ÉCOLE DE TECHNOLOGIE SUPÉRIEURE
UNIVERSITÉ DU QUÉBEC

MÉMOIRE PRÉSENTÉ À
L'ÉCOLE DE TECHNOLOGIE SUPÉRIEURE

COMME EXIGENCE PARTIELLE
À L'OBTENTION DE LA
MAÎTRISE EN GÉNIE ÉLECTRIQUE
M. Ing

PAR
Louis-Bernard LAGUEUX

DÉTECTION ET LOCALISATION DES DISCONTINUITÉS DANS LA SIMULATION TEMPS RÉEL DES CIRCUITS D'ÉLECTRONIQUE DE PUISSANCE

MONTRÉAL, LE 4 MAI 2010

PRÉSENTATION DU JURY

CE MÉMOIRE A ÉTÉ ÉVALUÉ

PAR UN JURY COMPOSÉ DE :

M. Bruno De Kelper, directeur de mémoire
Département de Génie Électrique à l'École de technologie supérieure

M. Louis A. Dessaint, président du jury
Département de Génie Électrique à l'École de technologie supérieure

Mme Ouassima Akhrif, membre du jury
Département de Génie Électrique à l'École de technologie supérieure

IL A FAIT L'OBJET D'UNE SOUTENANCE DEVANT JURY ET PUBLIC

LE 7 AVRIL 2010

À L'ÉCOLE DE TECHNOLOGIE SUPÉRIEURE

REMERCIEMENTS

En premier lieu, je tiens à remercier mon directeur de recherche Bruno De Kelper. Ce dernier a tout fait en son possible pour me faciliter la tâche dans ce projet autant au point de vue administratif que pédagogique. À chaque fois que j'en faisais la requête, il trouvait du temps dans son horaire chargé pour m'aider. À plusieurs reprises, lorsque je m'écartais du sujet, il a su fournir les explications nécessaires pour me remettre sur le bon chemin. Finalement, il a été très compréhensif face à mes retards chroniques dans l'écriture de ce mémoire. Du fond du cœur, merci beaucoup !

Ensuite, il est plus que nécessaire de remercier Mélanie et Léanne, car pour écrire ce document j'ai dû m'enfermer d'innombrables heures dans mon bureau et jamais elles n'ont critiqué. Au contraire, elles m'ont toujours encouragé à persévérer. Pour tous les sacrifices que vous avez faits pour moi et tous les encouragements que vous m'avez donnés, je vous remercie énormément.

Finalement, je tiens à remercier mes parents, car depuis le début de mes études ils ont été derrière moi pour m'encourager à persévérer et à me surpasser dans ce que j'entreprenais. Murielle, Pierre, merci infiniment.

DÉTECTION ET LOCALISATION DES DISCONTINUITÉS DANS LA SIMULATION TEMPS RÉEL DES CIRCUITS D'ÉLECTRONIQUE DE PUISSANCE

Louis-Bernard LAGUEUX

RÉSUMÉ

Une des principales sources d'imprécision dans une simulation temps réel des circuits d'électronique de puissance est la commutation des interrupteurs dans le circuit. Dans une simulation à pas fixe, des retards de commutation peuvent être introduits et ceci fausse grandement les résultats. Il est donc nécessaire de bien localiser ces discontinuités afin de pouvoir obtenir des résultats précis.

Plusieurs algorithmes de localisation ont déjà été proposés, mais ces derniers ont généralement deux problèmes communs. Dans un premier temps, le pas suivant la discontinuité doit nécessairement être calculé afin de pouvoir effectuer la localisation d'une discontinuité et, dans un second temps, la méthode de localisation étant rudimentaire, au plus une seule discontinuité peut être localisée par pas de simulation. Si un nombre pair de discontinuité est présent dans un même pas de simulation, la localisation sera impossible.

Un nouvel algorithme permettant de pallier à ces deux problèmes majeurs est donc proposé dans ce document. Ce dernier est basé sur une méthode utilisée dans le domaine de la robotique et utilise les principes de contrôle des systèmes non linéaires. Il permet de localiser, à l'aide des pas précédents, si des discontinuités sont présentes dans le pas suivant. De plus, puisqu'il utilise un polynôme afin de faire la localisation, un nombre de discontinuité égal à l'ordre de ce polynôme peut être localisé dans un même pas.

Cet algorithme a l'avantage d'être complètement indépendant de la méthode numérique de résolution des équations différentielles de la simulation proprement dite. Il peut donc être utilisé avec différentes méthodes, sans pour autant compromettre son efficacité et sa précision.

Des tests permettront de démontrer que ce nouvel algorithme effectue son travail avec une précision qui surpasse les algorithmes existants et ceci sans introduire de grand retard dans la simulation. Ce dernier point est important, car dans une simulation temps réel, les contraintes de temps ne doivent en aucun cas être dépassées et le gain en précision ne doit pas se faire au prix d'une importante charge de travail supplémentaire.

Mots-clés : simulation temps réel, interrupteurs, simulation à pas fixe, retards de commutation, discontinuités, localisation

DETECTION AND LOCALISATION OF DISCONTINUITIES IN REAL TIME SIMULATION OF POWER ELECTRONIC CIRCUITS

Louis-Bernard LAGUEUX

ABSTRACT

One of the main sources of inaccuracies in a real-time simulation of power electronic circuits comes from the switching of electronic switches in the circuit. In a fixed step simulation, switching delays may be experienced that lead to the degradation of the simulation results. Consequently, it is of the utmost importance to locate these discontinuities to improve the overall precision of the simulation.

Many localisation algorithms have been proposed which generally have two common problems. First of all, the time step following the discontinuity must be evaluated in order to locate the discontinuity; secondly, these localisation algorithms being somewhat rudimentary, at most one discontinuity can be localised in any simulation time step. If an even number of discontinuity are present in a given time step, detection is impossible.

A new algorithm that solves these two problems is proposed in this document. It is based on a method used in the field of robotics and uses non-linear system control principles. It detects and locates discontinuities in the following step, based on previous time steps. Moreover, as it uses a polynomial equation for the localisation, it can detect in the time step a number of discontinuities equal to the order of the polynomial.

This algorithm has the merit to be totally independent from the numerical method used to solve the differential equations set of the simulation itself. Therefore, it can be used with different methods without compromising its efficiency and its accuracy.

Further tests will show that this new algorithm meets its goals with a better accuracy then current algorithms, without incurring significant delays in the simulation. The latter is important in real-time simulations, since time-constrains must not be exceeded for any reason, and accuracy must not be attained at the price of an important additional work load.

Keywords: real-time simulation, switches, fixed step simulation, switching delays, discontinuities, localisation

TABLE DES MATIÈRES

LISTE DES TABLEAUX

Page

LISTE DES FIGURES

Page

LISTE DES ABRÉVIATIONS, SIGLES ET ACRONYMES

AB2	Adams-Bashforth d'ordre 2
AB3	Adams-Bashforth d'ordre 3
CAN	Convertisseur Analogique à Numérique
CNA	Convertisseur Numérique à Analogique
EAD	Équation Algébrique Différentielle
EDO	Équation Différentielle Ordinaire
EDO3	Équation Différentielle Ordinaire d'ordre 2
EDO4	Équation Différentielle Ordinaire d'ordre 3
EDO5	Équation Différentielle Ordinaire d'ordre 4
FPGA	Field-programmable gate array
HILS	Hardware In the Loop Simulation
RK2	Runge-Kutta d'ordre 2
RK3	Runge-Kutta d'ordre 3
RK4	Runge-Kutta d'ordre 4
RK45	Runge-Kutta d'ordre 5
TET	Task Execution Time
UCM	Unité de Contrôle pour Moteur

LISTE DES SYMBOLES ET UNITÉS DE MESURE

A	Ampère
D_x	Diode numéro x
e	Erreur
H	Henry
h_k	Pas de simulation numéros k
h_{k+x}	Pas de simulation numéros k+x
Hz	Hertz
I_{D_x}	Courant dans la diode numéro x
L	Inductance
R	Résistance
R_{D_x}	Résistance équivalente de la diode numéro x
t_k	Temps au pas de simulation numéro k
t_{k+x}	Temps au pas de simulation numéro k+x
V_{D_x}	Tension aux bornes de la diode numéro x
V_L	Tension aux bornes de l'inductance
V_R	Tension aux bornes de la résistance
V_{Src}	Tension de la source
x	Vecteur d'états
ϕ	Déphasage de la source
y	Vecteur de sorties
ϖ	Pulsation de la source
μ	Vecteur d'entrées
ΔT	Variation du temps
Ω	Ohm

INTRODUCTION

De nos jours, les outils de simulation mis à la disposition des départements de recherche et de développement jouent un rôle important dans l'évolution des nouveaux produits. Il en est ainsi pour différentes raisons, parmi lesquelles nous retrouvons notamment:

- Les outils de simulation permettent de détecter plus tôt dans l'avancement du produit des anomalies qui étaient, auparavant, détectées sur les prototypes physiques. Ceci permet de réduire grandement le cycle de développement d'un produit.
- La simulation permet d'isoler et de reproduire un phénomène ou une situation donnée aussi souvent que désiré. L'étude de ce phénomène, ou de cette situation, est donc beaucoup plus simple.
- Les risques reliés aux tests impliquant des humains peuvent être grandement réduits en utilisant la simulation. Par exemple, la validation d'un système de contrôle d'un avion est très risquée pour le pilote de test. En utilisant la simulation, il est possible de réduire ces risques au minimum.

Plusieurs preuves supportant ces idées ont été largement diffusées dans la littérature scientifique. Par exemple, BAE Systems (Mathworks, 2006) a réussi, en utilisant des outils de simulation, à réduire le temps de mise en marché d'une radio logiciel en détectant plus tôt dans le processus certains problèmes qui étaient auparavant détectés dans les dernières étapes du cycle de développement. De son côté, Cessna (Mathworks, 2007), a économisé une grande somme d'argent et beaucoup de temps en simulant le comportement d'un avion afin de tester un contrôleur, plutôt qu'en utilisant un véritable appareil. Finalement, en utilisant des outils de simulation, Hitachi (Mathworks, 2005) a été en mesure de faire une preuve de concept en un temps record et ainsi prendre de bonnes décisions dans le développement d'une unité de contrôle pour moteur (UCM). Il est donc catégorique que la simulation soit désormais un outil d'aide à la décision indispensable aux différentes industries afin de rester concurrentielle dans leur secteur d'activité respectif.

Les entreprises œuvrant dans le domaine de l'appareillage électrique utilisé dans de grands réseaux électriques, comme celui d'Hydro Québec, ne font pas exception à cette tendance. Comme les compagnies citées plus haut, elles utilisent les outils de simulation pour réduire le cycle de développement des nouveaux produits. Par exemple, afin de réduire les coûts de développement, ces compagnies utilisent la possibilité de faire interagir un nouvel appareil avec la simulation du système dans lequel il serait censé normalement fonctionner. Ce type de simulation, avec un équipement relié à un simulateur numérique, se nomme, en anglais, *hardware-in-the-loop simulation* (HILS). Elle permet de tester un équipement dans des conditions de fonctionnement autrement compliquées à reproduire. La Figure 0.1 illustre la configuration de base d'une HILS. Dans le simulateur, des modèles mathématiques du système simulé sont exécutés. Ces modèles fournissent des stimuli à l'équipement testé au travers de convertisseurs numériques à analogiques (CNA) et d'un circuit de conditionnement. Ce dernier sert à adapter les signaux de faibles puissances provenant des CNA aux signaux de puissances élevées d'un équipement électrique. Finalement, la réponse à ces stimuli est récupérée à l'aide d'un autre circuit de conditionnement et des convertisseurs analogiques à numériques (CAN).

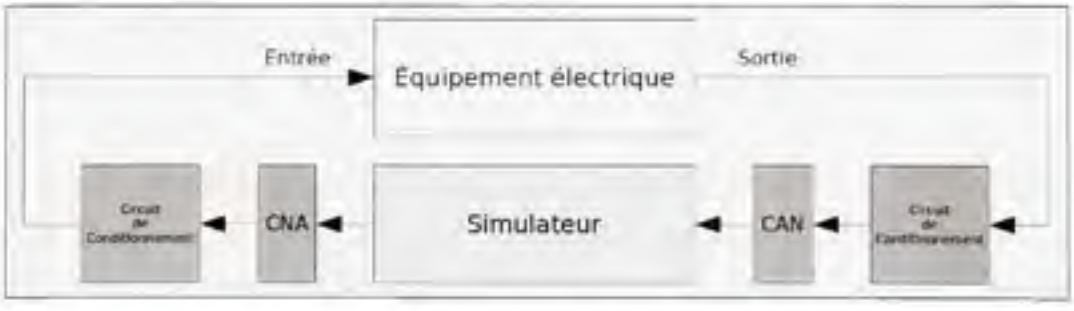

Figure 0.1 Configuration de base d'une HILS.

Dans ce genre de simulation, il est impératif que les équations du modèle mathématique soient résolues aussi rapidement que le ferait le véritable système. Il en va de même pour les mises à jour des sorties du simulateur. Or, ceci est exactement la définition que donne Hartley (Tom T. Hartley, 1994) à une simulation temps réel. Si nous ne respectons pas les

contraintes de temps imposées par le système, les conditions de test ne seront plus optimales et des erreurs pourraient être introduites dans les résultats.

Autrement, lorsque les contraintes de temps sont respectées, le modèle mathématique préalablement établi fournit des résultats en continu. Or il arrive parfois des situations où ce modèle mathématique doit être modifié en cours de simulation. Par exemple, dans un circuit d'électronique de puissance, lorsqu'un interrupteur ouvre un circuit et modifie, par le fait même, la topologie du circuit, le modèle mathématique préalablement établi n'est plus valide et un nouveau système d'équations doit être utilisé pour simuler le circuit dans sa nouvelle topologie. Ces modifications des équations introduisent des discontinuités dans les résultats. Nous sommes donc en présence d'un système hybride : plusieurs sections continues entrecoupées par des discontinuités provoquées par une mise à jour du modèle mathématique.

Nous verrons dans ce qui suit que la simulation temps réel d'un système hybride n'est pas une simple tâche. En effet, ce type de système demande des algorithmes spéciaux qui minimiseront l'effet des discontinuités sur l'exactitude des résultats. Ce document tentera donc de proposer un nouvel algorithme qui minimisera le temps nécessaire à la simulation de ce type de système tout en augmentant l'efficacité de cette dernière en termes de précision des résultats.

Dans un premier temps, la problématique reliée à la simulation en temps réel des systèmes hybrides sera présentée. Ensuite, une revue littéraire sera faite dans le but d'établir l'état de l'art de ce domaine précis de la simulation. Différents algorithmes de simulation seront analysés et, dans chaque cas, une liste des avantages et des inconvénients sera faite. Ceci permettra d'établir les bases du nouvel algorithme qui sera proposé pour améliorer la simulation en temps réel des systèmes hybrides.

Par la suite, les bases théoriques de cet algorithme seront présentées de façon générale, sans prendre en considération la méthode numérique de résolution des équations différentielles ordinaires (EDO). Ce dernier point sera traité au chapitre suivant. Plusieurs méthodes telles

que Runge-Kutta ou Adams-Bashforth seront analysées. Ceci permettra d'établir précisément les étapes de l'algorithme proposé. Des tests fonctionnels, et en temps réel, seront par la suite effectués dans le but de confronter la théorie à la pratique.

CHAPITRE 1

PROBLÉMATIQUE

Normalement, lorsque nous tentons de résoudre une EDO en utilisant une méthode analytique nous obtenons une fonction continue. Par exemple, pour un circuit électrique, nous pouvons obtenir le courant qui parcourt le circuit en fonction du temps. Tel que le démontre la Figure 1.1, avec cette fonction, il est possible d'établir la valeur du courant pour toutes les valeurs du temps. En contrepartie, lorsque nous effectuons une simulation numérique à l'aide d'un ordinateur, nous n'obtenons pas autant d'information. En effet, la mémoire d'un ordinateur étant en quantité limitée, nous obtenons uniquement un ensemble de valeurs sous forme vectorielle représentant la solution du problème. Il est possible de voir sur la Figure 1.2 ce qu'implique la numérisation de la solution d'une EDO.

