajectoire, à partir d'un point de départ x'_0 situé dans une ligne droite du circuit, durant 800 pas de temps. Lorsque le vélo approche une zone dangereuse (les limites du circuit ou les limites du noyau de viabilité), les contrôles sont modifiés et le vélo commence à tourner, afin de conserver le système viable. On remarque que, dans la ligne droite du circuit, le vélo va tout droit : les contrôles lui permettent d'être en sécurité et n'ont donc pas besoin d'être modifiés. Mais lorsqu'il approche un virage, il doit modifier sa trajectoire en mettant à jour les contrôles, et le vélo commence à tourner.

FIG. 6.8 – Exemple de trajectoire du vélo dans un circuit, avec comme point de départ x'_0, et durant 800 pas de temps. Le circuit est représenté en blanc. Le contrôleur est défini en utilisant une marge de sécurité $\Delta = 2$, en anticipant sur 4 pas de temps.

Même si le calcul du noyau de viabilité est long, la fonction SVM fournit un contrôleur parcimonieux qui permet de déterminer rapidement des politiques de contrôle à chaque pas de temps.

6.5 Limites de l'algorithme - perspectives

Si on ne garde pas en mémoire tous les points associés à leur étiquette, la principale caractéristique de la procédure est que l'on calcule à chaque itération les étiquettes de tous les exemples qui sont proches de la frontière de la SVM courante f_n^k. Si la surface ne change pas trop entre 2 itérations, on refait les mêmes calculs plusieurs fois, ce qui peut être très cher en terme de temps de calcul, spécialement lorsque les dynamiques du modèle sont complexes, et lorsque l'optimisation est faite sur plusieurs pas de temps. Un compromis intéressant est de stocker en mémoire les points de la grille grossière pour lesquels l'arbre a été complètement développé, ou le premier point mal-classé sinon. Généralement, cette liste est de taille bien inférieure au nombre de points testés. Grâce à cette information, on peut utiliser la fonction SVM précédente pour étiqueter directement les points. Si un point de la grille grossière x est dans la liste des points bien classés, alors la fonction SVM précédente donne directement les étiquettes de tous les points de l'arbre. Si une feuille issue de x est dans la liste des points mal classés, alors la fonction SVM précédente donne directement l'étiquette de tous les points de l'arbre situés au-dessus de lui. Si un point n'appartient à aucune des deux listes, les tests doivent être effectués entièrement. En utilisant cette approche, le calcul total d'une étiquette est effectué uniquement une fois pour chaque point.

Dans l'algorithme proposé, la grille est développée complètement à chaque fois que l'on se trouve près de la frontière. Cependant, la vérification au niveau le plus fin n'est pas toujours totalement nécessaire, notamment lorsque la frontière bougera à l'instant suivant. Il serait intéressant de définir une discrétisation fonction de la précision nécessaire, en s'inspirant des procédures issues de la programmation dynamique (par exemple [Munos et Moore, 1999 ; Munos et Moore, 2002]).

Conclusion

Bilan

Dans cette thèse, nous avons fait le pari que les machines à vecteurs de support (SVMs) permettent de développer de nouvelles méthodes, plus efficaces, pour résoudre des problèmes de viabilité. En effet, comme leurs excellentes performances sur les problèmes de discrimination l'indiquent, elles permettent de représenter des variétés complexes dans des espaces de dimension importante, de manière très parcimonieuse. En proposant un rapide bilan des réalisations principales de notre travail, nous tentons maintenant d'apprécier dans quelle mesure les espoirs suscités par ce pari étaient fondés.

Nous avons tout d'abord proposé un algorithme d'approximation de noyau de viabilité qui utilise une méthode de discrimination pour représenter le noyau. Nous avons établi des conditions sur la méthode d'apprentissage qui garantissent la convergence de l'approximation vers le noyau de viabilité lorsque la résolution de la grille d'apprentissage tend vers 0. A partir des approximations obtenues, nous avons introduit des contrôleurs lourds qui permettent de définir des politiques de contrôle qui gardent le système viable. Nous avons testé l'algorithme en utilisant les SVMs comme cas particulier de méthode d'apprentissage.
Nous avons adapté cet algorithme à l'approximation de bassins de capture. Pour ce type de problème, nous avons défini deux variantes de l'algorithme : l'une approche le bassin de capture par l'extérieur et l'autre par l'intérieur. Nous avons montré que cette dernière approximation permet de construire aisément un contrôleur optimal : à partir d'un point contenu dans le bassin de capture du système, le contrôleur fournit une séquence de contrôles qui permet d'atteindre la cible en un temps minimal. L'approximation par l'intérieur garantit que le système atteint bien la cible. En outre, la comparaison des résultats entre les deux variantes permet d'estimer empiriquement l'erreur d'approximation de l'algorithme sur un exemple donné.
Nous avons étendu les résultats obtenus pour l'approximation de bassins de capture aux problèmes de minimisation d'une fonction de coût. Nous avons considéré le calcul des valeurs de résilience dans ce contexte. Lorsque la dérivée de la fonction de coût par unité de temps est non-nulle et négative, nous avons défini un algorithme utilisant des SVMs qui permet d'estimer les valeurs de la résilience dans l'espace d'état initial.

Nous avons testé nos algorithmes sur de nombreux cas d'application, pour des problèmes dont les dimensions de l'espace d'état allaient de 2 à 6, et dans des espaces de contrôles de 1 à 17 dimensions.

Nous n'avons pas pu procéder à des comparaisons systématiques avec d'autres méthodes. Cependant, il apparaît que nos méthodes, en utilisant les SVMs présentent les avantages suivants :

Utiliser des SVMs pour approcher un noyau de viabilité ou un bassin de capture permet de vaincre la malédiction de la dimensionnalité sur l'espace des contrôles.
En effet, l'utilisation des SVMs comme méthode de discrimination permet de définir une sorte de fonction barrière sur l'approximation du noyau de viabilité ou du bassin de capture. Cette propriété permet d'utiliser une méthode d'optimisation, par exemple une descente de gradient, pour trouver des contrôles viables. Ces méthodes sont beaucoup plus rapides que l'exploration systématique de toutes les valeurs de contrôle possibles. Ainsi, il devient possible de travailler avec des problèmes dans des espaces de

contrôles très importants. De plus, cela permet de considérer plusieurs pas de temps à chaque itération avec un temps de calcul raisonnable, ce qui, sur au moins quelques exemples, réduit l'effet diffusif de l'algorithme et donne des approximations plus fines.