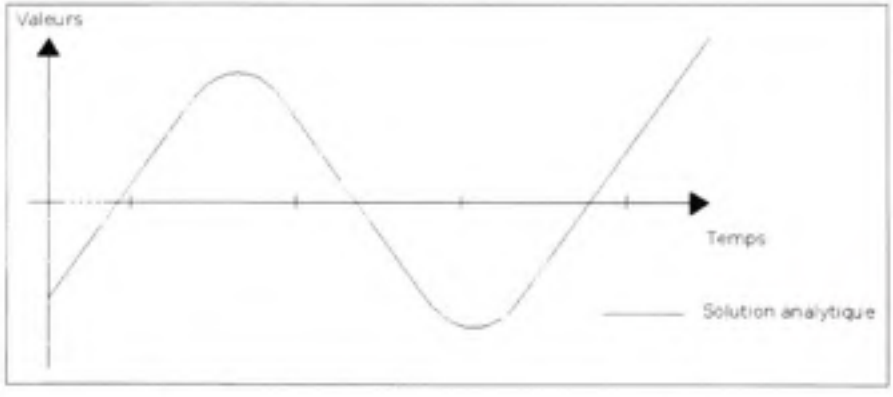

Figure 1.1 Solution analytique à une EDO.

Cette discrétisation des résultats introduit plusieurs complications sur lesquelles nous avons une influence plus ou moins limitée. Trois de ces problèmes seront présentés dans ce chapitre.

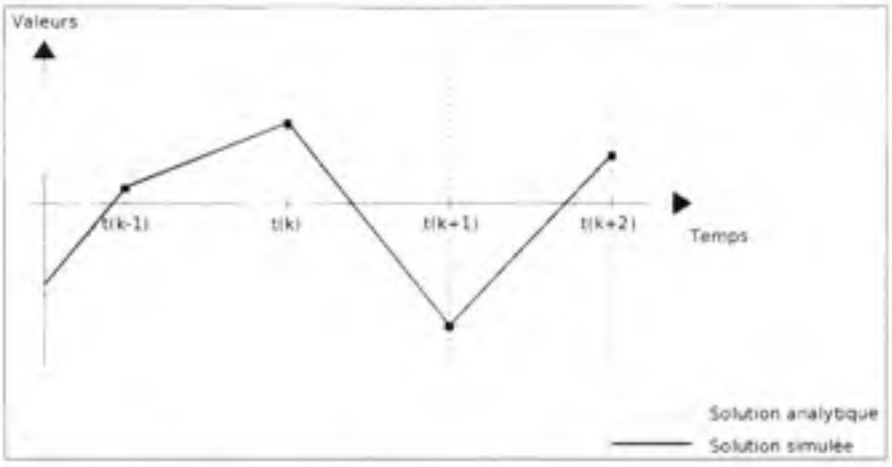

Figure 1.2 Solution numérique d'une EDO.

Le premier problème est relié à la précision des méthodes de résolution des EDO. Chaque méthode sans exception introduit des erreurs d'approximation. Ce type d'erreur dépend notamment de la dynamique du système entre les pas de simulation. Une manière simple de la minimiser serait d'utiliser un pas de calcul le plus petit possible. Toutefois, plus le pas de simulation est petit, plus les efforts de traitement pour calculer un même intervalle de temps sont considérables. Nous sommes donc en présence d'une imprécision sur laquelle nous avons une influence limitée. Le seul contrôle que nous avons sur cette complication est au niveau du choix de méthode de résolution du modèle mathématique. Selon le type de système sur lequel nous devons travailler et de sa dynamique, une méthode pourrait être plus efficace qu'une autre et ainsi il serait possible de contenir l'erreur d'approximation sous un seuil acceptable.

Aussi petit soit-il, le pas de simulation ne sera jamais nul, il y aura toujours un laps de temps entre deux pas de simulation. Or dans ce laps de temps, il n'est pas impossible qu'un évènement, comme l'ouverture ou la fermeture d'un interrupteur, provoque un changement de topologie. Si nous n'utilisons aucun algorithme pour la localisation et le traitement de ces évènements, le simulateur en tiendra compte uniquement au pas de simulation suivant et ceci influencera beaucoup les résultats de la simulation. Il est possible de voir sur la Figure 1.3 ce qu'implique un tel retard de commutation. Dans cet exemple, un évènement est déclenché à

l'instant t_p. Puisque la simulation est numérique, cet évènement sera détecté uniquement au pas de calcul t_k. Les équations à résoudre seront alors modifiées et le nouveau système sera pris en compte seulement à partir du pas de calcul t_{k+1}. Nous avons donc un retard minimum d'un pas de simulation et d'au maximum de deux pas de simulation sur l'instant réel de la discontinuité. De plus, ce changement de topologie implique invariablement un changement du système d'équations du modèle mathématique. Pour pouvoir utiliser ce nouveau modèle, nous devons avoir les conditions initiales. Ces conditions sont en fait l'état du système simulé juste après la discontinuité. En ne localisant pas cette discontinuité, il n'est pas possible d'obtenir cet état si important.

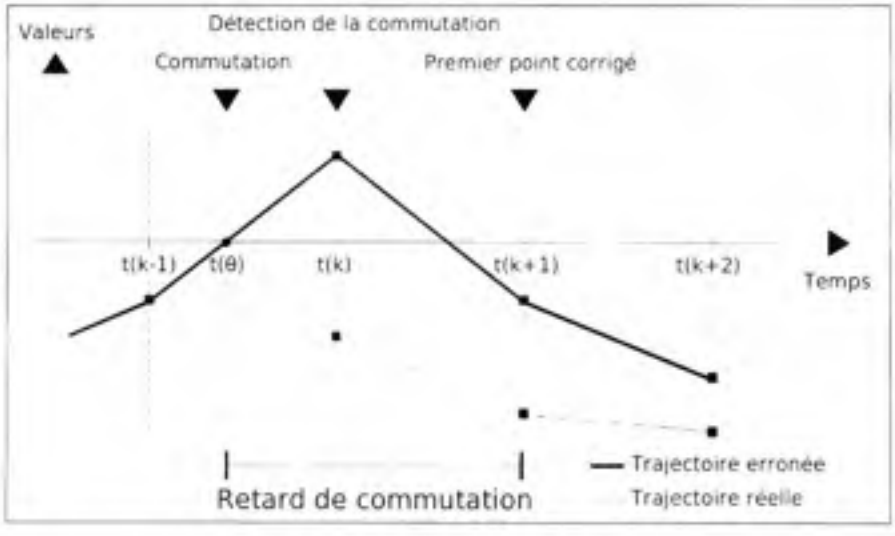

Figure 1.3 Exemple d'un retard de commutation.

Omettre de localiser les discontinuités dans une simulation est une bien mauvaise pratique. Selon Kuffel (Kuffel, 1997) cette omission peut provoquer un mauvais écoulement du flux d'énergie dû à la topologie erronée. Si nous ne localisons pas correctement cet évènement, l'énergie des composantes ne s'écoulera pas comme elle le devrait sur toute la durée du retard de commutation et des erreurs en découleront dans les résultats. Ensuite, des harmoniques non caractéristiques seront nécessairement générées. Pour toutes ces raisons, les résultats globaux de la simulation seront inévitablement erronés.

Il est donc clair que l'utilisation d'un algorithme pour détecter, localiser, franchir l'évènement (établir les conditions initiales juste après l'évènement) et resynchroniser la simulation est primordiale pour limiter les erreurs causées par les retards de commutations.

Dans un autre ordre d'idée, qu'arrive-t-il lorsque le signal simulé croise à plus d'une reprise la limite qui déclenche un évènement ? En effet, tel qu'illustré sur la Figure 1.4, il peut arriver qu'à l'intérieur d'un même pas de simulation, l'état du système varie suffisamment pour croiser à plusieurs occasions cette frontière qui déclenche des discontinuités. Dans cet exemple, entre le pas de simulation t_{k+1} et le pas de simulation t_{k+2}, l'état du système croise à deux instants différents la limite. Un algorithme simple de simulation échouerait à localiser convenablement ce cas pathologique. De façon plus précise, si le signal croise un nombre pair de fois la frontière qui déclenche une discontinuité, le simulateur continuera son travail sans même avoir détecté les évènements et les résultats seront évidemment faux.

Il est donc impératif d'être en mesure de détecter tous les évènements sans exception. En d'autres mots, si un intervalle contient un ou plusieurs évènements, l'algorithme doit être en mesure de toutes les localiser dans l'ordre précis de leurs avènements même dans le cas où nous avons un nombre pair d'évènements sur un même signal.

Suite à l'énumération de ces trois problèmes, il est clair que nous avons une influence très limitée sur le premier. Par contre, il est évident que des algorithmes spéciaux pour la simulation numérique doivent impérativement être utilisés pour minimiser les erreurs dues aux discontinuités et rendre la simulation plus fidèle à la réalité. Ces problèmes ont été abordés à plus d'une reprise dans la littérature spécialisée. Dans le chapitre suivant, différents algorithmes proposés dans le passé seront donc présentés et nous verrons comment ils répondent à ces problèmes. Dans un premier temps, les algorithmes à pas variables seront présentés et par la suite les algorithmes à pas fixe seront abordés.

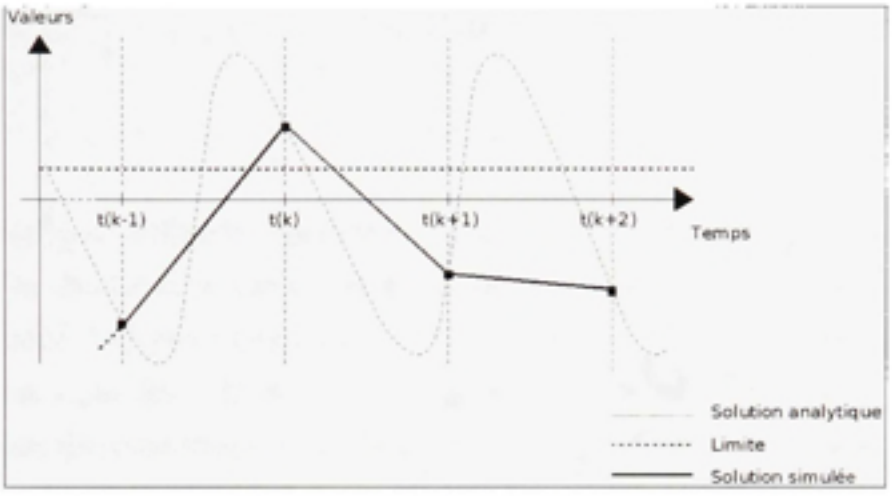

Figure 1.4 Deux évènements dans un même pas de simulation.

CHAPITRE 2

ÉTAT DE L'ART

Dans ce chapitre, différents algorithmes seront abordés afin d'établir l'état de l'art concernant la simulation en temps réel des systèmes hybrides. Deux types d'algorithmes seront présentés. Le premier type concerne la simulation à pas variable et le second concerne la simulation à pas fixe. Dans les deux cas, nous aborderons différentes méthodes en décrivant leurs fonctionnements et en expliquant comment elles effectuent une simulation en temps réel des systèmes hybrides. Il sera donc possible d'établir les points forts et les points faibles de chacun des cas et ceci servira finalement à établir les bases de ce que serait le meilleur algorithme pour simuler des systèmes hybrides en temps réel.

2.1 Les algorithmes à pas variable

Pour résoudre les difficultés reliées aux retards de commutation, une solution simple serait d'utiliser un pas de simulation variable choisi en fonction de la dynamique du système simulé. Ainsi, lorsqu'une discontinuité serait détectée, il serait possible de refaire les calculs du point actuel en utilisant un pas de simulation plus petit. Ceci permettrait d'approcher la discontinuité sans la dépasser et ainsi éviter les imprécisions induites par les retards de commutation. Osterby (Osterby, 1984) et Brankin (Brankin, 1991) ont tous les deux proposé des algorithmes reposant sur ce principe.

Dans le cas d' Osterby (Osterby, 1984), il propose un algorithme qui permet de détecter et de traiter une discontinuité lorsque nous n'avons aucune information sur la fonction qui a déclenché cette dernière. Dans l'article, il est proposé d'établir l'ordre et la grandeur de la discontinuité traitée afin d'ajuster le pas de simulation pour ainsi passer le point critique tout en gardant l'erreur sous une certaine tolérance. Cette méthode est découpée en 4 étapes distinctes:

1. Détecter la possibilité qu'une discontinuité soit présente dans un intervalle.

2. Déterminer l'ordre, la grandeur et la position de cette discontinuité.
3. Passer le point critique.
4. Continuer la simulation une fois le point critique franchi

Pour s'affranchir de la première étape dans une simulation, les auteurs proposent d'utiliser, comme indice, la différence entre le point prédit et le point corrigé d'une méthode de type prédicteur-correcteur. Un grand écart entre les deux points (donc une grande erreur d'estimation) indique qu'une discontinuité est possiblement présente dans le pas de simulation. Autrement, lorsque l'erreur d'estimation est gardée sous un seuil de tolérance acceptable, le simulateur peut assumer qu'aucune discontinuité n'est présente.

Dans le cas où l'erreur d'estimation dépasse le seuil de tolérance et que, par conséquent, le simulateur a détecté ce qu'il croit être une discontinuité, il doit alors passer à la seconde étape et tenter de localiser cette dernière. Lorsque nous avons des informations concernant la frontière ayant déclenché cet évènement, il est possible d'utiliser une interpolation pour localiser de façon approximative la position de la discontinuité. Autrement, les auteurs proposent de diviser le pas de simulation par deux et de vérifier si le nouveau point est toujours dans l'erreur. Si tel est le cas, le pas de simulation est encore divisé par deux jusqu'à ce que nous obtenions un point valide, ce dernier étant toujours antérieur au point critique. À partir de ce nouveau point, l'algorithme tente encore de s'approcher de la discontinuité en avançant la simulation d'un pas supplémentaire. Par contre, le pas de simulation est à priori divisé par deux, car il est stipulé que les chances sont très bonnes pour que la distance entre le nouveau point et le point critique soit plus petite que le pas actuel. Ces étapes sont itérées jusqu'à ce que le pas soit suffisamment petit pour évaluer la position de la discontinuité. Se faisant, les informations récoltées durant ces itérations sont utilisées pour évaluer l'ordre et la grandeur de la discontinuité. Ces données seront utilisées, par la suite, pour effectuer la troisième étape : passer le point critique. Par exemple, si l'ordre de la discontinuité est égal à deux, l'algorithme choisit alors un pas qui permettra de sauter exactement à l'emplacement de la discontinuité et celle-ci pourra alors être facilement franchie. D'autres techniques, non présentées dans le présent document, mais très bien expliquées dans Osterby (Osterby, 1984),

existent pour des ordres différents. Finalement, afin d'effectuer la quatrième étape et continuer la simulation, on enchaîne tout simplement en prenant soin de ne pas utiliser des résultats antérieurs au point critique. Puisque les techniques de résolution des EDO utilisent souvent les points antérieurs pour effecteur leurs calculs, il est important de noter qu'il se peut que, pour cette étape, une technique d'intégration différente soit utilisée afin de s'assurer qu'uniquement les résultats suivants la discontinuité ne soient pris en considération.

De son côté, Brankin (Brankin, 1991) s'attaque aux problèmes pour lesquels la fonction qui déclenche les discontinuités est connue. Cet algorithme a été mis au point dans le but de régler, entre autres, deux problèmes classiques déjà cités plus haut dans la simulation des systèmes hybrides:

1. Être en mesure de détecter tous les évènements dans un intervalle donné.
2. Être certain que la localisation de tous les évènements dans un même intervalle s'effectue en respectant l'ordre chronologique de leur déclenchement.

Pour parvenir à ce but, les auteurs utilisent, à chaque pas de simulation, les points générés par l'intégrateur numérique afin de produire une approximation polynomiale de la fonction simulée. Par la suite, à l'aide des séquences de Sturm appliquées à l'approximation, ils évaluent la présence, ou non, d'un ou plusieurs évènements dans cet intervalle. Le théorème de Sturm (Wikipedia, 2009) est tout simplement une méthode pour trouver le nombre de racines d'un polynôme dans un intervalle donné. Lorsqu'un intervalle contient plus d'un évènement, la méthode de la bissection peut être utilisée jusqu'à ce que les séquences de Sturm indiquent uniquement une discontinuité. De cette manière, le simulateur est certain d'obtenir la liste complète des évènements générés par l'approximation polynomiale et de pouvoir isoler la première, chronologiquement parlant. Il ne reste plus qu'à les traiter convenablement. Pour ce faire, les auteurs proposent plusieurs techniques qui ne seront pas résumées ici puisqu'elles ne sont d'aucun intérêt pour la compréhension de l'algorithme. Si une racine valide est trouvée, le pas de simulation doit être modifié pour faire évoluer le système jusqu'à cette dernière. Il est important de noter que le dénombrement, le classement

chronologique et la localisation des racines sont des étapes ayant un lourd coût de calcul ce qui peut rendre compliqué l'application de cette méthode dans le cadre d'une simulation temps réel.

Cette technique est très intéressante puisqu'elle permet de localiser plusieurs évènements dans un même intervalle, mais il y a certaines limites à son efficacité. Premièrement, si plusieurs évènements sont déclenchés dans un intervalle de temps très court, l'algorithme va échouer dans la localisation de ces derniers, car il est limité par la précision des calculs. Deuxièmement, si le système d'équations est redéfini, suite à une discontinuité, les informations sur l'intervalle, et par conséquent l'approximation polynomiale, ne correspondent plus au nouveau système qui en résulte. Si nous voulons nous assurer de traiter tous les évènements, il faut donc tout recalculer de nouveau avec une nouvelle approximation polynomiale. Ce processus peut être très long et coûteux. Troisièmement, le calcul de la séquence de Sturm, à chaque pas de simulation, introduit des délais considérables dus aux efforts de calcul nécessaires. Il y a donc un très grand risque que les contraintes de temps ne soient plus respectées.

2.1.1 Les problèmes avec les algorithmes à pas variable

En plus des problèmes inhérents à chaque technique décrite plus haut, le fait d'utiliser un pas de calcul variable, introduit d'autres inconvénients que nous devons considérer. Comme le stipule De Kelper (Kelper, 2002), il est très compliqué de fabriquer un circuit de conditionnement ayant un comportement linéaire sur une large plage de fréquence. De plus, le prix d'un convertisseur est directement proportionnel au taux d'échantillonnage avec lequel il est utilisé. Ensuite, dû aux calculs supplémentaires entraînés par un pas de calcul variable, il se pourrait que la simulation ne respecte plus les contraintes de temps réel imposées par le système dynamique simulé. Il est donc à notre avantage de faire la simulation avec un pas de calcul fixe afin que les contraintes soient toujours respectées, que le circuit de conditionnement soit bien calibré et que les convertisseurs soient judicieusement sélectionnés.

2.2 Les algorithmes à pas fixe

Étant établie qu'une simulation à pas fixe doit impérativement être utilisée afin de minimiser les erreurs dues aux retards de commutation, une option simple serait de diminuer de façon drastique ce pas tout en le gardant fixe. Les erreurs dues aux évènements survenus entre deux pas seraient alors tenues sous un seuil acceptable, car l'erreur introduite par un retard de commutation est d'autant plus importante que le pas de calcul est grand. Par contre, il a déjà été établi dans un chapitre précédent que nous ne pouvons diminuer indéfiniment ce pas, car nous sommes limités par la puissance des ordinateurs utilisés pour effectuer la simulation. En effet, la charge de calcul de chaque pas de simulation est la même, peu importe sa taille. De ce fait, la réduction du pas de simulation entraîne invariablement une augmentation de la charge de calcul pour une durée de simulation donnée. Nous sommes donc, encore une fois, limités par les contraintes de temps imposées par le système simulé. Il faut alors utiliser des algorithmes spéciaux qui permettront d'utiliser un pas de simulation fixe suffisamment grand pour ne pas trop surcharger le calculateur tout en permettant de localiser convenablement l'ensemble des évènements entre chacun de ces pas de calcul. Dans ce qui suit, quelques-uns de ces algorithmes à pas fixe seront présentés afin de connaître l'état actuel de l'art dans ce domaine.

En tout, cinq méthodes seront comparées dans cette section. La première, développée en 1997, par Kuffel, Kent et Irwin (Kuffel, 1997). La seconde écrite par Dinavahi, Iravine et Bonert (Dinavahi, 2001), a été publiée en 2001. La troisième a été élaborée par Do, McCallum, Giroux et De Kelper (Kelper, 2001) également en 2001. La quatrième est la méthode de De Kelper, Dessaint, Al-Haddad et Nakra (Nakra, 2002) publiée en 2002 et, finalement, la dernière méthode est celle de Strunz (Strunz, 2004) développée en 2004.

Puisque toutes ces méthodes sont souvent identiques, à quelques détails près, une présentation exhaustive pour chacune d'entre elles ne sera pas faite dans ce document. Il sera

plutôt indiqué, de quelle manière elles répondent aux besoins d'une simulation d'un système hybride tout en mettant en évidence les différences qui existent entre elles.

De façon générale, toutes ces méthodes tentent de résoudre le problème de la simulation des systèmes hybrides en respectant les 4 étapes suivantes:

1. Détection d'un évènement.
2. Localisation de cet évènement.
3. Initialisation du nouveau système d'équations (les conditions initiales).
4. Resynchronisation sur le pas de simulation initiale.

Les principales différences entre les méthodes évaluées dans cette section résident essentiellement dans les deux dernières étapes. Pour ce qui est des deux premiers points, aucun nouvel élément n'est apporté d'une méthode à l'autre.

La détection d'un évènement est l'étape qui a été le moins souvent abordée dans les méthodes proposées par les auteurs. Bien qu'ils précisent tous qu'une telle étape devait impérativement exister, ils ne la décrivent pas. Nous pouvons donc présumer que tous les algorithmes utilisent une méthode classique telle que l'observation d'un changement de signe pour effectuer cette détection. Cette méthode consiste à soustraire, à la valeur du signal simulé, la limite qui déclenche un évènement. Un changement de signe de ce résultat entre deux pas de calcul consécutif signifie invariablement que le signal simulé a croisé la limite. Or qu'arrive-t-il si le signal croise à plus d'une reprise la limite dans le même pas de calcul ? Le signe du résultat de la soustraction restera le même et les discontinuités ne seront pas prises en considération dans la simulation. Il est possible de voir cette méthode en action sur la Figure 2.1. Entre le pas t_{k-1} et le pas t_k il y a uniquement une discontinuité et la soustraction change de signe d'un pas à l'autre. Par contre, entre le pas de calcul t_{k+1} et t_{k+2}, il y a un nombre pair de discontinuités et la soustraction reste négative pour les deux pas de calculs. Bien qu'il y ait eu deux discontinuités dans le même intervalle, un simulateur qui utilise cette méthode, ne sera en mesure de localiser et traiter ni l'une ni l'autre.

Si les techniques proposées utilisent bel et bien cette soustraction pour effectuer la détection, il se pose donc un premier problème: aucune méthode présentée ici n'est en mesure de détecter un nombre pair d'évènements à l'intérieur d'un même pas de calcul. Un mécanisme tel que celui présenté par Brankin (Brankin, 1991) doit absolument être utilisé pour détecter ces cas pathologiques.

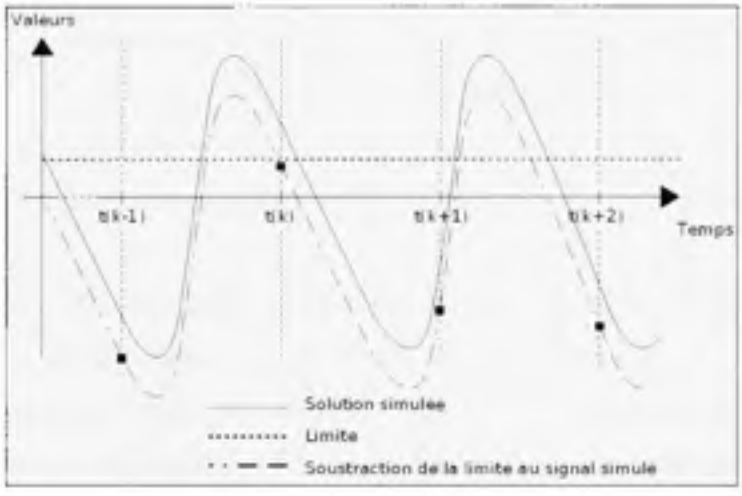

Figure 2.1 Méthode classique pour la détection d'une discontinuité.

Pour ce qui est de la localisation des évènements, toutes les méthodes utilisent la même procédure. Une fois que l'évènement a été détecté entre deux pas de simulation, une interpolation linéaire est effectuée entre le pas de simulation suivant l'évènement, situé à t_k, et le pas de simulation précédant l'évènement, situé à t_{k-1}. Pour déterminer l'instant exact t_θ auquel s'est produit l'évènement, il suffit de trouver à quel moment cette interpolation croise la limite ayant déclenché l'évènement. Nous pouvons voir ces étapes sur la Figure 2.2. Cette opération est simple et peu coûteuse en temps de calcul et c'est précisément pour ces deux raisons qu'elle est si souvent utilisée.

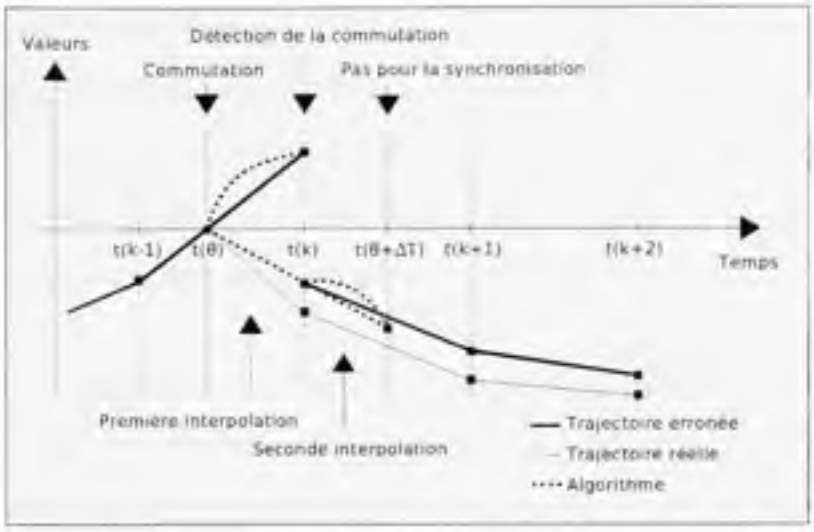

Figure 2.2 Étapes des algorithmes à pas fixe avec correction du pas t_k.

Par la suite, il est nécessaire d'établir les conditions initiales après la discontinuité. Ces conditions doivent impérativement tenir compte de l'évènement et de l'influence que ce dernier a sur le système simulé. Par exemple, dans un circuit électrique avec interrupteurs, lors d'une fermeture ou une ouverture d'un interrupteur, la topologie du système est changée. Les écoulements d'énergie sont donc nécessairement différents et les équations du système doivent tenir compte de cette subtilité. Uniquement De Kelper, Dessaint, Al-Haddad et Nakra (Nakra, 2002) proposent une méthode intéressante pour résoudre ce problème. Les auteurs proposent d'utiliser le principe de conservation d'énergie et la loi des tensions et courants de Kirchoff pour trouver l'état du système juste après une commutation. Toutes les autres méthodes supposent un écoulement d'énergie continue lors d'un évènement, mais ceci est très souvent irréaliste, voir non justifiable à cause des changements de topologie.

Toutes les techniques abordées dans ce document effectuent une synchronisation à la suite du traitement d'un évènement. Par contre, il est important de noter que deux options différentes sont généralement utilisées pour effectuer cette étape. La première, employée par Kuffel, Kent et Irwin (Kuffel, 1997), Do, McCallum, Giroux et De Kelper (Kelper, 2001) et De Kelper, Dessaint, Al-Haddad et Nakra (Nakra, 2002), utilise le nouveau système d'équations, qui tient compte de l'évènement, pour corriger le point t_i suivant l'évènement avant que la

simulation puisse calculer les points suivants (t_{k+1}, t_{k+2}, ...). Pour ce faire, les méthodes effectuent un pas de simulation supplémentaire à partir de la localisation de l'évènement à t_θ pour obtenir le point $t_{\theta+\Delta t}$ qui est situé après t_k. Par la suite, une deuxième interpolation entre ces deux points est utilisée pour déterminer l'état du système corrigé au point t_k. Cette méthode est également illustrée sur la Figure 2.2.

La seconde option préconisée par Dinavahi, Iravine et Bonert (Dinavahi, 2001) et Strunz (Strunz, 2004), utilise ce nouveau système pour calculer directement le point t_{k+1} et laisse inchangé le point t_k (qui est nécessairement dans l'erreur puisqu'il a été calculé avec le système d'équations ne tenant pas compte de l'évènement). Les auteurs justifient ce choix en évoquant le respect des contraintes de temps dans la simulation temps réel.

2.2.1 Les problèmes avec la simulation à pas fixe

À la lumière de ces informations, il est possible de tirer quelques conclusions importantes concernant ces méthodes:

1. Pour la détection des évènements, il faut impérativement élaborer un algorithme de détection plus robuste que la simple observation du changement de signe dans une soustraction. Il y a plusieurs cas pathologiques qui rendent les techniques discutées dans cette section complètement impuissantes.
2. L'utilisation d'une interpolation entre les deux points de simulation semble une technique de localisation efficace et rapide qui est acceptée par la majorité des auteurs.
3. Les conditions initiales sont souvent laissées de côté, mais une technique existe qui permet de les évaluer correctement. Une méthode de simulation robuste et précise devrait utiliser cette technique.
4. Il existe un dilemme au niveau de la synchronisation d'une simulation: est-ce que le point t_k, suivant l'évènement doit ou ne doit pas être corrigé après le traitement de cette discontinuité. Dans le premier cas, il y a perte de temps, mais il y a un gain au niveau de

la précision et dans le cas suivant, c'est l'inverse, il n'y a aucune perte de temps supplémentaire, mais il y a une perte au niveau de la précision.

Finalement, toutes ces méthodes reposent sur le fait que le point t_i qui suit l'évènement doit impérativement être connu avant même de pouvoir détecter une discontinuité. Si ce point nécessite d'être recalculé suite au traitement, il y a une perte de temps non négligeable à ce niveau. Il pourrait donc être intéressant d'élaborer une méthode qui permettrait d'anticiper un évènement avant de calculer le point t_i. Ceci règlerait assurément le dilemme au point 4 de la liste précédente. Il ne serait plus question de le corriger ou non puisque ce dernier ne serait pas encore calculé. De plus, des économies de temps seraient effectuées ce qui laisserait plus de temps pour les autres étapes. Ceci est d'ailleurs un des points les plus importants justifiant l'élaboration d'un nouvel algorithme.

2.3 Proposition d'un nouvel algorithme

Dans le but de respecter les contraintes de temps imposées par une simulation en temps réel, il serait intéressant de minimiser les pertes de temps inhérentes aux algorithmes de simulation à pas fixe actuel. Un nouvel algorithme qui serait en mesure de détecter à l'avance si un évènement doit se produire dans l'intervalle de temps qui suit pourrait donc être développé. En d'autres mots, cette nouvelle méthode devrait être en mesure, à l'aide du point actuel et des points précédents (t_{i-1}, t_{i-2}, ...), de détecter si un ou plusieurs évènements devraient se produire dans l'intervalle $[t_{i-1}, t_i]$. De cette manière, nous serions en mesure de faire l'économie de plusieurs cycles de calcul qui étaient, auparavant, effectués inutilement.

Pour être efficace et utilisable, ce nouvel algorithme devra comporter certaines caractéristiques. La liste suivante, extraite de Esposito (Esposito, 2002), présente les plus importantes.

1. La méthode doit garantir de détecter tous les évènements sans exception.

2. Si plusieurs évènements existent dans un même intervalle, la méthode doit garantir de détecter le premier chronologiquement parlant.
3. Une fois qu'un évènement est détecté, la méthode doit être en mesure de localiser le plus précisément possible le temps t_n auquel s'est produit l'évènement.
4. La méthode doit être le plus efficace possible en termes de temps de calcul afin que la simulation respecte toutes les contraintes de temps imposées par le système simulé.

Dans les chapitres suivants, un algorithme basé sur la théorie de contrôle des systèmes non linéaire sera présenté et analysé. Ce nouvel algorithme a le potentiel de contourner les problèmes cités dans les chapitres précédents, tout en respectant la liste des caractéristiques énumérée ci-dessus. Nous commencerons donc par poser les bases théoriques utilisées par l'algorithme. Par la suite, les étapes générales de cette nouvelle méthode seront développées et nous enchaînerons en appliquant ces étapes à différentes méthodes de résolution d'équations différentielles. Finalement, des tests fonctionnels et en temps réel seront effectués pour prouver son efficacité.