Nos méthodes fournissent des contrôleurs compacts et rapides.
En effet, la frontière du noyau de viabilité ou des bassins de capture d'un système est utilisée pour définir un contrôleur. Représenter cette frontière avec une fonction SVM permet au contrôleur d'hériter de sa propriété de parcimonie : même si la taille de la grille est importante, seulement une partie des points sera utilisée pour définir la frontière. La formulation du contrôleur est légère et pratique à manipuler. De plus, même si le calcul de l'approximation du noyau de viabilité ou des bassins de capture est long, lorsque les systèmes sont en grande dimension et que l'approximation est fine, le temps de calcul d'un contrôle viable ou optimal est faible. En effet, l'évaluation de l'étiquette d'un état (en fonction de s'il appartient ou non à l'approximation du noyau de viabilité ou des bassins de capture) est rapide lorsque le nombre de points vecteurs de support est faible. De plus, nous utilisons une méthode d'optimisation pour déterminer une politique de contrôle, ce qui permet de réduire considérablement le temps de calcul lorsque l'espace des contrôles est important.

Nous avons proposé des méthodes d'apprentissage actif qui utilisent des grilles virtuelles à plusieurs résolutions, et diminuent considérablement la taille des échantillons à garder en mémoire.
En effet, en nous appuyant sur la propriété de parcimonie des SVMs, nous avons proposé une procédure d'apprentissage actif qui sélectionne les points à ajouter dans la base d'apprentissage afin d'en utiliser un nombre qui soit proche du nombre de vecteurs de support. Afin d'obtenir une approximation plus fine du noyau de viabilité ou du bassin de capture, nous combinons une procédure d'apprentissage actif avec une grille virtuelle multi-résolution. Les points de la grille ne sont pas stockés mais recalculés à chaque itération. Cette procédure permet de gagner en précision et quelques dimensions. Malheureusement si les gains en taille mémoire sont significatifs, le temps de calcul reste exponentiel. Cependant, les différents calculs nécessaires dans la boucle d'apprentissage actif sont facilement parallélisables, ce qui permet de réduire les temps de calcul.

Notre travail garde un caractère exploratoire, et les perspectives qu'il ouvre nous paraissent aussi importantes que ses réalisations bien établies.

Perspectives

Les pistes de travail les plus importantes ouvertes par notre travail nous paraissent être les suivantes :

Établir un théorème de convergence de l'algorithme utilisant les SVMs.
Tout d'abord, il reste à démontrer qu'il est possible de trouver des paramètres des SVMs tels qu'elles respectent les conditions d'application du théorème. En comparant les SVMs avec l'algorithme des $k - ppv$, nous avons énoncé quelques arguments qui font penser qu'il est possible de toujours trouver de tels paramètres, mais nous n'avons pas établi de démonstration.
De plus, dans la plupart des cas d'applications, nous avons utilisé une version simplifiée de l'algorithme (qui utilise la fonction SVM comme approximation d'une distance algébrique), qui tend à diminuer les temps de calcul et qui, pour certains exemples au moins, réduit l'effet diffusif de l'algorithme. Cependant, la convergence de cette version simplifiée reste à prouver.

Définir une approximation du noyau de viabilité par l'intérieur.
Seule l'approximation par l'extérieur garantit dans tous les cas la convergence vers le noyau de viabilité, lorsque la résolution de la grille d'apprentissage tend vers 0. Il est cependant important de définir un algorithme d'approximation par l'intérieur, même si celui-ci ne converge que dans certains cas vers un domaine de viabilité strictement inclus dans le noyau de viabilité. En effet, une telle approximation par l'intérieur permet de définir des contrôleurs qui garantissent la viabilité.

Étendre les algorithmes aux jeux dynamiques.
Une hypothèse de départ de ce travail était que le système est déterministe. Il serait intéressant d'étendre

les procédures proposées aux jeux dynamiques, où l'on introduit une incertitude v et où l'on cherche les états viables (ou capturant la cible) quelle que soit l'incertitude. Dans le problème d'eutrophication des lacs dans le chapitre 5, nous avons fait un premier pas dans cette direction mais il reste insuffisant. En effet, les incertitudes sont simplement discrétisées et l'algorithme est donc exponentiel avec la dimension de l'espace des incertitudes. Une piste serait d'utiliser une procédure similaire à celle de recherche d'un contrôle viable en utilisant une méthode d'optimisation, sauf qu'ici on ne recherche pas s'il existe un contrôle qui permet de rester viable au pas de temps suivant, mais s'il en existe au moins un qui engendre une situation non viable. Une démarche semblable pourrait également être définie pour l'approximation de bassins de capture et de valeurs de résilience. Ainsi, une première idée d'algorithme est de fixer une incertitude initiale v_0 et de calculer l'approximation du noyau de viabilité du système avec $v = constante = v_0$. A partir de cette approximation initiale, on parcourt les points contenus dans l'approximation et proches de la frontière. Avec $v = v_0$, on utilise une méthode d'optimisation afin de trouver le meilleur contrôle u^* et on réalise ensuite une deuxième optimisation, mais cette fois-ci sur v avec $u = u^*$ afin de trouver v^* qui maximise la distance entre un point et l'approximation. Dès que l'on trouve un point tel que, en appliquant le contrôle u^* et v^*, la trajectoire sort « franchement » du noyau de viabilité courant, on réalise une deuxième boucle de l'algorithme d'approximation avec $v =$ constante $= v^*$. Et ainsi de suite, jusqu'à ce qu'il n'existe plus aucune valeur de v qui produise des points non-viables. Bien sûr, les conditions d'application de cet algorithme et les preuves de convergence sont encore à déterminer.