CHAPITRE 3

BASE THÉORIQUE DE L'ALGORITHME PROPOSÉ

Dans le but de minimiser les difficultés reliées aux impacts dans la simulation de système robotique, les auteurs de (Esposito, Kumar et Pappas, 2001) ont développé une méthode de détection des collisions basée sur la théorie du contrôle des systèmes non linéaires. Cette méthode considère la simulation dans son ensemble comme étant un système qu'il faut contrôler. L'entrée de ce système est le pas de calcul, la sortie est l'expression mathématique de la frontière qui déclenche une collision et la dynamique du système est l'équation aux différences de la méthode de résolution des EDO. Ainsi, les auteurs proposent une méthode pour trouver une loi de contrôle qui fera varier le pas de simulation afin que le système s'arrête à la frontière de la collision sans jamais la dépasser. Cette loi de contrôle est trouvée à l'aide de la linéarisation par rétroaction (feedback linearization). Étant donné que l'équation mathématique qui décrit la frontière est exprimée en fonction des états du système et que cette dernière peut être autant linéaire que non linéaire, l'approche du contrôle des systèmes non linéaires est donc une solution intéressante et viable pour résoudre le problème décrit dans l'article.

Dans sa forme originale, cette méthode de détection est applicable à une simulation à pas variable. Elle consiste à anticiper une collision dans la simulation pour ensuite moduler le pas de calcul en fonction de la distance entre l'état du système et ladite frontière générant l'impact. Or dans le domaine d'intérêt de ce document soit la simulation temps réel des systèmes hybrides, les méthodes à pas variable ne sont pas optimales et nous avons tout intérêt à utiliser un pas fixe pour effectuer la simulation. Dans ce chapitre, il sera donc proposé de modifier l'algorithme présenté de l'article afin de l'appliquer à une simulation temps réel à pas fixe des systèmes électriques exhibant des discontinuités.

En résumé, le but du nouvel algorithme ne sera pas de moduler le pas de calcul de façon à s'approcher lentement d'une discontinuité, mais bien de localiser cette dernière à l'intérieur

d'un pas de calcul fixe et c'est justement à ce niveau que se trouve l'originalité du travail présenté dans ce document.

Pour ce faire, différents concepts seront présentés. En premier lieu, une définition de la fonction gardienne en tant que frontière sera donnée. Par la suite, quelques explications sur la linéarisation par rétroaction ainsi que sur les équations aux différences seront données et, finalement, les équations du nouvel algorithme seront développées en utilisant ces mêmes concepts.

3.1 Fonction gardienne

À plusieurs reprises dans l'article de référence le terme « guard function », qui peut être traduit par « fonction gardienne », est utilisé pour décrire la frontière générant une collision. Dans le cadre de ce travail, la fonction gardienne $g(x)$ est la frontière qui génère une discontinuité dans la simulation et déclenche un changement du modèle mathématique utilisé pour effectuer la simulation. Dans une forme très simple, cette fonction peut être exprimée comme suit :

$$g(x) = x - b \tag{3.1}$$

Dans cet exemple, b est la limite qui déclenche un évènement. Par exemple, cette limite peut être la tension maximale dans un réseau électrique avant qu'une protection se déclenche ou le courant minimal pour lequel une diode reste en conduction. De son côté, x est la quantité simulée. La tension aux bornes d'une résistance ou le courant dans une diode sont deux exemples d'une quantité pouvant être simulées. Une discontinuité survient lorsque la fonction $g(x)$ change de signe et c'est ce moment précis qui doit être localisé dans la simulation temps réel des systèmes hybrides.

Bien que dans les explications qui précèdent, la fonction gardienne est donnée dans une forme linéaire, celle-ci peut tout aussi bien être non linéaire. Il est donc important de

développer une méthode qui sera applicable dans les deux cas et c'est exactement pour cette raison que l'algorithme s'inspirera des mathématiques utilisés dans une technique issue du domaine du contrôle des systèmes non linéaires : la linéarisation par rétroaction.

3.2 Linéarisation par rétroaction

La linéarisation par rétroaction est une technique utilisée pour contrôler un système non linéaire de la forme donnée par l'équation suivante :

$$\begin{aligned} \dot{x} &= a(x) + b(x)\mu \\ y &= c(x) \end{aligned} \tag{3.2}$$

Avec x étant le vecteur d'états du système, μ le vecteur d'entrées et y le vecteur de sorties.

La première étape de cette méthode consiste à dériver un certain nombre de fois le vecteur des sorties y du système de manière à obtenir une équation qui est fonction du vecteur d'entrées μ. Le nombre de dérivées qui est nécessaire d'effectuer au cours de cette étape est égal au degré relatif du système. Nous obtenons alors ce qui suit:

$$\dot{y} = \frac{dc(x)}{dt} = \frac{\partial c(x)}{\partial x}\frac{dx}{dt} = \frac{\partial c(x)}{\partial x}\dot{x} \tag{3.3}$$

$$\frac{\partial c(x)}{\partial x}\dot{x} = \frac{\partial c(x)}{\partial x}a(x) + \frac{\partial c(x)}{\partial x}b(x)\mu \tag{3.4}$$

$$\frac{\partial c(x)}{\partial x}\dot{x} = L_a c(x) + L_b c(x)\mu \tag{3.5}$$

Dans l'équation 3.5, $L_a c(x)$ et $L_b c(x)\mu$ sont des dérivées de Lie. Plus précisément, cette dernière est la dérivée d'une fonction le long de la trajectoire d'un système du type $\dot{x} = a(x)$ et d'une façon générale, $L_f j(x)$ est donné par:

$$L_f j(x) = \frac{dj(x)}{dx} i(x) \tag{3.6}$$

Maintenant, dans l'équation 3.5, si nous effectuons le changement suivant:

$$\mu = \frac{1}{L_g c(x)} \left[- L_f c(x) + v \right] \tag{3.7}$$

Nous obtenons alors le système linéaire donné par l'équation 3.8, que nous pouvons contrôler de façon linéaire.

$$\dot{y} = v \tag{3.8}$$

Ici, uniquement les principales étapes de la méthode ont été décrites. Les étapes complètes de cette méthode sont décrites plus en détail dans (Marquez, 2003).

3.3 L'équation aux différences

Contrairement à une méthode analytique, par laquelle nous obtenons la solution pour toutes valeurs du temps, lorsque nous utilisons une méthode numérique pour résoudre une équation différentielle, nous obtenons uniquement une série de points, sous forme de vecteur, qui forme la solution au problème. Pour produire ces points, la grande majorité des méthodes reposent sur le même principe. Elles utilisent les itérations successives d'une équation aux différences afin de générer le vecteur de points constituant la solution au problème. De façon générale, cette équation aux différences se pose comme suit:

$$x_{k+1} = x_k + h_{k+1} \Phi\left(x_k, x_{k-1}, \ldots, \mu_k, \mu_{k-1}, \ldots \right) \tag{3.9}$$

Le terme $\Phi(x_{i_k}, x_{i_{k-1}}, \ldots, \mu_{i_k}, \mu_{i_{k-1}}, \ldots)$ étant une fonction quelconque qui change selon la méthode de résolution numérique utilisée. Au début de ce chapitre, lorsque la dynamique du système a été posée comme étant l'équation aux différences de la méthode de résolution des EDO, il était en fait question de cette fonction qui est nommée $\bar{f}_\beta$ dans (Esposito, Kumar et Pappas, 2001). À titre d'exemple, pour les méthodes Adams explicites, tel que proposé par (Esposito, Kumar et Pappas, 2001), nous avons:

$$\Phi(\ldots) = \sum_{i=1}^{m} \beta_i f_{k-i+1} = \bar{f}_\beta \tag{3.10}$$

Dans le cas d'une simulation à pas variable, les coefficients β_i ne sont pas constants, ils sont fonction du pas de simulation et nous verrons plus loin qu'il est possible de tirer profit de cette particularité afin d'évaluer la présence d'une discontinuité dans un intervalle.

3.4 Développement d'une première approche du nouvel algorithme

Dans un système simulé, un évènement se produit dès que la fonction gardienne $g(x(t), \mu(t))$ croise une limite préalablement établie. Or cet instant peut se produire à tout moment y compris entre deux pas de simulation. Tel que proposé par les auteurs de (Esposito, Kumar et Pappas, 2001), une première idée pourrait être d'utiliser le pas de simulation h, la fonction gardienne $g(x(t), \mu(t))$, l'équation aux différences $\bar{f}_\beta$ ainsi que la théorie de la linéarisation par rétroaction pour localiser efficacement cette discontinuité quelque soit sa position.

De façon générale, lorsque nous travaillons dans le domaine discret, la variation des variables d'états d'un système est donnée par l'équation suivante. Cette forme itérative découle directement de la série de Taylor :

$$x(t_k + h_{k+1}) = x(t_k) + h_{k+1} \bar{f}_\beta \tag{3.11}$$

$$x_{t_{k+1}} = x_{t_k} + h_{k+1}\bar{f}_\beta \tag{3.12}$$

Par conséquent, la dynamique de la fonction gardienne est exprimée par:

$$g\left(x_{t_{k+1}}, \mu_{t_{k+1}}\right) = g\left(x_{t_k} + h_{k+1}\bar{f}_\beta\right) \tag{3.13}$$

Avec cette équation, il peut être intéressant de se demander quel pas de calcul h_{k+1} ferait tendre la fonction gardienne vers 0. À l'instar du changement de variable effectué pour la linéarisation par rétroaction à l'équation 3.7, nous pouvons poser h_{k+1} comme suit:

$$h_{k+1} = \frac{-x_{t_k} + g^{-1}\left(\gamma g\left(x_{t_k}, \mu_{t_k}\right)\right)}{\bar{f}_\beta} \tag{3.14}$$

Dans cette équation γ est une constante positive choisie par l'utilisateur. Dans l'algorithme original présenté dans (Esposito, Kumar et Pappas, 2001), elle est l'équivalent d'un gain qui aura comme effet de ralentir la simulation en choisissant des pas de calcul de plus en plus petit. Plus petit sera ce gain, plus rapidement la simulation va converger vers la frontière de la collision et vice versa.

Maintenant, en appliquant l'équation 3.14 à l'équation 3.13 nous obtenons:

$$g\left(x_{t_{k+1}}, \mu_{t_{k+1}}\right) = \gamma g\left(x_{t_k}, \mu_{t_k}\right) \tag{3.15}$$

qui tend vers 0 si $0 < \gamma < 1$. Le problème avec ce changement de variable est que nous n'avons pas d'expression pour $g^{-1}\left(x_{t_k}, \mu_{t_k}\right)$. Il nous est donc impossible d'utiliser cette équation. Pour contourner cette impasse, nous devons alors nous tourner vers quelque chose que nous connaissons mieux soit la série de Taylor.

D'une façon générale, la série de Taylor d'une fonction quelconque $q(t)$ autour d'un point x_0, en gardant uniquement deux termes, est donnée comme suit:

$$q(x) = q(x_0) + \frac{dq(x_0)}{dt}(x - x_0) + \ldots \tag{3.16}$$

Or, en effectuant les affectations suivantes :

$$q(x) = g(x_{t_{k+1}}, \mu_{t_{k+1}}) \tag{3.17}$$

$$q(x_0) = g(x_{t_k}, \mu_{t_k}) \tag{3.18}$$

Nous obtenons pour la fonction gardienne ce qui suit (sachant que $t_{k+1} = t_k + h_{k+1}$):

$$g(x_{t_{k+1}}, \mu_{t_{k+1}}) = g(x_{t_k}, \mu_{t_k}) + \frac{dg(x_{t_k}, \mu_{t_k})}{dt}(t_k + h_{k+1} - t_k) + \ldots \tag{3.19}$$

$$g(x_{t_{k+1}}, \mu_{t_{k+1}}) = g(x_{t_k}, \mu_{t_k}) + \dot{g}(x_{t_k}, \mu_{t_k})h_{k+1} + \ldots \tag{3.20}$$

Puisque nous avons stipulé que la fonction gardienne était fonction du signal simulé x_{t_k} et de l'entrée μ_{t_k}, que la dynamique du système non linéaire $\dot{x}$ était l'équation aux différences $\bar{f}_\beta$ ($\dot{x} = \bar{f}_\beta$) et connaissant la loi de la dérivée en chaîne, nous trouvons:

$$\frac{dg(x_{t_k}, \mu_{t_k})}{dt} = \frac{\partial g}{\partial x}\frac{dx}{dt} + \frac{\partial g}{\partial \mu}\frac{d\mu}{dt} \tag{3.21}$$

$$\frac{\partial g}{\partial x}\frac{dx}{dt} + \frac{\partial g}{\partial \mu}\frac{d\mu}{dt} = \frac{\partial g}{\partial x}\dot{x} + \frac{\partial g}{\partial \mu}\dot{\mu} \tag{3.22}$$

$$\frac{\partial g}{\partial x}\bar{f}_\beta + \frac{\partial g}{\partial \mu}\dot{\mu} = L_{\bar{f}_\beta} g(x_{t_k}, \mu_{t_k}) + L_\mu g(x_{t_k}, \mu_{t_k}) \tag{3.23}$$

Et ainsi de suite, pour les termes dérivés d'ordre supérieure.

Notons aussi que l'utilisation de la notion des dérivées de Lie n'est pas essentielle pour la méthode. Elle est utilisée ici uniquement pour maintenir la cohérence avec la méthode de linéarisation par rétroaction de la théorie du contrôle non-linéaire décrite à la section 3.2.

Pour simplifier la démonstration qui suit, si nous conservons uniquement les deux premiers termes de la série de Taylor, nous obtenons finalement:

$$g(x_{t_{k+1}}, \mu_{t_{k+1}}) = g(x_{t_k}, \mu_{t_k}) + L_{f_\mu} g(x_{t_k}, \mu_{t_k}) h_{k+1} + L_\mu g(x_{t_k}, \mu_{t_k}) h_{k+1} \tag{3.24}$$

Rappelons que nous avons posé la sortie du système comme étant la fonction gardienne $g(x(t), \mu(t))$ et que l'entrée est le pas de simulation soit h_{k+1}. Nous pouvons donc, encore une fois à l'instar de l'équation 3.7, effectuer le changement de variable suivant:

$$h_{k+1} = \frac{(\gamma - 1) g(x_{t_k}, \mu_{t_k})}{L_{f_\mu} g(x_{t_k}, \mu_{t_k}) + L_\mu g(x_{t_k}, \mu_{t_k})} \tag{3.25}$$

Pour obtenir:

$$g(x_{t_{k+1}}, \mu_{t_{k+1}}) = \gamma g(x_{t_k}, \mu_{t_k}) \tag{3.26}$$

Dans leur article, les auteurs de (Esposito, Kumar et Pappas, 2001) proposent de choisir le gain γ entre 0.05 et 0.5 afin d'approcher la discontinuité de façon exponentielle. Or dans le cas qui nous intéresse, nous ne désirons pas faire varier le pas de simulation afin d'approcher tranquillement une discontinuité. Ce que nous exigeons, dans le cas où une discontinuité est présente, est de sauter directement à son emplacement dans le temps. Dans ce cas, γ doit être égal à 0 afin d'obtenir le pas de simulation h_{k+1} qui ferait évoluer le système directement sur la discontinuité.

Il a déjà été expliqué plutôt que, pour une simulation à pas variable, $\bar{f}_\beta$ est fonction de h_{k+1}. Afin de trouver le pas de simulation h_{k+1} qui serait le plus approprié pour nous approcher d'une discontinuité, il faut donc former un polynôme en h_{k+1} avec l'équation suivante et ensuite trouver ses racines.

$$h_{k+1} = \frac{(\gamma - 1)g(x_{i_c}, \mu_{i_c})}{L_{\dot{x}_\beta(t_{k+1})}g(x_{i_c}, \mu_{i_c}) + L_{\mu(t_{k+1})}g(x_{i_c}, \mu_{i_c})} \tag{3.27}$$

Une fois les racines trouvées, il est nécessaire d'éliminer les résultats incongrus tels que les racines négatives ou les racines complexes, car ces dernières n'ont aucun sens physique dans une simulation. Le choix le plus approprié est effectué en ordonnant dans le temps les racines restantes et en gardant celle qui se produit la première. Si nous obtenons un $h_{k+1} < h$ (h étant le pas de calcul utilisé pour effectuer la simulation à pas fixe) nous saurons alors qu'une discontinuité est présente dans l'intervalle et nous aurons une évaluation suffisamment précise de son emplacement dans le temps. Autrement, lorsque $h_{k+1} > h$, nous pourrons tout simplement continuer la simulation sans avoir à traiter une discontinuité.

Cette approche, tel que proposé dans (Esposito, Kumar et Pappas, 2001), semble très intéressante. Par contre, un seul problème majeur persiste. Dans cette forme, l'algorithme de localisation est dépendant de la méthode numérique de résolution des EDO utilisée pour effectuer la simulation. En effet, pour chaque méthode nous avons un $\bar{f}_\beta$ différent donc pour chaque méthode, nous avons un polynôme différent. Ceci est une limitation très importante qu'il serait intéressant de régler.

3.5 Développement d'une seconde approche du nouvel algorithme

Jusqu'à présent, pour obtenir une expression de la fonction gardienne, nous avons utilisé la forme discrétisée des variables d'état qui est fonction de la méthode de résolution utilisée pour effectuer la simulation. Or pour résoudre le problème de taille cité dans le paragraphe

précédent, il pourrait être intéressant d'exprimer la fonction gardienne non plus par la forme discrète des variables d'états mais bien par la forme continue et par la suite discrétiser la fonction gardienne elle-même $g(x_{t_k}, \mu_{t_k})$ avec sa propre méthode de résolution. La méthode de résolution alors utilisée peut être complètement indépendante de celle utilisé pour effectuer la simulation. Pour ce faire, au lieu d'utiliser l'équation aux différences $\Phi(...)$ afin d'évaluer les variables d'états du système simulé $x(t_k)$, il faut l'appliquer, de manière identique, à la fonction gardienne comme suit :

$$g(x_{t_{k+1}}, \mu_{t_{k+1}}) = g(x_{t_k}, \mu_{t_k}) + h_{k+1}\Phi(x_{t_k}, x_{t_{k-1}}, ... \mu_{t_k}, \mu_{t_{k-1}}, ...) \tag{3.28}$$

Posons $\Phi(...) = \overline{g}_\beta$, qui est fonction de méthode de résolution utilisée pour résoudre l'équation 3.28, et essayons de trouver le pas de calcul qui ferait tendre la fonction gardienne $g(x_{t_{k+1}}, \mu_{t_{k+1}})$ vers 0. Ce dernier est trouvé en effectuant, à l'instar de l'équation 3.7, le changement de variable suivant:

$$h_{k+1} = \frac{(\gamma - 1)g(x_{t_k}, \mu_{t_k})}{\overline{g}_\beta} \tag{3.29}$$

Nous obtenons alors :

$$g(x_{t_{k+1}}, \mu_{t_{k+1}}) = \gamma g(x_{t_k}, \mu_{t_k}) \tag{3.30}$$

Qui tendra vers 0 si $0 < \gamma < 1$.

Sachant que $\overline{g}_\beta$ est fonction de h_{k+1} ($\overline{g}_\beta \rightarrow \overline{g}_\beta(h_{k+1})$), afin de trouver le pas de simulation h_{k+1} qui serait le plus approprié pour nous approcher d'une discontinuité, il suffit de former un polynôme en h_{k+1} avec l'équation suivante et de trouver ses racines.

$$h_{i+1} = \frac{(\gamma - 1)g(x_{t_i}, \mu_{t_i})}{\overline{g}_{\beta}(h_{i+1})} \tag{3.31}$$

Une fois les racines trouvées, il faut effectuer les mêmes étapes que précédemment à savoir :

1. Éliminer les résultats incongrus tels que les racines négatives ou les racines complexes.
2. Classer chronologiquement les racines et effectuer le choix le plus approprié de racine afin d'évaluer la présence ou non d'une discontinuité.

Ainsi, en utilisant les mêmes principes proposés dans (Esposito, Kumar et Pappas, 2001), il est possible d'effectuer la localisation des discontinuités de façon complètement indépendante de la méthode de résolution des ODE utilisée dans la simulation du système. Il reste maintenant à déterminer si cet algorithme est rentable au point de vue du temps de calcul. En effet, faire la recherche de racines dans un polynôme au degré élevé peut être une étape longue et coûteuse. Le nombre de cycles nécessaires pour effectuer la détection est proportionnel à la complexité de $\overline{g}_{\beta}(h_{i+1})$. Il est donc important, à ce niveau, de nous donner une définition exacte de cette fonction et d'en évaluer la complexité. Le prochain chapitre traitera de ce sujet en utilisant différentes méthodes de résolution numérique et tentera de conclure sur la meilleure option.

CHAPITRE 4

APPLICATION DE L'ALGORITHME AUX MÉTHODES NUMÉRIQUES DE RÉSOLUTION DES ÉQUATIONS DIFFÉRENTIELLES

Dans ce chapitre seront décrites certaines méthodes numériques pour la résolution d'équations différentielles couramment utilisées dans les simulations. De plus, il sera démontré comment appliquer ces méthodes à la fonction $\overline{g}_\beta(h_{k+1})$ décrite dans le chapitre précédent. Par contre, avant d'entrer dans le vif du sujet, le type de méthode de résolution des EDO sur lequel l'algorithme est applicable sera établi. Une fois ce sujet réglé, la méthode d'Euler simple sera analysée. Par la suite, les différents ordres de la famille Adams-Bashforth seront couverts et, en derniers lieux, les méthodes Runge-Kutta seront abordées. Pour chacune de ces méthodes, des détails qui permettront de définir $\overline{g}_\beta$ seront donnés et, lorsque nécessaire, le polynôme pour lequel nous devrons chercher les racines sera formé. En développant ainsi les détails de chaque méthode, nous aurons une idée de l'ampleur du travail à effectuer à chaque pas de simulation afin de localiser les discontinuités et nous serons en mesure de savoir si l'algorithme est applicable ou non dans une simulation temps réel.

4.1 Les méthodes explicites et implicites

En ce qui concerne les équations différentielles, deux familles de méthodes existent pour les résoudre. Il y a les méthodes dites explicites et les méthodes dites implicites. Dans le premier cas, l'état du système au temps t_{k+1} est évalué uniquement avec l'historique de ce dernier soit les états et les entrés aux temps précédents ($x_{t_k}, x_{t_{k-1}}, \ldots, \mu_{t_k}, \mu_{t_{k-1}}, \ldots$). Dans le second cas, l'état du système au temps t_{k+1} est évalué à l'aide de l'historique du système et de certaines informations aux temps t_{k+1} comme par exemple les entrées ($\mu_{t_{k+1}}$). Le choix entre ces deux familles influence énormément la stabilité d'une simulation. En effet, les méthodes explicites sont inconditionnellement stables, ce qui signifie que ces méthodes sont

stables, peu importe le pas de calcul choisi, ce qui n'est pas le cas pour les méthodes implicites. Dans ce document, il a été décidé de traiter uniquement les méthodes explicites.

4.2 La méthode d'Euler

La méthode d'Euler est possiblement la plus simple de toutes les techniques. Elle permet d'obtenir des résultats acceptables avec un effort de calcul très faible, mais la précision des données reste très limitée. Selon (Fortin, 2001) c'est principalement pour cette raison que cette méthode n'est pas souvent utilisée dans des cas pratiques. Elle a été élaborée en gardant uniquement les deux premiers termes de la série de Taylor autour d'un point g_{t_k}

$$g(x_{t_{k+1}}, \mu_{t_{k+1}}) = g(x_{t_k}, \mu_{t_k}) + \dot{g}(x_{t_k}, \mu_{t_k}) h_{k+1} \tag{4.1}$$

Dans le cas de Euler simple, la méthode est dite explicite, car le point $g(x_{t_{k+1}}, \mu_{t_{k+1}})$ est exprimé uniquement en fonction des points précédents et sa forme itérative est donnée par:

$$g(x_{t_{k+1}}, \mu_{t_{k+1}}) = g_{t_{k+1}} = g_{t_k} + \dot{g}_{t_k} h_{k+1} \tag{4.2}$$

Dans ce cas, $\bar{g}_\beta$ est égal à :

$$\bar{g}_\beta = \dot{g}_{t_k} \tag{4.3}$$

En incluant cette expression de $\bar{g}_\beta$ dans l'équation 3.31 et en considérant que $\gamma = 0$ nous obtenons :

$$h_{k+1} = \frac{(\gamma - 1) g(x_{t_k}, \mu_{t_k})}{\bar{g}_\beta(h_{k+1})} \tag{4.4}$$

$$h_{i+1} = -\frac{g_{i_1}}{\dot{g}_{i_1}} \tag{4.5}$$

Pour cette méthode, il n'y a même pas de polynôme à former donc pas de racine à trouver. Le résultat pour h_{i+1} est directement interprétable. Si h_{i+1} est plus petit que le pas de simulation, nous sommes en présence d'une discontinuité.

Un point intéressant à noter est que le nouvel algorithme, exprimé à l'aide de la méthode d'Euler, nous ramène à la définition de la méthode Newton-Raphson. Cette méthode est un algorithme utilisé pour trouver des approximations du zéro d'une fonction, ce qui peut être équivalent à la recherche de l'instant d'une discontinuité à l'intérieur d'un pas de calcul d'une simulation. Le problème avec cette méthode est qu'elle ne permet pas de trouver plus d'une seule discontinuité, puisque le polynôme utilisé pour effectuer la recherche est de degré 1. De plus, le pas de simulation est tellement simple à effectuer que la complexité du nouvel algorithme reste proportionnelle au simple fait de localiser la discontinuité par interpolation. Finalement, la méthode Newton-Raphson suppose un traitement itératif afin d'atteindre la précision désirée; à défaut de quoi, elle n'est guère plus précise qu'une simple interpolation et souvent moins précise. Ainsi, ni l'utilisation de la méthode d'Euler, avec l'algorithme proposé, ni la méthode Newton-Raphson ne nous permettent d'atteindre les buts fixés pour la nouvelle méthode de détection.

4.3 Les Méthodes Adams-Bashforth

Étant donné qu'avec la méthode d'Euler et, par le fait même, la méthode Newton-Raphson nous ne pouvons trouver qu'une seule racine donc qu'une seule discontinuité, elles ne peuvent convenir au nouvel algorithme. Il faut donc chercher une alternative qui donnera la possibilité d'obtenir un polynôme de degré plus élevé et ainsi localiser plus d'une discontinuité dans un même pas de simulation. C'est pour cette raison que les méthodes Adams-Bashforth ont été considérées. Le polynôme ainsi obtenu n'a plus aucun lien avec

celui de la méthode Newton-Raphson et permet de trouver un nombre de discontinuités égal à l'ordre de la méthode Adams-Bashforth utilisé.

Pour présenter les méthodes de la famille Adams-Bashforth, la série de Taylor autour d'un point $g(x_{t_i}, \mu_{t_i})$ sera une fois de plus utilisée. Par contre, cette fois, plus d'un terme seront gardés. Une fois la série posée, il sera alors possible de développer les équations de ces méthodes en effectuant la substitution des dérivées dans la série.

La série de Taylor autour du point $g(x_{t_i}, \mu_{t_i})$, en gardant 4 termes, est donc la suivante:

$$g(x_{t_{i+1}}, \mu_{t_{i+1}}) = g(x_{t_i}, \mu_{t_i}) + h_{i+1}\dot{g}(x_{t_i}, \mu_{t_i}) + \frac{h_{i+1}^2}{2!}\ddot{g}(x_{t_i}, \mu_{t_i}) + \frac{h_{i+1}^3}{3!}\dddot{g}(x_{t_i}, \mu_{t_i}) + \ldots \tag{4.6}$$

Que nous pouvons écrire de façon plus succincte:

$$g_{t_{i+1}} = g_{t_i} + h_{i+1}\dot{g}_{t_i} + \frac{h_{i+1}^2}{2!}\ddot{g}_{t_i} + \frac{h_{i+1}^3}{3!}\dddot{g}_{t_i} + \ldots \tag{4.7}$$

Or, tel que le démontre (Rao, 2002), ces dérivées peuvent être approximées à l'aide des formules des différences finies (différences arrière). Pour les preuves de ces substitutions, le lecteur est invité à consulter la référence. Pour l'instant, voici la substitution pour les dérivées d'ordre 2 et 3:

$$\ddot{g}_{t_i} = \frac{\dot{g}_{t_i} - \dot{g}_{t_{i-1}}}{h_{i-1}} \tag{4.8}$$

$$\dddot{g}_{t_i} = \frac{\dot{g}_{t_i} - 2\dot{g}_{t_{i-1}} + \dot{g}_{t_{i-2}}}{h_{i-1}^2} \tag{4.9}$$

Pour simplifier la démonstration qui suit, appliquons-la à la méthode d'Adams-Bashforth du second ordre, c'est-à-dire en ne conservant que les 3 premiers termes de l'équation 4.7 et effectuons la substitution pour $\ddot{g}_{t_k}$. Le résultat sera alors:

$$g_{t_{k+1}} = g_{t_k} + h_{k+1}\dot{g}_{t_k} + \frac{h_{k+1}^2}{2!}\ddot{g}_{t_k} = g_{t_k} + h_{k+1}\dot{g}_{t_k} + \frac{h_{k+1}^2}{2!}\left(\frac{\dot{g}_{t_k} - \dot{g}_{t_{k-1}}}{h_{k-1}}\right) \tag{4.10}$$

$$g_{t_{k+1}} = g_{t_k} + h_{k+1}\left[\dot{g}_{t_k} + \frac{h_{k+1}}{2}\left(\frac{\dot{g}_{t_k} - \dot{g}_{t_{k-1}}}{h_{k-1}}\right)\right] \tag{4.11}$$

Dans ce cas, $\overline{g}_\beta$ est donc égal à :

$$\overline{g}_\beta = \dot{g}_{t_k} + \frac{h_{k+1}}{2}\left(\frac{\dot{g}_{t_k} - \dot{g}_{t_{k-1}}}{h_{k-1}}\right) \tag{4.12}$$

En substituant cette expression $\overline{g}_\beta$ dans l'équation 3.31 (avec $\gamma = 0$) nous obtenons :

$$h_{k+1} = \frac{(\gamma - 1)g_{t_k}}{\overline{g}_\beta(h_{k+1})} \tag{4.13}$$

$$h_{k+1} = -\frac{g_{t_k}}{\dot{g}_{t_k} + \frac{h_{k+1}}{2}\left(\frac{\dot{g}_{t_k} - \dot{g}_{t_{k-1}}}{h_{k-1}}\right)} \tag{4.14}$$

Nous pouvons alors former le polynôme suivant :

$$ah_{k+1}^2 + bh_{k+1} + c = 0 \tag{4.15}$$

Avec ($h_{k-1} = h_k$) il faut noter que :

$$a = \frac{\dot{g}_{t_i} - \dot{g}_{t_{i-1}}}{2h_i} \tag{4.16}$$

$$b = \dot{g}_{t_i} \tag{4.17}$$

$$c = g_{t_i} \tag{4.18}$$

Il est aisé de trouver les racines de ce polynôme du second degré avec la méthode de complétion du carré. Cette méthode utilise un nombre limité d'opérations mathématiques de base et puisque nous avons deux racines, si deux discontinuités se produisent dans un même intervalle, nous pourrons être en mesure de les détecter.

Il en va de même pour les ordres supérieurs de cette famille. Il suffit de faire la substitution des dérivées et ensuite rassembler les termes ensemble pour former le polynôme. Afin de ne pas alourdir le texte, ces démonstrations pour les ordres 3, 4 et 5 ont été placées en annexe. Le lecteur intéressé pourra donc parcourir l'annexe « Démonstrations pour les méthodes Adams-Bashforth d'ordre supérieur » à la page 76.

Finalement, il est important de noter que dans ces cas, le degré des polynômes est égal à l'ordre de la méthode, nous devons donc trouver des racines pour des polynômes d'ordre 3, 4 et 5. Ceci ne se fait pas de manière aussi simple qu'avec un polynôme de degré 2. De nouvelles techniques devront être utilisées. Dans (Esposito, Kumar et Pappas, 2001), il est suggéré de former la matrice compagnons et de trouver les valeurs propres de celle-ci. Il faut cependant faire attention, car ces méthodes demandent beaucoup de calcul et nous nous devons de respecter les contraintes de temps imposées par le système simulé.

4.4 Les Méthodes Runge-Kutta

L'idée des méthodes Runge-Kutta est de choisir des points dans l'intervalle courant (entre t_i et t_{i+1}), de prédire la valeur de la dérivée de la fonction en ces points et de faire une moyenne pondérée de ces dérivées afin d'effectuer un pas de simulation à l'aide de cette moyenne. Le

fait d'utiliser une moyenne pondérée des différents points nous permet d'obtenir une meilleure approximation de la pente dans l'intervalle et donc un meilleur résultat pour l'évaluation de la fonction simulé à la fin de l'intervalle.