Améliorer la procédure d'apprentissage actif.
Des modifications de la procédure d'apprentissage actif sont à envisager pour gagner encore quelques dimensions. Tout d'abord, il faudra certainement abandonner l'idée d'avoir des approximations précises dans ces espaces. De plus, on peut envisager d'utiliser des grilles à discrépance[1] faible au lieu de grilles régulières. Cependant, même en intégrant ces modifications, l'algorithme tel qu'il est présenté ne suffira pas pour résoudre des problèmes en très grande dimension, et d'autres procédures doivent être définies.

Proposer d'autres algorithmes pour vaincre la malédiction de la dimensionnalité.
Pour l'instant, la procédure proposée et les pistes de travail suggérées ne sont pas décisives : elles permettent de gagner quelques dimensions mais pas de résoudre des problèmes en dimension très importante (il paraît difficile qu'elles puissent traiter des problèmes en dimension 15 par exemple). Les SVMs semblent un outil très intéressant pour continuer à explorer d'autres pistes mais les algorithmes seront certainement très différents de ceux présentés ici.

Alors, maintenir la viabilité ou la résilience d'un système : les machines à vecteurs de support permettront-elles de conjurer la malédiction de la dimensionnalité dans l'espace d'état ? Quelques éléments de réponse se dessinent mais le suspense reste entier. Le projet européen PATRES[2] et le projet ANR Déduction[3] permettront, nous l'espérons, de progresser sur ces pistes de recherches que nous avons initiées.

[1] La discrépance représente l'écart d'une grille à l'uniformité parfaite.
[2] http://www.patres-project.eu
[3] http://mann/deduction/

Bibliographie

[Aubin, 1991] AUBIN, J.-P. (1991). *Viability theory*. Birkhäuser.

[Aubin, 1997] AUBIN, J.-P. (1997). *Dynamic economic theory: a viability approach*. Springer-Verlag.

[Aubin, 2001] AUBIN, J.-P. (2001). Viability kernels and capture basins of sets under differential inclusions. *SIAM Journal on Control and Optimization*, 40(3):853–881.

[Aubin, 2002] AUBIN, J.-P. (2002). An introduction to viability theory and management of renewable resources. *Dans Coupling Climate and Economic Dynamics, Geneva*.

[Aubin *et al.*, 2001] AUBIN, J.-P., BONNEUIL, N., MAURIN, F. et SAINT-PIERRE, P. (2001). Viability of pay-as-you-go systems. *Journal of Evolutionary Economics*, 11(5):555–571.

[Aubin et Frankowska, 1985] AUBIN, J.-P. et FRANKOWSKA, H. (1985). Heavy viable trajectories of controlled systems. *Annales de l'institut Henri Poincaré (C) Analyse non linéaire*, 5:371–395.

[Aubin et Frankowska, 1996] AUBIN, J.-P. et FRANKOWSKA, H. (1996). The viability kernel algorithm for computing value functions of infinite horizon optimal control problems. *Journal of mathematical analysis and applications*, 201(2):555–576.

[Aubin *et al.*, 2005] AUBIN, J.-P., PUJAL, D. et SAINT-PIERRE, P. (2005). *Numerical Methods in Finance*, chapitre Dynamic Management of Portfolios with Transaction Costs under Tychastic Uncertainty. Springer US.

[Bellman, 1957] BELLMAN, R. (1957). *Dynamic Programming*. Princeton University Press.

[Bellman, 1961] BELLMAN, R. (1961). *Adaptive Control Process*. Princeton University Press.

[Bennett *et al.*, 2001] BENNETT, E.-M., CARPENTER, S.-R. et CARACO, N.-F. (2001). Human impact on erodable phosphorus and eutrophication: A global perspective. *Bioscience*, 51:227–234.

[Bertsekas, 2005] BERTSEKAS, D. (2005). *Dynamic Programming and Optimal Control*, volume 1 et 2. Athena Scientific, Nashua.

[Béné *et al.*, 2001] BÉNÉ, C., DOYEN, L. et GABAY, D. (2001). A viability analysis for a bio-economic model. *Ecological Economics*, 36(3):385–396.

[Bokanowski *et al.*, 2006] BOKANOWSKI, O., MARTIN, S., MUNOS, R. et ZIDANI, H. (2006). An anti-diffusive scheme for viability problems. *Applied Numerical Mathematics*, 56(9):1147–1162.

[Bonneuil, 2003] BONNEUIL, N. (2003). Making ecosystem models viable. *Bulletin of Mathematical Biology*, 65(6):1081–1094.

[Bonneuil, 2006] BONNEUIL, N. (2006). Computing the viability kernel in large state dimension. *Journal of Mathematical Analysis and Applications*, 323:1444–1454.

[Boser *et al.*, 1992] BOSER, B.-E., GUYON, I. et VAPNIK, V. (1992). A training algorithm for optimal margin classifiers. *Dans Proceedings of the Fifth Annual Workshop on Computational Learning Theory, Pittsburgh, USA*, pages 144–152.

[Burges, 1998] BURGES, C. (1998). A tutorial on Support Vector Machines for pattern recognition. *Data Mining and Knowledge Discovery*, 2(2):121–167.

[Campbell *et al.*, 2000] CAMPBELL, C., CRISTIANINI, N. et SMOLA, A. (2000). Query learning with large margin classifiers. *Dans Proceedings of the 17th International Conference on Machine Learning*, pages 111–118. Morgan Kaufmann, San Francisco, CA.

[Cardaliaguet, 1994] CARDALIAGUET, P. (1994). *Domaines discriminants en jeux différentiels*. Thèse de doctorat, Université de Paris 09.

[Cardaliaguet *et al.*, 1997] CARDALIAGUET, P., QUINCAMPOIX, M. et SAINT-PIERRE, P. (1997). Optimal times for constrained nonlinear control problems without local optimality. *Applied Mathematics & Optimization*, 36:21–42.