Toutes les formes itératives pour les différents ordres de la méthode Runge-Kutta sont développées de la même manière. Pour cette raison, dans ce document nous présenterons les étapes pour trouver Runge-Kutta d'ordre 2 (RK2) et ceci nous servira de base pour comprendre le développement des ordres supérieurs. Il sera donc possible d'éviter les lourdes démonstrations de ces ordres sans compromettre pour autant la compréhension des enjeux reliés à ces méthodes.

Pour commencer, la forme itérative pour la méthode RK2 est donnée, pour une fonction quelconque $q(t, x(t))$, par les équations suivantes:

$$q(t_{k+1}, x_{t_{k+1}}) = q(t_k, x_{t_k}) + ak_1 + bk_2 \tag{4.19}$$

$$k_1 = h\dot{q}(t_k, x_{t_k}) \tag{4.20}$$

$$k_2 = h\dot{q}(t_k + \alpha h, x_{t_k} + \beta k_1) \tag{4.21}$$

En incluant les équations 4.20 et 4.21 dans l'équation 4.19, nous obtenons:

$$q(t_{k+1}, x_{t_{k+1}}) = q(t_k, x_{t_k}) + ah\dot{q}(t_k, x_{t_k}) + bh\dot{q}(t_k + \alpha h, x_{t_k} + \beta h\dot{q}(t_k, x_{t_k})) \tag{4.22}$$

Le dernier terme de cette équation peut être approximé en utilisant la série de Taylor autour des points t_k et x_{t_k} comme suit:

$$\dot{q}(t_k + \alpha h, x_{t_k} + \beta h\dot{q}(t_k, x_{t_k})) = \dot{q}(t_k, x_{t_k}) + \alpha h\dot{q}_t(t_k, x_{t_k}) + \beta\dot{q}_x(t_k, x_{t_k})\dot{q}(t_k, x_{t_k}) \tag{4.23}$$

Avec :

$$\dot{q}_t(t_k, x_{i_x}) = \frac{\partial q(t_k, x_{i_x})}{\partial t} \tag{4.24}$$

$$\dot{q}_x(t_k, x_{i_x}) = \frac{\partial q(t_k, x_{i_x})}{\partial x} \tag{4.25}$$

En incluant l'équation 4.23 dans l'équation 4.22 et si nous modifions un peu l'ordre d'apparition des coefficients nous obtenons finalement:

$$q(t_{k+1}, x_{i_{x+1}}) = q(t_k, x_{i_x}) + (a+b)h\dot{q}(t_k, x_{i_x}) + h^2(\alpha b\dot{q}_t(t_k, x_{i_x}) + \beta b\dot{q}_x(t_k, x_{i_x})\dot{q}(t_k, x_{i_x})) \tag{4.26}$$

Comme il est désirable d'obtenir, pour cette méthode, un ordre d'erreur similaire à celui de la série de Taylor, il est important de rappeler la forme de cette série lorsque nous avons deux variables. En gardant uniquement les trois premiers termes, cette forme itérative est donnée par:

$$q(t_{k+1}, x_{i_{x+1}}) = q(t_k, x_{i_x}) + h\dot{q}(t_k, x_{i_x}) + \frac{h^2}{2}\ddot{q}(t_k, x_{i_x}) + \ldots \tag{4.27}$$

Or en utilisant la dérivée totale de la fonction $q(t_k, x_{i_x})$ par rapport à t nous obtenons :

$$\ddot{q}(t_k, x_{i_x}) = \frac{\partial \dot{q}(t_k, x_{i_x})}{\partial t} + \frac{\partial \dot{q}(t_k, x_{i_x})}{\partial x}\frac{\partial x}{\partial t} = \dot{q}_t(t_k, x_{i_x}) + \dot{q}_x(t_k, x_{i_x})\dot{q}(t_k, x_{i_x}) \tag{4.28}$$

Donc, l'équation 4.27 peut être écrite de la façon suivante :

$$q(t_{k+1}, x_{i_{x+1}}) = q(t_k, x_{i_x}) + h\dot{q}(t_k, x_{i_x}) + \frac{h^2}{2}(\dot{q}_t(t_k, x_{i_x}) + \dot{q}_x(t_k, x_{i_x})\dot{q}(t_k, x_{i_x})) \tag{4.29}$$

En comparant l'équation 4.26 avec l'équation 4.29 nous pouvons trouver que:

$$a + b = 1 \tag{4.30}$$

$$\alpha b = \frac{1}{2} \tag{4.31}$$

$$\beta b = \frac{1}{2} \tag{4.32}$$

Nous avons donc 4 inconnus et trois équations. Ceci est un système avec un degré de liberté. En fonction des choix que nous effectuerons pour chaque paramètre, une variante différente de la méthode Runge-Kutta sera produite. En guise d'exemple, pour $\alpha = \beta = 1/2$, $a = 0$ et $b = 1$, nous obtenons la méthode du point milieu et pour $\alpha = \beta = 1$ et $a = b = 1/2$, nous obtenons la méthode de Euler modifiée.

Il en va de même pour les méthodes Runge-Kutta d'ordres supérieurs. Dans les équations suivantes, nous pouvons trouver la forme itérative pour la méthode RK3 (équations 4.33 à 4.36), pour la méthode RK4 (équations 4.37 à 4.41) et pour la méthode RK45 (équations 4.42 à 4.48) (ces formules, avec ces choix de coefficients, sont utilisées dans Matlab pour les méthodes OD3, ODE4 et ODE5 respectivement).

$$q(t_{k+1}, x_{t_{k+1}}) = q(t_k, x_{t_k}) + \left(\frac{2}{9}k_1 + \frac{1}{3}k_2 + \frac{4}{9}k_3\right) \tag{4.33}$$

$$k_1 = h\dot{q}(t_k, x_{t_k}) \tag{4.34}$$

$$k_2 = h\dot{q}\left(t_k + \frac{1}{2}h, x_{t_k} + \frac{1}{2}k_1\right) \tag{4.35}$$

$$k_3 = h\dot{q}\left(t_k + \frac{3}{4}h, x_{t_k} + \frac{3}{4}k_2\right) \tag{4.36}$$

$$q(t_{k+1}, x_{t_{k+1}}) = q(t_k, x_{t_k}) + \frac{1}{6}(k_1 + 2k_2 + 2k_3 + k_4) \tag{4.37}$$

$$k_1 = h\dot{q}(t_k, x_{t_k}) \tag{4.38}$$

$$k_2 = h\dot{q}\left(t_k + \frac{1}{2}h, x_{i_k} + \frac{1}{2}k_1\right) \tag{4.39}$$

$$k_3 = h\dot{q}\left(t_k + \frac{1}{2}h, x_{i_k} + \frac{1}{2}k_2\right) \tag{4.40}$$

$$k_4 = h\dot{q}\left(t_k + h, x_{i_k} + k_3\right) \tag{4.41}$$

$$q\left(t_{k+1}, x_{i_{k+1}}\right) = q\left(t_k, x_{i_k}\right) + \left(\frac{35}{384}k_1 + \frac{500}{1113}k_3 + \frac{125}{192}k_4 + \frac{2187}{6784}k_5 + \frac{11}{84}k_6\right) \tag{4.42}$$

$$k_1 = h\dot{q}\left(t_k, x_{i_k}\right) \tag{4.43}$$

$$k_2 = h\dot{q}\left(t_k + \frac{1}{5}h, x_{i_k} + \frac{1}{5}k_1\right) \tag{4.44}$$

$$k_3 = h\dot{q}\left(t_k + \frac{3}{10}h, x_{i_k} + \left[\frac{3}{40}k_1 + \frac{9}{40}k_2\right]\right) \tag{4.45}$$

$$k_4 = h\dot{q}\left(t_k + \frac{4}{5}h, x_{i_k} + \left[\frac{44}{45}k_1 + \frac{56}{15}k_2 + \frac{32}{9}k_3\right]\right) \tag{4.46}$$

$$k_5 = h\dot{q}\left(t_k + \frac{8}{9}h, x_{i_k} + \left[\frac{19372}{6561}k_1 + \frac{25360}{2187}k_2 + \frac{64448}{6561}k_3 + \frac{212}{729}k_4\right]\right) \tag{4.47}$$

$$k_6 = h\dot{q}\left(t_k + h, x_{i_k} + \left[\frac{9017}{3168}k_1 + \frac{355}{33}k_2 + \frac{46732}{5247}k_3 + \frac{49}{176}k_4 + \frac{5103}{18656}k_5\right]\right) \tag{4.48}$$

Le problème ici est que ces méthodes ne permettent pas une application directe de l'algorithme décrit plus haut. Elles utilisent bien plusieurs points afin d'effectuer leur approximation de la dérivée, mais, contrairement aux méthodes multipas linéaires comme Adams-Bashforth, qui utilisent les résultats précédant l'intervalle, ces points sont choisis à même le pas courant (entre t_k et t_{k+1} inclusivement). Aucune information des intervalles précédents n'est utilisée. En d'autres mots, il n'est pas possible de donner une expression pour $\overline{g}_\beta$ en fonction du point actuel et des points précédents ($g_{i_k}, g_{i_{k-1}}, g_{i_{k-2}}, \ldots$). Nous devons donc repenser légèrement l'algorithme afin de pouvoir l'appliquer aux méthodes Runge-Kutta.

Pour résoudre ce problème, il pourrait être intéressant d'utiliser les points intermédiaires des méthodes Runge-Kutta. En effet, de cette manière, il serait possible de tirer profit de cette information supplémentaire fournie par la méthode au lieu d'utiliser uniquement le résultat final soit la valeur de la fonction gardienne à chaque pas de calcul. Une fenêtre qui évolue au fur et à mesure que les points k_i sont calculés pourrait être utilisée pour évaluer la présence d'une discontinuité entre chacun d'entre eux. Pour ce faire, la fenêtre pourrait utiliser le point k_1 du pas actuel et les deux derniers k_s du pas précédent pour évaluer la présence d'une discontinuité entre le point k_1 et k_2. Ensuite, si aucune discontinuité n'est localisée, la fenêtre avancerait dans le temps et les points k_1 et k_2 du pas actuel serait utilisés ainsi que le dernier k_s du pas de calcul précédent (et ainsi de suite).

Dans ce chapitre, plusieurs méthodes ont été analysées afin de voir si elles étaient applicables à notre nouvel algorithme. Ceci nous a permis d'établir les points forts et les points faibles de chaque méthode et de voir s'il était avantageux de les utiliser. Il est maintenant temps de synthétiser ces informations afin d'être en mesure de poser l'algorithme définitif. Cette étape sera effectuée dans le chapitre suivant.

CHAPITRE 5

L'ALGORITHME

Nous avons vu au cours de la dernière section les avantages et les inconvénients de l'application de l'algorithme aux méthodes de résolution des équations différentielles les plus populaires. Pour Euler simple, il est impossible de localiser plus d'une discontinuité à la fois et la complexité de l'algorithme, par rapport à celle de la méthode de résolution, ne justifie pas son utilisation. Pour Adams-Bashforth d'ordres 3, 4 et 5, l'efficacité de l'algorithme est trop dépendante de la méthode de recherche des racines et, finalement, la nature même des méthodes Runge-Kutta ne permet pas une utilisation simple et directe de l'algorithme. Il faut donc trouver un moyen pour être en mesure d'utiliser l'algorithme de façon optimale. Dans cette section, des choix seront effectués afin de poser l'algorithme final qui sera ensuite utilisé pour effectuer les tests.

Premièrement, étant donné la synthèse des inconvénients qui a été faite dans l'introduction de ce chapitre, il a été décidé d'utiliser le polynôme d'Adams-Bashforth d'ordre 2 afin de détecter les évènements. Il sera donc possible de détecter jusqu'à deux évènements sans pour autant nuire à l'efficacité de l'ensemble de la simulation. Il a été décidé de ne pas utiliser d'ordre plus élevé que 2 puisque la recherche des racines pour un polynôme d'ordre 3 ou plus est trop ardue. De plus, le fait que 3 évènements peuvent se produire dans un même intervalle est un signe que le pas de calcul est peut-être trop grand. Il est donc plus aisé de réduire ce pas au lieu d'effectuer la recherche de plusieurs évènements

Puisque nous utilisons le polynôme d'Adams-Bashforth d'ordre 2, l'algorithme 1, donné plus bas, sera utilisé. Dans cet algorithme, nous effectuons M itérations d'une simulation avec un pas de calcul fixe égal à h. Aux lignes 2 et 3, nous calculons la fonction gardienne $g(x_{t_i}, \mu_{t_i})$ ainsi que sa dérivée $\dot{g}(x_{t_i}, \mu_{t_i})$. Ensuite, aux lignes 4 à 6 nous calculons les coefficients du polynôme du deuxième degré pour finalement y rechercher les racines à la ligne 7. Si l'une des racines est plus petite que h alors nous avons localisé un évènement.

Autrement, la simulation peut continuer sachant qu'il n'y aura pas de discontinuité dans le pas de calcul suivant.

Algorithme 6.1 Localisation des évènements d'ordre $2(M, h)$

1 : **pour** $k = 0$ à M **faire**
2 : $g_{t_k} = g(x_{t_k}, \mu_{t_k})$
3 : $\dot{g}_{t_k} = \dot{g}(x_{t_k}, \mu_{t_k})$
4 : $a = \left[\frac{\dot{g}_{t_k} - \dot{g}_{t_{k-1}}}{2h_{t_k}} \right]$
5 : $b = \dot{g}_{t_k}$
6 : $c = g_{t_k}$
7 : *Trouver, sélectionner et trier les racines du polynôme* $ah_{k+1}^2 + bh_{k+1} + c$
8: h_{k+1} = *La première racine dans le temps*
9: ***si*** $h_{k+1} < h$ ***alors***
10: *Évaluer* t_θ *et* $x(t_\theta)$
11: *Passer la discontinuité*
12: *Resynchroniser la simulation et calculer* $x(t_{k+1})$
13: ***Sinon***
14: $x_{t_{k+1}} = x_{t_k} + h\Phi(x_{t_k}, x_{t_{k-1}}, ..., \mu_{t_k}, \mu_{t_{k-1}}, ...)$
15: ***fin si***
16: ***fin pour***

Dans ce cas, les racines sont alors trouvées avec la méthode de la complétion du carré. Un maximum de deux évènements pourra être localisé avec l'aide des équations suivantes:

$$r_1 = \frac{\sqrt{b^2 - 4ac} - b}{2a} \tag{5.1}$$

$$r_2 = \frac{-\sqrt{b^2 - 4ac} - b}{2a} \tag{5.2}$$

Il nous reste à écarter les racines négatives ou complexes et de retourner la première chronologiquement parlant.

Lorsqu'une discontinuité sera détectée, disons au temps t_a, nous devrons déterminer les conditions initiales après l'évènement. Il s'agit en fait de faire évoluer le système jusqu'au temps t_a et de passer la discontinuité. La méthode d'écrite dans le chapitre 3 de (Kelper, 2002) sera utilisée. Cette méthode utilise le fait que les commutations sont considérées comme étant instantanées ainsi que le principe de la conservation d'énergie. En bref, lors d'un changement de topologie, l'énergie accumulée dans le circuit et celle accumulée dans chaque composante réactive est conservée. Il s'agit donc de déterminer le nouvel écoulement de cette énergie en appliquant la loi de tensions et des courants de Kirchhoff.

Finalement, ayant les conditions initiales en main, nous devons resynchroniser la simulation sur le pas initial et évaluer $x_{t_{n+1}}$. Pour ce faire, un pas de simulation complet sera effectué à partir de t_a pour évaluer $x_{t_{n+1}}$. Par la suite, une interpolation linéaire entre ces deux points sera effectuée afin d'évaluer $x_{t_{n+1}}$ se trouvant nécessairement entre les deux. La simulation peut alors reprendre son cours normal.

Une fois l'algorithme bien défini, nous devons maintenant le tester. Deux types de tests seront effectués afin d'établir l'efficacité de ce dernier. Les premiers tests seront effectués afin d'analyser la fonctionnalité de l'algorithme et, par la suite, nous analyserons si l'algorithme est utilisable dans un contexte temps réel.

CHAPITRE 6

TESTS FONCTIONNELS

Ayant présenté tout ce qui est nécessaire pour bâtir l'algorithme, il est maintenant temps de confronter la théorie à la pratique. Ce chapitre servira donc à rendre compte des tests fonctionnels qui ont été effectués sur un circuit de base. Il ne sera pas encore question du concept de temps réel, car il est nécessaire de démontrer dans un premier temps la capacité de l'algorithme et par la suite son efficacité. Pour ce faire, l'environnement de test sera présenté. Par la suite, les résultats pour le circuit seront analysés et finalement une synthèse conclura le chapitre.

6.1 Descriptions de l'environnement de test

En premier lieu, une description des tests effectués doit être faite. Dans cette description les méthodes de simulation, le circuit électrique utilisé comme banc de test et les algorithmes de localisation utilisés aux fins de comparaison seront brièvement présentés.

6.1.1 Méthode de simulation

L'un des buts de ce travail est d'implémenter l'algorithme dans l'environnement de simulation proposé par les logiciels *Matlab* et *Simulink* de *The Mathworks*. Pour effectuer son travail de simulation, *Simulink* utilise l'approche des variables d'états. Cette approche est très simple d'utilisation et elle a donc été utilisée dans le cadre de ce travail.

Les variables d'états sont extraites des équations du premier ordre de la dynamique du système sous analyse. Dans le domaine du génie électrique, elles sont souvent reliées aux composantes dynamiques telles que les condensateurs et les inductances. Une fois extraites, la dynamique de ces états ainsi que les sorties du système sous analyse sont représentées à l'aide de 4 matrices. Par exemple, pour un système linéaire invariant dans le temps, la

représentation d'état est donnée par les deux équations 6.1 et 6.2. De plus, chacune des matrices formant ces équations porte un nom qui est indiqué dans le Tableau 6.1.

$$\dot{x} = Ax + B\mu \tag{6.1}$$

$$y = Cx + D\mu \tag{6.2}$$

Tableau 6.1 Nom des matrices pour la simulation par variables d'état

A	Matrice d'évolution ou matrice du système
B	Matrice de commande ou matrice de l'entrée de commande
C	Matrice d'observation ou matrice de sortie
D	Matrice directe

Dans le but d'effectuer les tests sur l'algorithme de localisation proposé dans ce travail et de pouvoir faire des comparaisons, un programme a été développé en C++ qui utilise cette méthode de simulation. Ce programme utilise donc les variables d'états pour effectuer la même simulation à l'aide de plusieurs méthodes de localisation des discontinuités différentes. Ceci permet d'obtenir une base solide pour effectuer des comparaisons, en plus de fournir une première idée sur l'efficacité de l'algorithme présenté dans ce document. La section suivante donnera plus d'information au sujet de ces algorithmes de localisation.

6.1.2 Les algorithmes de localisation utilisés dans les tests

En plus de l'algorithme présenté dans ce document, deux autres méthodes de localisation des discontinuités seront utilisées aux fins de comparaison.

En plus d'être utilisé comme référence pour la comparaison, le premier algorithme a été choisi pour mettre en évidence l'importance d'utiliser une méthode de localisation plus sophistiquée et plus efficace pour effectuer une simulation d'un système dynamique hybride. Cet algorithme consiste à attendre au pas de calcul suivant afin de vérifier si le signal simulé

a croisé la fonction gardienne qui déclenche les évènements. Dans les lignes qui suivent, le pseudo-code de cet algorithme est présenté.

Algorithme 7.1 Localisation des évènements au pas suivant *(M. h. limite)*

pour k :=0 à M **faire**
  $g_k = g(x_{t_k})$
  si $g_k >$ *limite* ***alors***
    Changer la dynamique du système (modification des matrices du système)
  fin si
  $x_{t_{k+1}} = x_{t_k} + hf(x_{t_k}, u_{t_k})$
fin pour

Le deuxième algorithme utilisé se nomme la commutation précise. Il a été développé par Bruno De Kelper et a été présenté dans (Kelper, 2002). Une fois les calculs d'un pas de simulation terminés et juste avant de passer au pas suivant, cet algorithme vérifie s'il y a eu une discontinuité dans le pas présent (une méthode telle que la comparaison des signes du résultat de la fonction gardienne est utilisée pour cette étape). Si tel est le cas, une interpolation est effectuée et la discontinuité est ainsi localisée. Le système est alors mis à jour en tenant compte de la nouvelle topologie du circuit et le pas erroné est corrigé à l'aide d'une resynchronisation. Cette resynchronisation est en fait un pas complet supplémentaire qui a comme point de départ le moment de la discontinuité et par la suite, une seconde interpolation est effectuée pour corriger le pas initialement erroné se trouvant entre la discontinuité et le pas supplémentaire. Le pseudo-code de cet algorithme est présenté dans les lignes qui suivent. De plus, l'algorithme est illustré à la Figure 2.2.

Algorithme 7.2 Localisation des évènements avec la commutation précise *(M. h. limite)*

pour k :=0 à M **faire**
  $x_{t_{k+1}} = x_{t_k} + hf(x_{t_k}, u_{t_k})$
  $g_{k+1} = g(x_{t_{k+1}})$
  si $g_{k+1} >$ *limite* ***alors***
    localisation de la discontinuité avec une interpolation linéaire
    changer la dynamique du système (modification des matrices du système)
    pas supplémentaire avec point de départ t_0 : $x_{t_{0+h}} = x_{t_0} + hf(x_{t_0}, u_{t_0})$
    synchronisation pour corriger $x_{t_{k+1}}$ *avec une interpolation linéaire*

9 : *fin si*
10: *fin pour*

Déjà nous pouvons voir que la complexité de l'algorithme a augmenté. En effet, pour chaque discontinuité, la charge de calcul augmente considérablement, car le pas de calcul présent doit être recalculé complétement afin de le corriger. Par contre, nous verrons dans une section subséquente de ce chapitre que nous gagnons énormément au point de vue précision des résultats.

6.1.3 Les circuits utilisés dans le test

Afin de faire les tests, un circuit de base a été utilisé. Ce circuit sera simulé dans deux configurations différentes. Chacune de ces configurations sera utilisée afin de démontrer une caractéristique précise du nouvel algorithme. Dans sa première configuration, le circuit démontrera l'efficacité de la détection d'une discontinuité par rapport aux autres méthodes et dans sa seconde configuration, il sera utilisé pour démontrer que, même avec un nombre pair de discontinuité, l'algorithme présenté dans ce document demeure valide ce qui n'est pas nécessairement le cas pour toutes les méthodes de détection.

a) Circuit à deux diodes avec une charge RL

Le montage est un simple circuit à deux diodes avec une charge inductive alimentée par une source de tension sinusoïdale. Le schéma de ce circuit est présenté sur la Figure 6.1. De ce circuit, nous nous intéressons particulièrement au courant parcourant la branche RL en fonction du temps. Il sera démontré que ce courant dépend de la tension appliquée à l'entrée du circuit et que le simulateur aura à utiliser deux topologies différentes. En effet, lorsque la tension appliquée à l'entrée sera positive, la diode D1 sera en conduction, mais pas la diode D2 qui sera polarisée en inverse. La boucle D1, R et L reliée à la source de tension sinusoïdale est donc la première topologie dans laquelle évoluera le simulateur. Dans un deuxième temps, lorsque la tension sera négative, ce sera au tour de la diode D1 d'être

polarisé en inverse et le courant évoluera donc dans la boucle D2, R et L. Ceci est la deuxième topologie. Les algorithmes devront donc détecter le plus précisément possible l'instant où le simulateur devra changer de topologie. Finalement, il est important de noter que pour les fins de ces tests, les diodes seront considérées comme idéales. De plus, les commutations seront simultanées. Ceci veut dire que si la diode D2 ouvre, la diode D1 ferme instantanément et vice versa. Nous pouvons donc représenter les diodes comme des résistances à deux états. Lorsque la diode sera en conduction, la résistance équivalente sera très faible (de l'ordre de 1μ Ω) et lorsque la diode sera ouverte, la résistance équivalente sera très grande (de l'ordre de 1M Ω)

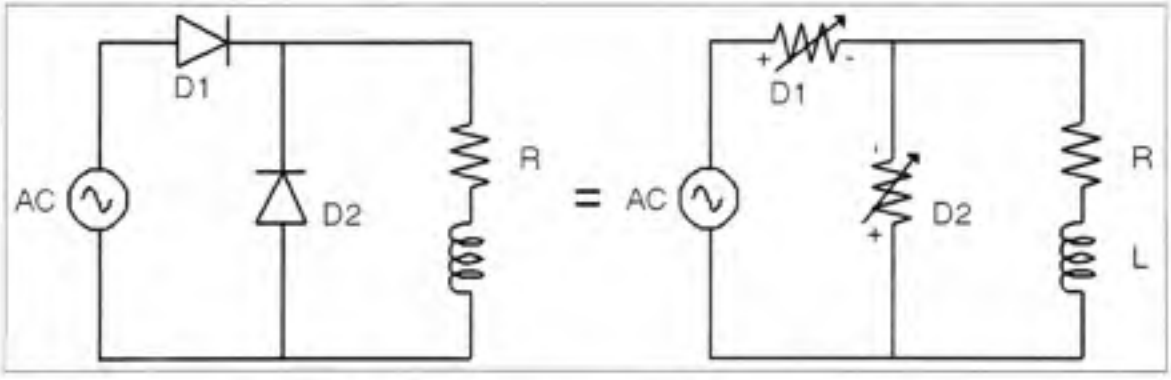

Figure 6.1 Circuit à deux diodes avec une charge RL.

b) Différence entre les deux configurations

Avant d'aller plus loin, il est important de décrire les deux configurations avec lesquelles nous effectuerons les tests afin de valider deux points importants du nouvel algorithme. Rappelons que dans un premier temps, nous voulons valider l'efficacité du nouvel algorithme à détecter des discontinuités. Le circuit évoluera donc de manière à obtenir une discontinuité simple dans un pas de simulation. Par la suite, nous validerons la capacité du nouvel algorithme à détecter plusieurs discontinuités dans un même pas de simulation. Le circuit sera donc modifié de manière à obtenir cette fameuse discontinuité double. La différence entre ces deux configurations se situe au niveau de la source. Cette dernière est décrite par les équations 6.3 et 6.4.

$$A\sin(\omega t + \phi) + B \tag{6.3}$$

$$\omega = 2\pi f \tag{6.4}$$

Dans la première configuration, la source aura les caractéristiques suivantes :

$$A = 100 \tag{6.5}$$

$$B = 100 - 0.001 \tag{6.6}$$

$$f = 60 \tag{6.7}$$

$$\phi = 0 \tag{6.8}$$

Ainsi, la première discontinuité sera située à $t = 0.012488137281057$ et elle sera simple (uniquement une discontinuité dans le même pas de simulation).

Dans sa deuxième configuration, la source aura les mêmes caractéristiques à l'exception du déphasage qui sera :

$$\phi = 3\pi/2 \tag{6.9}$$

Ce déphasage aura pour effet de déplacer la première discontinuité dans le temps. De plus, avec ce décalage, il y aura deux discontinuités dans le même pas de calcul. Ces dernières se situeront à $t = 0.0166554803853567$ et $t = 0.016678529459879$. (Il est important de noter que la localisation des discontinuités pour les deux configurations a été effectuée à l'aide du logiciel *Matlab*.).

c) Les matrices d'états et les fonctions gardiennes

Afin de former les matrices pour la représentation d'état nécessaire pour la simulation, nous devons d'abord établir les équations qui régissent le circuit en fonction du temps. Étant

donné qu'il y a uniquement une composante dynamique dans le circuit, nous aurons besoin d'un seul état pour représenter le circuit. De plus, puisque nous nous intéressons particulièrement au courant il va de soit que cet état soit en fait le courant qui parcourt la branche RL en fonction du temps, le courant I_L. Aussi, puisque nous devons localiser les moments où la polarisation d'une des deux diodes change, les fonctions gardiennes (une par diode) seront données en fonction de leur tension respective soit V_{D1} et V_{D2}. Nous devons donc nous appliquer à trouver une expression pour chacune de ces deux variables qui devra être fonction de la variable d'état et de l'entrée du système (I_L et V_{SRC}). De plus, nous devons trouver une expression pour la dynamique de la variable d'état ($\dot{I}_L$) qui elle aussi devra être fonction de la variable d'état et de l'entrée.

Donc, en premier lieu, par la loi d'ohm, nous pouvons poser les équations suivantes :

$$V_{D1} = R_{D1} I_{D1} \tag{6.10}$$

$$V_{D2} = R_{D2} I_{D2} \tag{6.11}$$

$$V_R = R I_L \tag{6.12}$$

$$V_L = L \frac{dI_L}{dt} = L \dot{I}_L \tag{6.13}$$

De plus, nous savons que, par la loi des mailles, la somme des tensions aux bornes des composantes présentes dans une boucle fermée d'un circuit doit être égale à zéro. Nous obtenons donc les équations suivantes:

$$V_{SRC} - V_{D1} + V_{D2} = 0 \tag{6.14}$$

$$-V_{D2} - V_R - V_L = 0 \tag{6.15}$$

Finalement, avec la loi des nœuds, nous trouvons que :

$$I_{D1} + I_{D2} - I_L = 0 \tag{6.16}$$

En combinant les équations 6.12, 6.13 et 6.15, nous pouvons obtenir :

$$I'_{D2} = -RI_L - L\dot{I}_L \tag{6.17}$$

Des équations 6.10, 6.11, 6.16 et 6.17, nous pouvons tirer :

$$I'_{D1} = \frac{R_{D1}}{R_{D2}}(R_{D2} + R)I_L - \frac{R_{D1}}{R_{D2}}\dot{I}_L \tag{6.18}$$

Ensuite, avec les équations 6.14, 6.17 et 6.18, nous trouvons que:

$$\dot{I}_L = \frac{-(RR_{D1} + RR_{D2} + R_{D1}R_{D2})}{(R_{D1} + R_{D2})L} I_L + \frac{R_{D2}}{(R_{D1} + R_{D2})L} I'_{SRC} \tag{6.19}$$

De plus en combinant 6.17 et 6.19, nous pouvons poser I'_{D2} ainsi:

$$I'_{D2} = \frac{R_{D1}R_{D2}}{(R_{D1} + R_{D2})} I_L - \frac{R_{D2}}{(R_{D1} + R_{D2})} I'_{SRC} \tag{6.20}$$

Finalement, en combinant 6.18 et 6.19, nous pouvons poser I'_{D1} comme étant égal à :

$$I'_{D1} = \frac{R_{D1}R_{D2}}{(R_{D1} + R_{D2})} I_L + \frac{R_{D1}}{(R_{D1} + R_{D2})} I'_{SRC} \tag{6.21}$$

En résumé, lors de la simulation, les algorithmes devront localiser le plus précisément possible le temps où l'une de deux fonctions gardienne, $g_1 = I'_{D1}$ et $g_2 = I'_{D2}$, changera de polarité. De plus, la simulation évoluera en fonction des équations suivantes :

$$\dot{x} = Ax + B\mu \tag{6.22}$$

$$y = Cx + D\mu \tag{6.23}$$

$$x = I_L \tag{6.24}$$

$$\dot{x} = \dot{I}_L \tag{6.25}$$

$$y = I_L \tag{6.26}$$

$$A = \frac{-(RR_{t1} + RR_{t2} + R_{t1}R_{t2})}{(R_{t1} + R_{t2})L} \tag{6.27}$$

$$B = \frac{R_{t2}}{(R_{t1} + R_{t2})L} \tag{6.28}$$

$$C = 1 \tag{6.29}$$

$$D = 0 \tag{6.30}$$

6.2 Représentation de l'erreur lors des tests

Lors de la présentation des résultats obtenus à l'aide des différentes simulations, il sera important de donner l'ordre de l'erreur dans la localisation des discontinuités afin d'évaluer l'efficacité des différentes méthodes utilisées. Pour ce faire, il a été choisi de présenter cette erreur à l'aide d'un pourcentage par rapport au pas de simulation. En effet, aussi petite soit l'erreur, si elle est deux fois plus grandes que le pas de simulation, cette dernière devient considérable et la localisation devient inutilisable. Or, en utilisant un pourcentage par rapport à une référence bien connue (caractéristique des simulations à pas fixe), le lecteur sera en mesure de se faire une meilleure idée de cette erreur et par conséquent de la précision des résultats. Le calcul pour obtenir ce taux est donc le suivant :

$$e = \frac{\left| T_{théorique} - T_{estimé} \right|}{\Delta T} \tag{6.31}$$

Dans cette équation, $T_{discontinuité}$ est la localisation exacte présentée plus haut, $T_{calculée}$ est la localisation obtenue dans la simulation avec l'une des méthodes de localisation et ΔT est le pas de calcul utilisé lors de la simulation.

Une erreur de 0% est donc égale à une localisation parfaite et une erreur de 100 % est équivalente à une erreur aussi grande que le pas de calcul de la simulation. De plus, il est important de noter que cette erreur peut devenir plus grande que 100 %.