[Cardaliaguet *et al.*, 1998] CARDALIAGUET, P., QUINCAMPOIX, M. et SAINT-PIERRE, P. (1998). *Set-Valued Numerical Analysis for Optimal control and Differential Games*. Annals of the International Society of Dynamic Games.

[Cardaliaguet *et al.*, 2000] CARDALIAGUET, P., QUINCAMPOIX, M. et SAINT-PIERRE, P. (2000). Numerical schemes for discontinuous value functions of optimal control. *Set-Valued Analysis*, 8:111–126.

[Carpenter *et al.*, 1999a] CARPENTER, S.-R., BROCK, W. et HANSON, P. (1999a). Ecological and social dynamics in simple models of ecosystem management. *Conservation Ecology*, 3.

[Carpenter *et al.*, 1999b] CARPENTER, S.-R., LUDWIG, D. et BROCK, W.-A. (1999b). Management of eutrophication for lakes subject to potentially irreversible change. *Ecological Applications*, 9(3):751–771.

[Carpenter *et al.*, 2001] CARPENTER, S.-R., WALKER, B., ANDERIES, J.-M. et ABEL, N. (2001). From metaphor to measurement: Resilience of what to what? *Ecosystems*, 4(8):765–781.

[Chang et Lin, 2001] CHANG, C.-C. et LIN, C.-J. (2001). Libsvm: a library for support vector machines. Disponible sur "http://www.csie.ntu.edu.tw/ cjlin/libsvm/".

[Chapel et Deffuant, 2006] CHAPEL, L. et DEFFUANT, G. (2006). SVM viability controller active learning. *Dans Workshop Kernel machines and reinforcement learning - ICML 2006, Pittsburgh, June 26*.

[Chapel et Deffuant, 2007] CHAPEL, L. et DEFFUANT, G. (2007). SVM viability controller active learning: Application to bike control. *Dans Proceedings of the IEEE International Symposium on Approximate Dynamic Programming and Reinforcement Learning (ADPRL'07), Honolulu, USA, April 1-5*, pages 193–200.

[Chapel et Deffuant, Subm] CHAPEL, L. et DEFFUANT, G. (Subm). SVM approximation of value function contours in target hitting problems. *Journal of Mathematical Analysis and Applications*. Soumis.

[Chapel *et al.*, 2005] CHAPEL, L., DEFFUANT, G., MARTIN, S. et MULLON, C. (2005). Defining yield policies in a viability theory approach. *Dans 5th European Conference on Ecological Modelling (ECEM), Pushchino, Russie, September 19-23*, pages 35–36.

[Chapel *et al.*, Acc] CHAPEL, L., DEFFUANT, G., MARTIN, S. et MULLON, C. (Acc). Defining yield policies in a viability theory approach. *Ecological Modelling*, Special Issue on the Fifth European Conference on Ecological Modelling - Selected Papers from the Fifth European Conference on Ecological Modelling, September 19 - 23, 2005. Accepté.

[Chapel *et al.*, 2007] CHAPEL, L., MARTIN, S. et DEFFUANT, G. (2007). Lake eutrophication: using resilience evaluation to compute sustainable policies. *Dans Proceedings of the 10th international conference on environmental science and technology (CEST 2007), Kos Island, Greece, September 5-7*, pages A204–A211.

[Cohn *et al.*, 1995] COHN, D.-A., GHAHRAMANI, Z. et JORDAN, M.-I. (1995). Active learning with statistical models. *Dans* TESAURO, G., TOURETZKY, D. et LEEN, T., éditeurs : *Advances in Neural Information Processing Systems*, volume 7, pages 705–712. The MIT Press.

[Coquelin *et al.*, 2007] COQUELIN, P. A., MARTIN, S. et MUNOS, R. (2007). A dynamic programming approach to viability problems. *Dans Proceedings of the IEEE International Symposium on Approximate Dynamic Programming and Reinforcement Learning (ADPRL'07)*.

[Coursol, 2007] COURSOL, F. (2007). Implémentation d'un logiciel de visualisation 3d de noyaux de viabilité. Mémoire de Master, Université Blaise Pascal.

[Crandall et Lions, 1983] CRANDALL, M.-G. et LIONS, P.-L. (1983). Viscosity solutions of hamilton-jacobi equations. *Transactions of the American Mathematical Society*, 277(1):1–42.

[Cristianini et Shawe-Taylor, 2000] CRISTIANINI, N. et SHAWE-TAYLOR, J. (2000). *Support Vector Machines and other kernel-based learning methods*. Cambridge University Press.

[Cury *et al.*, 2005] CURY, P.-M., MULLON, C., GARCIA, S.-M. et SHANNON, L.-J. (2005). Viability theory for an ecosystem approach to fisheries. *ICES journal of marine science*, 62(3):577–584.

[De Lara *et al.*, 2007] DE LARA, M., DOYEN, L., GUILBAUD, T. et ROCHET, M.-J. (2007). Is a management framework based on spawning-stock biomass indicators sustainable? a viability approach. *ICES Journal of Marine Science*, 64(4):761–767.

[Deffuant *et al.*, 2007] DEFFUANT, G., CHAPEL, L. et MARTIN, S. (2007). Approximating viability kernels with support vector machines. *IEEE transactions on automatic control*, 52(5):933–937.

[Deffuant *et al.*, 2005] DEFFUANT, G., MARTIN, S. et CHAPEL, L. (2005). Utiliser des "support vector machines" pour apprendre un noyau de viabilité. *Dans Actes de la conférence Majecstic 2005, Rennes, France, November 16-18*, pages 195–201.

[Dent *et al.*, 2002] DENT, C., CUMMING, G.-S. et CARPENTER, S.-R. (2002). Multiple states in river and lake ecosystems. *Philosophical transactions of the Royal society B: Biological Sciences*, 357(1421):635–645.

[Doyen et Saint-Pierre, 1997] DOYEN, L. et SAINT-PIERRE, P. (1997). Scale of viability and minimal time of crisis. *Set-Valued Analysis*, 5(3):227–246.

[Fan *et al.*, 2005] FAN, R.-E., CHEN, P.-H. et LIN, C.-J. (2005). Working set selection using second order information for training svm. *Journal of Machine Learning Research*, 6:1889–1918.

[Frankowska, 1989] FRANKOWSKA, H. (1989). Optimal trajectories associated with a solution of the contingent hamilton-jacobi equation. *Applied Mathematics and Optimization*, 19(1):291–311.

[Frankowska, 1993] FRANKOWSKA, H. (1993). Lower semicontinuous solutions to hamilton-jacobi-bellman equations. *SIAM Journal on Control and Optimization*, 31(1):257–272.

[Grüne, 1997] GRÜNE, L. (1997). An adaptive grid scheme for the discrete hamilton–jacobi–bellman equation. *Numerical Mathematics*, 75(3):319–337.

[Gunderson et Holling, 2002] GUNDERSON, L. et HOLLING, C.-S. (2002). *Panarchy: understanding transformations in human and natural systems*. Island Press.

[Holling, 1973] HOLLING, C.-S. (1973). Resilience and stability of ecological systems. *Annual Review of Ecology and Systematics*, 4:1–23.

[Holling, 1996] HOLLING, C.-S. (1996). *Engineering within ecological constraints*, chapitre Engineering resilience versus ecological resilience, pages 31–44. National Academy, Washington, D.C.

[Janssen et Carpenter, 1999] JANSSEN, M.-A. et CARPENTER, S.-R. (1999). Managing the resilience of lakes: A multi-agent modeling approach. *Conservation Ecology*, 3(2).

[Jeppesen *et al.*, 1998] JEPPESEN, E., SØNDERGAARD, M., JENSEN, J.-P., MORTENSEN, E. et HANSEN, A.-M. (1998). Cascading trophic interactions from fish to bacteria and nutrients after reduced sewage loading: An 18-year study of a shallow hypertrophic lake. *Ecosystems*, 1(3):250–267.

[Joachims, 1998] JOACHIMS, T. (1998). Text categorization with support vector machines: learning with many relevant features. *Dans Proceedings of ECML-98, 10th European Conference on Machine Learning*.

[Joachims, 1999] JOACHIMS, T. (1999). *Making large-Scale SVM Learning Practical. In Advances in Kernel Methods - Support Vector Learning*, chapitre 11. MIT Press.

[Kalisiak et Van de Panne, 2004] KALISIAK, M. et Van de PANNE, M. (2004). Approximate safety enforcement using computed viability enveloppes. *Dans IEEE International Conference on Robotics and Automation*, volume 5, pages 4289–4294.

[Lagoudakis et Parr, 2003] LAGOUDAKIS, M.-G. et PARR, R. (2003). Reinforcement learning as classification: Leveraging modern classifiers. *Dans ICML '03: Proceedings of the Twentieth International Conference on Machine Learning*, pages 424–431.

[Le Fur *et al.*, 1999] LE FUR, J., CURY, P., LALOË, F., DURAND, M.-H. et CHABOUD, C. (1999). Co-viabilité des systèmes halieutiques. *Nature, Sciences, Sociétés*, 7(2):19–32.

[Lhommeau *et al.*, 2007] LHOMMEAU, M., JAULIN, L. et HARDOUIN, L. (2007). Inner and outer approximation of capture basin using interval analysis. *Dans 4th International Conference on Informatics in Control, Automation and Robotics (ICINCO 2007)*.

[Lopez, 2004] LOPEZ, F. (2004). Approximation de noyaux de viabilité à l'aide de svms. Mémoire de Master, ISIMA.

[Ludwig *et al.*, 2003] LUDWIG, D., CARPENTER, S.-R. et BROCK, W.-A. (2003). Optimal phosphorus loading for a potentially eutrophic lake. *Ecological applications*, 13(4):1135–1152.

[Ludwig *et al.*, 1997] LUDWIG, D., WALKER, B. et HOLLING, C.-S. (1997). Sustainability, stability, and resilience. *Conservation Ecology*, 1(1).

[Martin, 2004] MARTIN, S. (2004). The cost of restoration as a way of defining resilience: a viability approach applied to a model of lake eutrophication. *Ecology and Society*, 9(2).

[Martin, 2005] MARTIN, S. (2005). *La résilience dans les modèles écologiques et sociaux*. Thèse de doctorat, école normale supérieure de Cachan.

[Martinet et Doyen, 2007] MARTINET, V. et DOYEN, L. (2007). Sustainability of an economy with an exhaustible resource: A viable control approach. *Resource and Energy Economics*, 29(1):17–39.

[Martinet *et al.*, Pres] MARTINET, V., THÉBAUD, O. et DOYEN, L. (Pres). Defining viable recovery paths toward sustainable fisheries. *Ecological Economics*. Sous Presse.

[Moore et Atkeson, 1995] MOORE, A. et ATKESON, C. (1995). The parti-game algorithm for variable resolution reinforcement learning in multidimensional state-spaces. *Machine Learning*, 21:199–233.

[Muller *et al.*, 2001] MULLER, K.-R., MIKA, S., RATSCH, G., TSUDA, K. et SCHLKOPF, B. (2001). An introduction to kernel-based learning algorithms. *IEEE Neural Networks*, 12(2):181– 201.

[Mullon *et al.*, 2004] MULLON, C., CURRY, P. et SHANNON, L. (2004). Viability model of trophic interactions in marine ecosystems. *Natural Resource Modeling*, 17(1):27–58.

[Munos et Moore, 1999] MUNOS, R. et MOORE, A. (1999). Variable resolution discretization for high-accuracy solutions of optimal control problems. *Dans Proceedings of the Sixteenth International Joint Conference on Artificial Intelligence*, pages 1348 – 1355.

[Munos et Moore, 2002] MUNOS, R. et MOORE, A. (2002). Variable resolution discretization in optimal control. *Machine Learning Journal*, 49:291–323.

[Pimm, 1984] PIMM, S. (1984). The complexity and stability of ecosystems. *Nature*, 307:321–326.

[Platt, 1999] PLATT, J.-C. (1999). *Advances in Kernel Methods - Support Vector Learning*, chapitre Fast training of support vector machines using sequential minimal optimization, pages 185–208. MIT Press.

[Pujal et Saint-Pierre, 2004] PUJAL, D. et SAINT-PIERRE, P. (2004). L'algorithme du bassin de capture appliqué à l'évaluation d'options. *Finance*, 25(1):75–106.

[Quincampoix, 1991] QUINCAMPOIX, M. (1991). Frontières de domaines d'invariance et de viabilité pour des inclusions différentielles avec contraintes. *Comptes-Rendus de l'Académie des Sciences. Série I*, 311:904–914.

[Quincampoix et Saint-Pierre, 1995] QUINCAMPOIX, M. et SAINT-PIERRE, P. (1995). An algorithm for viability kernels in hölderian case: approximation by discrete dynamical systems. *Journal of Mathematical Systems, Estimation, and Control*, 5:1–13.

[Randlov et Alstrom, 1998] RANDLOV, J. et ALSTROM, P. (1998). Learning to drive a bicycle using reinforcement learning and shaping. *Dans ICML '98: Proceedings of the Fifteenth International Conference on Machine Learning*, pages 463–471.

[Saint-Pierre, 1994] SAINT-PIERRE, P. (1994). Approximation of the viability kernel. *Applied Mathematics & Optimization*, 29(2):187–209.

[Scheffer, 1998] SCHEFFER, M. (1998). *Ecology of shallow lakes*. Chapman and Hall.

[Schohn et Cohn, 2000] SCHOHN, G. et COHN, D. (2000). Less is more: Active learning with Support Vector Machines. *Dans ICML '00: Proceedings of the Seventeenth International Conference on Machine Learning*, pages 839–846.

[Scholkopf et Smola, 2002] SCHOLKOPF, B. et SMOLA, A. (2002). *Learning with Kernels: Support Vector Machines, Regularization, Optimization, and Beyond*. MIT Press, Cambridge, MA, USA.

[Scholkopf *et al.*, 2000] SCHOLKOPF, B., SMOLA, A., WILLIAMSON, R. et BARTLETT, P. (2000). New support vector algorithms. *Neural Computation*, 12:1083–1121.

[Seube *et al.*, 2000] SEUBE, N., MOITIE, R. et LEITMANN, G. (2000). Aircraft take-off in windshear : a viability approach. *Set-Valued Analysis*, 8:163–180.

[Shannon *et al.*, 2003] SHANNON, L.-J., MOLONEY, C.-L., JARRE, A. et FIELD, J.-G. (2003). Trophic flows in the southern benguela during the 1980s and 1990s. *Journal of Marine Systems*, 39:83–116.

[Smith, 1998] SMITH, V. (1998). *Successes, Limitations and Frontiers in Ecosystem Science*, chapitre Cultural eutrophication of inland, estuarine, and coastal waters, pages 7–49. Springer-Verlag. New York.

[Spiteri *et al.*, 2000] SPITERI, R.-J., PAI, D.-K. et ASCHER, U.-M. (2000). Programming and control of robots by means of differential algebraic inequalities. *IEEE Transactions on Robotics and Automation*, 16(2):135–145.

[Tong, 2001] TONG, S. (2001). *Active Learning: Theory and Applications*. Thèse de doctorat, Stanford University.

[Tong et Chang, 2001] TONG, S. et CHANG, E. (2001). Support vector machine active learning for image retrieval. *Dans MULTIMEDIA '01: Proceedings of the ninth ACM international conference on Multimedia*, pages 107–118.

[Tong et Koller, 2000] TONG, S. et KOLLER, D. (2000). Support vector machine active learning with applications to text classification. *Dans* LANGLEY, P., éditeur : *Proceedings of ICML-00, 17th International Conference on Machine Learning*, pages 999–1006, Stanford, US. Morgan Kaufmann Publishers, San Francisco, US.

[Vapnik, 1995] VAPNIK, V. (1995). *The nature of statistical learning theory*. Springer Verlag.

[Vapnik, 1998] VAPNIK, V. (1998). *Statistical learning theory*. Wiley.

[Verrier, 2007] VERRIER, F. (2007). Apprentissage actif de noyau de viabilité. Mémoire de Master, Université Blaise Pascal.

[Walters *et al.*, 1997] WALTERS, C., CHRISTENSEN, V. et PAULY, D. (1997). Structuring dynamic models of exploited ecosystems from trophic mass-balance assessments. *Reviews in Fish Biology and Fisheries*, 7(2):139–172.

Annexe

Annexe A

Logiciel

En collaboration avec deux stagiaires, Florent Lopez [Lopez, 2004] et Frédéric Coursol [Coursol, 2007], j'ai développé un logiciel en Java qui permet d'approcher des noyaux de viabilité, bassins de capture et calculer des valeurs de résilience en utilisant des SVMs. L'application existe sous deux formes : un mode « batch » qui permet de lancer le programme en mode texte et un mode graphique qui lance le programme et fournit une visualisation (en 2 ou 3 dimensions) de l'approximation obtenue. La figure A.1 détaille les différents paquetages de l'application :

- *Outils* regroupe les outils de gestion d'entrées / sorties non présents dans Java ;
- *Dynamic_System* comprend la définition des systèmes dynamiques testés ;
- *LibSVM* contient les classes pour le calcul des SVMs et de l'algorithme d'approximation ;
- *Appli* permet de piloter l'application.

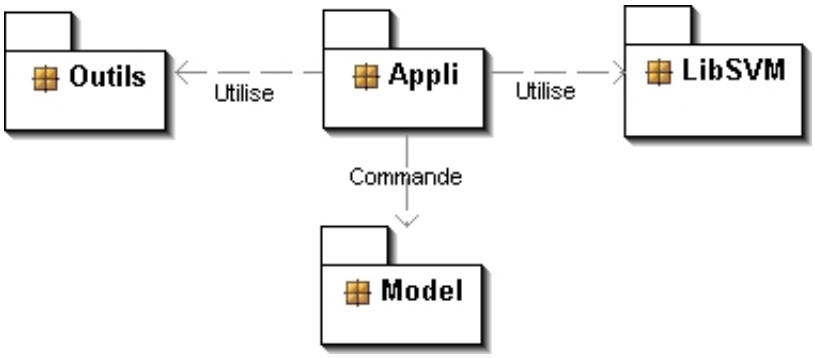


FIG. A.1 – Diagramme des paquetages de l'application

Pour le calcul des différentes fonctions SVMs, nous avons utilisé la librairie *libsvm* [Chang et Lin, 2001] qui implémente l'algorithme de type SMO proposé dans [Fan *et al.*, 2005].

En fonction du système dynamique choisi, l'application donne une approximation du noyau, des bassins de capture et fonctions valeurs ou des valeurs de résilience. Une fois l'approximation obtenue, l'utilisateur peut utiliser le contrôleur adapté (lourd ou optimal). L'utilisateur peut agir à plusieurs niveaux :

- sur le choix et le paramétrage du modèle : avec le choix de la valeur du pas de temps dt notamment ;
- sur le choix et le paramétrage de la SVM : C et σ^2 si on choisit un noyau gaussien ;
- sur la configuration de l'algorithme d'approximation : avec le choix du pas de grille et le nombre de pas pour l'optimisation ;
- sur les paramètres du contrôleur : choix du point de départ, de la valeur de Δ, le nombre de pas

de la trajectoire et le nombre de pas d'anticipation.
Dans les sections suivantes, nous détaillons plus en détail les modes de lancement de l'application.

Mode graphique

La version graphique de l'application permet à l'utilisateur de choisir facilement tous les paramètres liés au système dynamique, à la SVM, à l'algorithme d'approximation. Elle peut être utilisée pour des simulations rapides sur le poste de l'utilisateur, dans le cas de problèmes avec peu de dimensions. La figure A.2 présente l'interface principale. La fenêtre est découpée en 7 parties. Tout en haut de

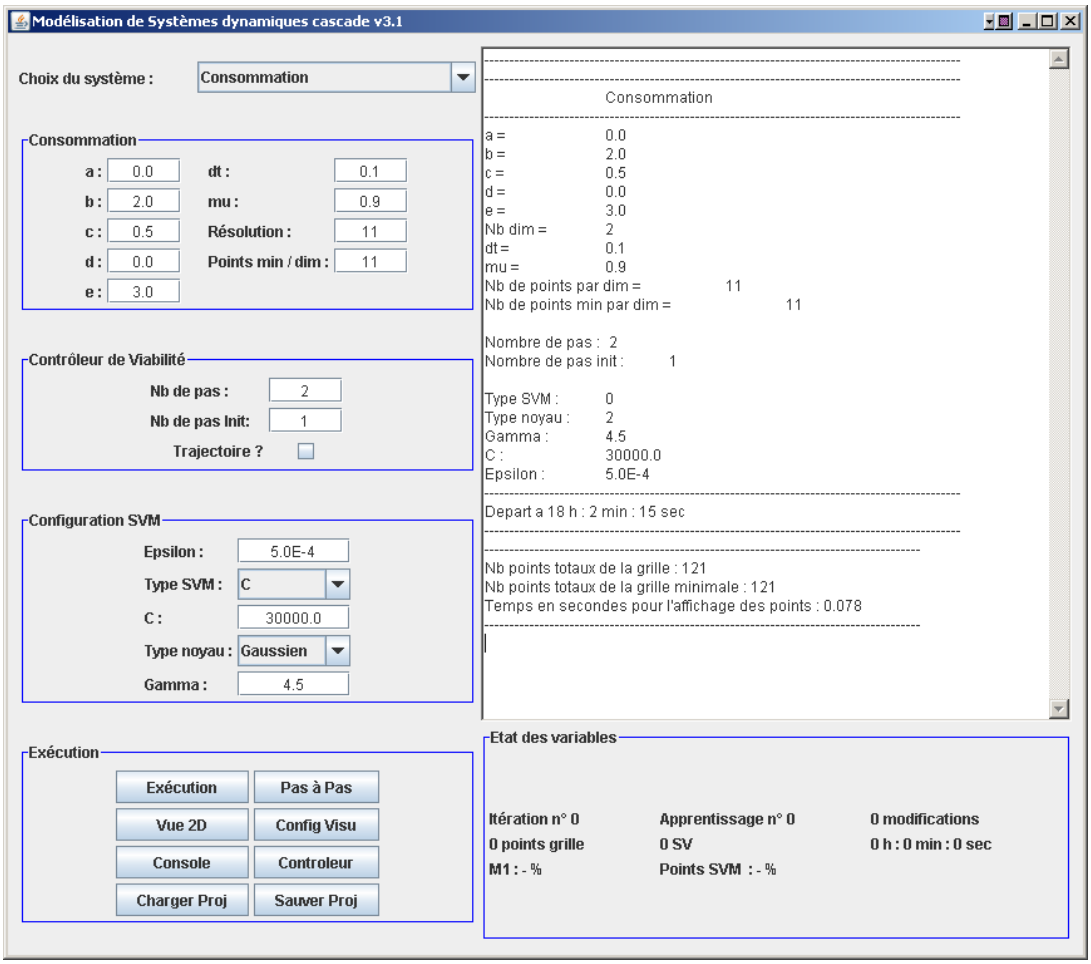


FIG. A.2 – Fenêtre principale

l'interface à gauche, l'utilisateur choisit le système dynamique sur lequel il veut travailler. La partie suivante (appelée *consommation* sur la figure A.2) permet à l'utilisateur d'agir sur différents paramètres du modèle : définition de l'espace des contraintes, pas de temps dt. Il choisit également dans cette partie le nombre de points par dimension de la grille. Ensuite vient la partie sur la configuration de la SVM. En haut à droite de l'interface, une console résume les différents paramètres choisis, et quelques résultats liés à l'approximation. En bas à droite, l'utilisateur trouve des informations générales comme la taille totale de la grille, le nombre d'apprentissage etc..

La dernière partie, appelée *Exécution* permet de piloter l'application. Tout d'abord, deux modes d'exécution sont définis : le bouton *Exécution* lance l'algorithme jusqu'à obtenir l'approximation finale,

le bouton *Pas à pas* permet d'effectuer une itération à la fois et permet ainsi de visualiser le comportement de l'algorithme.
Les boutons *Vue 2D* et *Config Visu* permettent d'afficher la fenêtre de visualisation de l'approximation en deux dimensions, ainsi que la fenêtre de paramétrage de la visualisation. La figure A.4 donne un exemple de fenêtre de visualisation et du panneau de commande associé. Lorsque l'utilisateur appuie

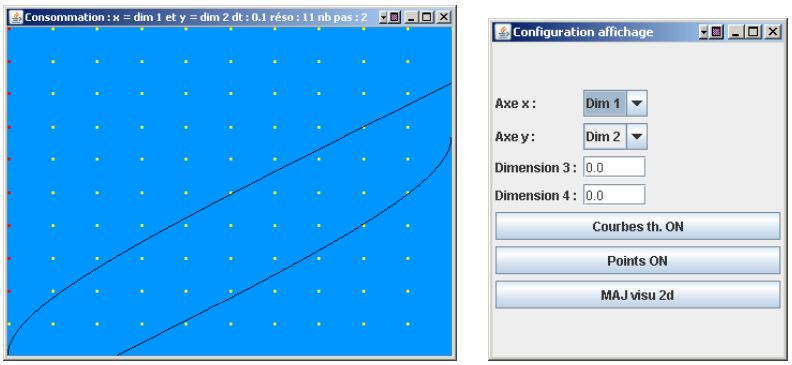


FIG. A.3 – Fenêtre de visualisation de l'approximation du noyau de viabilité ou du bassin de capture en 2 dimensions et panneau de configuration de l'affichage 2d

sur le bouton *Visu 2d*, le label du bouton devient *Visu 3d* et permet de lancer la fenêtre de visualisation en trois dimensions. La figure A.4 présente un exemple de cette fenêtre.

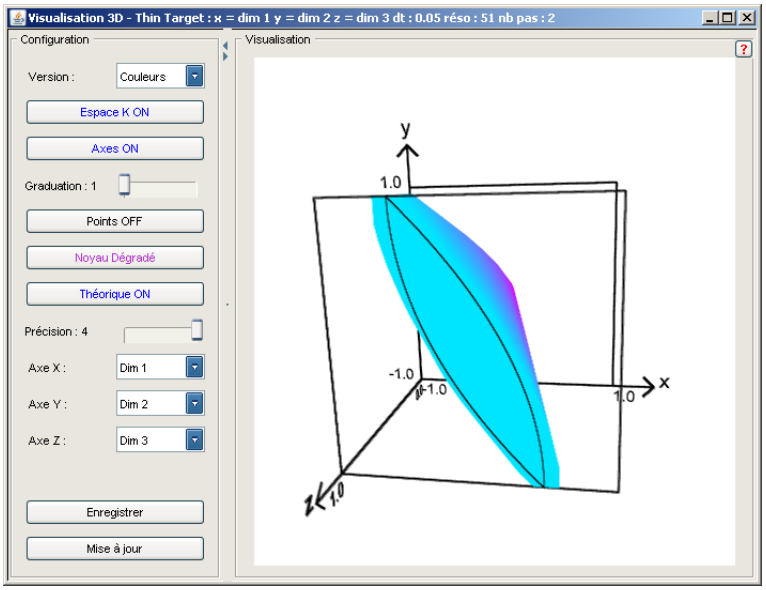


FIG. A.4 – Fenêtre de visualisation de l'approximation du noyau de viabilité ou du bassin de capture en 3 dimensions

Vient ensuite le bouton *console*, qui permet d'enregistrer dans un fichier texte l'affichage situé dans la partie console. Le bouton *Contrôleur* permet l'ouverture d'une fenêtre (figure A.5) qui permet de configurer le contrôleur : choix du point de départ, du nombre de pas de la trajectoire (dans le cas d'un contrôleur lourd), de la valeur du paramètre Δ, du choix ou non du contrôleur prudent (avec le nombre de pas de temps d'anticipation).
Pour terminer, le bouton *Sauver proj* permet de sauvegarder les paramètres choisis par l'utilisateur et l'approximation obtenue. Le bouton *Charger proj* permet de charger les fichiers sauvegardés.

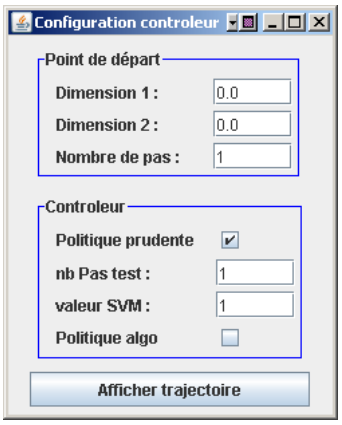


FIG. A.5 – Fenêtre de configuration du contrôleur

Mode batch

L'application existe également en mode batch, ce qui permet d'accélérer le logiciel (plus d'affichage graphique). L'utilisateur construit un fichier en format texte, où il précise les valeurs de tous les paramètres, en suivant un format déterminé. Une fois l'exécution terminée, il peut alors visualiser les résultats en utilisant l'interface graphique et le bouton *Charger projet*.

Ce mode est primordial car il permet de lancer des séries de calcul sur la ferme de calcul du Cemagref, car celle ci ne gère pas les outils graphiques. La ferme peut être utilisée pour lancer plusieurs simulations en même temps, et son utilisation est tout particulièrement intéressante dans les cas où les calculs sont longs et nécessitent des ressources importantes. Elle permet également de mettre en place des procédures de parallélisation. En collaboration avec un stagiaire, Frédéric Verrier [Verrier, 2007], j'ai mis en place une procédure qui parallélise les calculs dans l'algorithme d'apprentissage actif.

Un investissement supplémentaire pour documenter et de nettoyer les différentes versions de cette application, afin de les rendre disponibles comme livrables du projet PATRES.

Table des matières

Liste des tableaux

Table des figures