Il faut cependant faire attention dans l'interprétation de ce pourcentage, car ce dernier peut être influencé par plusieurs points tels que la taille du pas de simulation, la position de la discontinuité dans le pas de simulation et la dynamique de la fonction gardienne.

En effet, dans le premier cas, lorsqu'aucune méthode de détection n'est utilisée, le pas de calcul normal qui suit la discontinuité est pris comme l'instant réel de celle-ci. Ainsi, si la discontinuité est réellement située à la toute fin du pas de simulation, le pourcentage d'erreur sera beaucoup plus petit et inversement, si la discontinuité est située au tout début, le pourcentage d'erreur sera très grand. Il est impossible de contrôler cette variation dans l'erreur ni en changeant le pas de simulation ni en changeant la méthode d'intégration, car il est impossible de prédire la position de la discontinuité dans le pas. Ceci est un désavantage majeur de cette méthode de localisation.

Dans le cas de la commutation précise, le pourcentage d'erreur dépend principalement du pas de simulation par rapport à la dynamique de la fonction gardienne. Si le pas de calcul est très petit, la fonction gardienne sera presque une droite et la localisation sera plus précise (ce qui fera diminuer le pourcentage d'erreur). En contrepartie, si le pas de simulation est plus grand, la fonction gardienne ne pourra plus être considérée comme étant une droite et l'erreur dans la localisation sera grandement influencée par la dynamique de la fonction gardienne. Il est donc possible d'avoir un certain contrôle sur ce pourcentage d'erreur, mais au prix d'un processus d'essai-erreur qui rend ce contrôle de l'erreur difficile à appliquer.

Finalement avec la nouvelle méthode de localisation, le pourcentage d'erreur sera influencé par deux facteurs importants. Le premier est bien évidemment la taille du pas de calcul, car celui-ci influence le polynôme utilisé pour calculer les racines. Et le second est le choix de la méthode d'intégration utilisée pour implémenter la méthode de détection. Il a été prouvé dans le chapitre 5 que différentes méthodes pouvaient être utilisées pour calculer les points nécessaires pour former le polynôme et chacune de ces méthodes induit une certaine erreur d'approximation qui lui est propre. Cette erreur d'approximation est fonction de l'ordre de la méthode et du pas de calcul utilisé lors de la simulation. Par conséquent, le pourcentage d'erreur peut être contrôlé par ces deux facteurs qui sont connus dès le départ. Il est donc possible de choisir d'avance quel pas de simulation sera utilisé et quelle méthode d'intégration sera utilisée, afin d'obtenir un certain pourcentage d'erreur. Ce contrôle sur le pourcentage d'erreur représente un avantage considérable de la nouvelle méthode sur les méthodes précédentes.

6.3 Résultats obtenus avec la première configuration

Le but recherché en utilisant le circuit RL dans sa première configuration était de présenter le comportement général du nouvel algorithme en présence d'une discontinuité et aussi évaluer son efficacité à bien localiser les discontinuités. De plus, afin de démontrer que l'algorithme est indépendant de la méthode de résolution des ODE utilisée, deux groupes de résultats seront présentés. Dans un premier temps, les résultats avec la méthode Runge-Kutta d'ordre 4 (RK4) seront présentés et par la suite les résultats pour la méthode Adams-Bashforth d'ordre 3 (AB3) seront présentés.

Tel qu'espéré, le nouvel algorithme a obtenu de bons résultats. Dans le Tableau 6.2, la localisation de la discontinuité à l'aide des trois méthodes est présentée. Nous pouvons voir qu'en n'utilisant aucune méthode (S.M. dans le tableau), la localisation est très imprécise. En effet, une erreur représentant 23.73 % du pas de calcul a été introduite. Avec la commutation précise (C.P. dans le tableau), les résultats sont meilleurs, mais il y a encore place à l'amélioration. L'erreur introduite par cette méthode représente 18.096 % du pas de

calcul. Les meilleurs résultats ont été obtenus avec la nouvelle méthode (N.M. dans le tableau). Avec cette dernière, une erreur de l'ordre de 10^{-8} a été introduite soit 0.0381 % du pas de calcul. Ces chiffres démontrent bien que la méthode est très efficace pour localiser les discontinuités.

Tableau 6.2 Résultats obtenus avec la méthode RK4

	Localisation	**Écart**	**% du pas de calcul**
S.M.	0.0125000000000000	$1.1862718943 \times 10^{-5}$	23.73
C.P.	0.0124971853982406	$9.048117183 \times 10^{-6}$	18.096
N.M.	0.0124881182217550	1.9059302×10^{-8}	0.0381

Sur le Tableau 6.3, il est possible de voir que les résultats obtenus avec la méthode AB3 sont très similaires à ceux obtenus avec RK4. La précision des résultats est la même, ce qui démontre bien que l'efficacité de l'algorithme est indépendante de la méthode de résolution utilisée lors de la simulation.

Tableau 6.3 Résultats obtenus avec la méthode AB3

	Localisation	**Écart**	**% du pas de calcul**
S.M.	0.0125000000000000	$1.1862718943 \times 10^{-5}$	23.73
C.P.	0.0124971853982808	$9,048117223 \times 10^{-6}$	18.096
N.M.	0.0124881182218099	1.9059248×10^{-8}	0.0381

6.4 Résultats obtenus avec la deuxième configuration

Dans cette section, nous voulons valider si le nouvel algorithme est en mesure de détecter une double discontinuité dans un même pas de calcul. Encore une fois, une comparaison a été effectuée entre plusieurs méthodes de localisation. Normalement nous devrions nous attendre à ce qu'uniquement le nouvel algorithme détecte cette double discontinuité et c'est bel et bien ce qui a été obtenu comme résultats. En effet, la commutation précise n'a pas été en mesure de détecter cette double discontinuité ce qui est tout à fait normal, car le signe du signal de garde n'a pas changé entre les deux pas.

Comme il est possible de le voir avec le Tableau 6.4, l'algorithme a localisé les discontinuités avec une précision de l'ordre de 10^{-7} et 10^{-9}. Ceci représente 0.00901% et 1.353% du pas de calcul. Encore une fois, cette précision est très intéressante. Finalement avec le Tableau 6.5, il est possible de voir, ici aussi, que ces résultats sont indépendants de la méthode de résolution.

Tableau 6.4 Résultats obtenus avec la méthode RK4

	Localisation	**Écart**	**% du pas de calcul**
r_1	0. 0166548037348844	$6.76650472 \times 10^{-7}$	1.353
r_2	0. 0166785339654062	4.505527×10^{-9}	0.00901

Tableau 6.5 Résultats obtenus avec la méthode AB3

	Localisation	**Écart**	**% du pas de calcul**
r_1	0.0166548037349659	$6.76650391 \times 10^{-7}$	1.353
r_2	0.0166785339653447	4.505465×10^{-9}	0.00901

6.5 Synthèse des résultats

Suite à cette présentation des résultats, il est déjà possible de tirer quelques conclusions sur le nouvel algorithme. En effet, en ce qui concerne la précision des résultats, le nouvel algorithme est très performant. En utilisant cet algorithme, la précision de la localisation est passée de 18 % du pas de calcul avec la commutation précise à 0.0381 % avec la nouvelle méthode. Ceci est un gain très intéressant. Aussi, il a été prouvé que, contrairement aux autres méthodes de localisation, le nouvel algorithme était en mesure de détecter les doubles discontinuités en gardant la même précision.

Ceci est bien intéressant, mais il faut savoir à quel prix ces résultats ont été obtenus. Aussi précis que soient les résultats, si les contraintes de temps ne sont pas respectées, ces derniers n'ont aucune valeur pour le type de simulation qui nous intéresse. Le prochain chapitre

s'intéressera donc à l'implémentation temps réel de l'algorithme et il tentera de valider son utilisation dans un contexte où les contraintes de temps sont importantes.

CHAPITRE 7

IMPLÉMENTATION TEMPS RÉEL DE L'ALGORITHME

Une fois que nous avons démontré que l'algorithme est bel et bien fonctionnel et efficace pour localiser les discontinuités, il est nécessaire d'étudier sa capacité à effectuer des simulations temps réel. Cette étude est indispensable, puisque peu importe la précision de localisation d'un algorithme, s'il ne respecte pas les contraintes de temps imposé par le système simulé, il n'est d'aucun recourt dans une simulation temps réel. Le nouvel algorithme sera donc implémenté dans un environnement temps réel et une évaluation de la précision des résultats, ainsi que du temps nécessaire pour obtenir ces résultats, sera effectuée. Comme dans le chapitre précédent, ces tests seront effectués avec deux autres méthodes de localisation afin de nous donner une base pour la comparaison.

7.1 Environnement de test

L'environnement temps réel utilisé sera *xPC Target*. Comme l'indique (Mathworks, 2008), *xPC Target* est un environnement de travail pour effectuer des simulations temps réel sur des ordinateurs de type PC. Pour effectuer une simulation avec ce logiciel, il est nécessaire d'avoir un ordinateur hôte (« *host PC* ») sur lequel Matlab est exécuté et un ordinateur cible (« *target PC* ») sur lequel une instance de *xPC Target* est exécutée. Sur l'ordinateur hôte, une application basée sur le modèle *Simulink* est développée et convertie en application C/C++ avec *Real Time Workshop*. Cette application est ensuite envoyée par lien Ethernet vers la cible afin d'être exécutée en temps réel et sans être interrompue par des services de Windows ou toutes autres applications. Ceci est possible, car *xPC Target* est le seul programme en exécution sur l'ordinateur cible lors d'une simulation. Il est donc possible de penser à *xPC Target* comme étant un système d'exploitation dédié à la simulation temps réel de modèle *Simulink*.

Dans un premier temps, le modèle du circuit à deux diodes du chapitre précédent sera développé dans *Simulink*. Par la suite, ce modèle sera converti en une application C/C++

exécutable sur un ordinateur cible et, avant d'effectuer la compilation du code généré par *Real Time Workshop*, les modifications nécessaires pour implémenter les différents algorithmes de localisation seront effectuées sur le code nouvellement généré. Afin de ne pas alourdir inutilement ce texte, les étapes de ces modifications ne seront pas présentées dans ce chapitre. Par contre si le lecteur est intéressé à connaître ces étapes, il peut consulter l'annexe « Code source des applications modifiées *xPC Target* » à la page 80. Finalement, le code sera compilé et exécuté sur l'ordinateur cible. Les résultats seront ensuite récupérés pour être analysés sur l'ordinateur hôte.

La présentation de l'analyse se fera en deux étapes. Dans un premier temps, une présentation des résultats de chacune des méthodes sera effectuée. Dans cette présentation, les temps moyen et maximal pour effectuer un pas de simulation seront démontrés en plus du résultat de la localisation effectué par l'algorithme donné. Ces étapes seront effectuées pour un circuit présentant uniquement une discontinuité et un circuit ayant une discontinuité double. Par la suite, une synthèse de ces résultats sera effectuée et une conclusion déterminera l'intérêt à utiliser un tel algorithme dans un contexte temps réel.

7.2 Modèle *Simulink* utilisé pour les tests

Le modèle *Simulink* utilisé dans ce chapitre pour effectuer les tests est une adaptation du circuit présenté à la Figure 6.1 du chapitre précédent. De légères modifications ont été apportées à ce circuit afin de prendre en compte les simplifications reliées aux diodes. Ainsi, les diodes ont été remplacées par des résistances à valeur fixe. De plus, des ports de sortie et un oscilloscope de type *Scope (xPC)* ont été ajoutés afin de pouvoir visualiser et récupérer les résultats. Une fois ces modifications apportées, le modèle présenté à la Figure 7.1 a été obtenu.

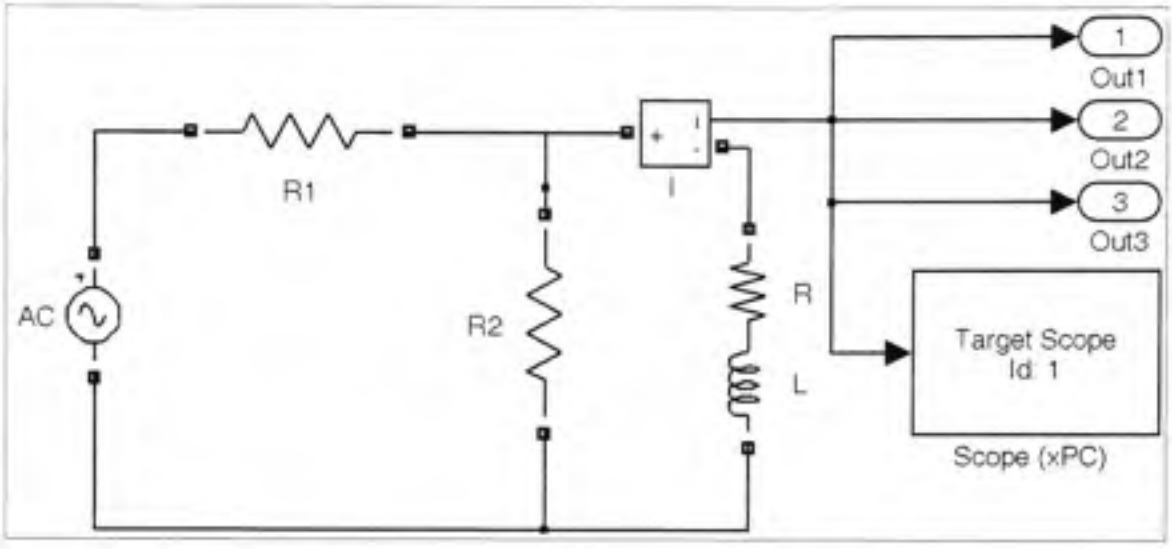

Figure 7.1 Modifications sur le circuit RL pour *xPC Target.*

Dans cette figure, les résistances R1 et R2 sont utilisées pour modéliser les diodes D1 et D2 respectivement (rappelons que ces diodes sont considérées comme idéales et qu'elles sont donc l'équivalente d'une résistance ayant une très grande valeur lorsqu'elles sont polarisées en inverse et qu'elles sont l'équivalente d'une résistance ayant une très faible valeur lorsqu'elles sont polarisées en directe). Dans le cas de cette simulation, la résistance de polarisation directe est égale à 1 μΩ et la résistance de polarisation indirecte est égale à 1 MΩ. Le bloc « *I* » est un ampèremètre utilisé pour mesurer les ampères qui parcourent la charge inductive représentée par la branche RL de 10Ω et 1mH. Ensuite, à la gauche, la source AC qui à été configurée avec une amplitude de 100 volts et une fréquence de 60 Hz. Finalement, à la droite du circuit il est possible de voir les ports de circuit et l'oscilloscope précédemment discuté.

7.3 Simulation temps réel

De ce modèle, un programme C/C++ généré par *Real Time Workshop* a été obtenu. Ce dernier à été exécuté sur l'ordinateur cible à 7 reprises. Dans un premier temps, la simulation a été effectuée sur 180 millisecondes avec un pas de simulation de 50 microsecondes et un biais de 101 volts afin d'être certain qu'aucune discontinuité ne perturbe ces résultats

préliminaires. Ceci était nécessaire pour obtenir une base pour le temps d'exécution normalement nécessaire afin d'effectuer la simulation temps réel. Par la suite, pour chaque méthode, le biais a été configuré 99.999 volts afin d'exhiber une discontinuité simple dans un premier temps et double dans un second temps. Le résultat pour chacun de ces cas est présenté dans les sections suivantes.

7.3.1 Simulation sans discontinuité ni méthode de localisation

Afin d'obtenir une base pour la comparaison, une configuration sans discontinuité a été effectuée. Il est possible de voir les résultats obtenus pour cette configuration sur la Figure 7.2. Le graphique du haut représente le courant dans la charge inductive en fonction du temps et le graphique du bas est le temps d'exécution pour la tâche en fonction du temps ou *Task Execution Time* (*TET*). De ce graphique il est possible de trouver le temps moyen pour chaque pas de simulation et le temps maximal pour ces mêmes pas de simulation. Dans ce cas précis, le temps moyen pour chaque pas est de 9.06579 microsecondes et le temps maximal est de 10.058 microsecondes. Il est clair que chaque pas est bien en dessous du pas de simulation donc toutes les contraintes de temps sont entièrement respectées. L'ajout des algorithmes de localisation à ce circuit devra être fait de manière à minimiser l'impact sur ce TET moyen et maximal, car ce sont ces derniers qui importent dans l'évaluation d'une méthode de localisation dans un contexte temps réel (en plus de la précision des résultats).

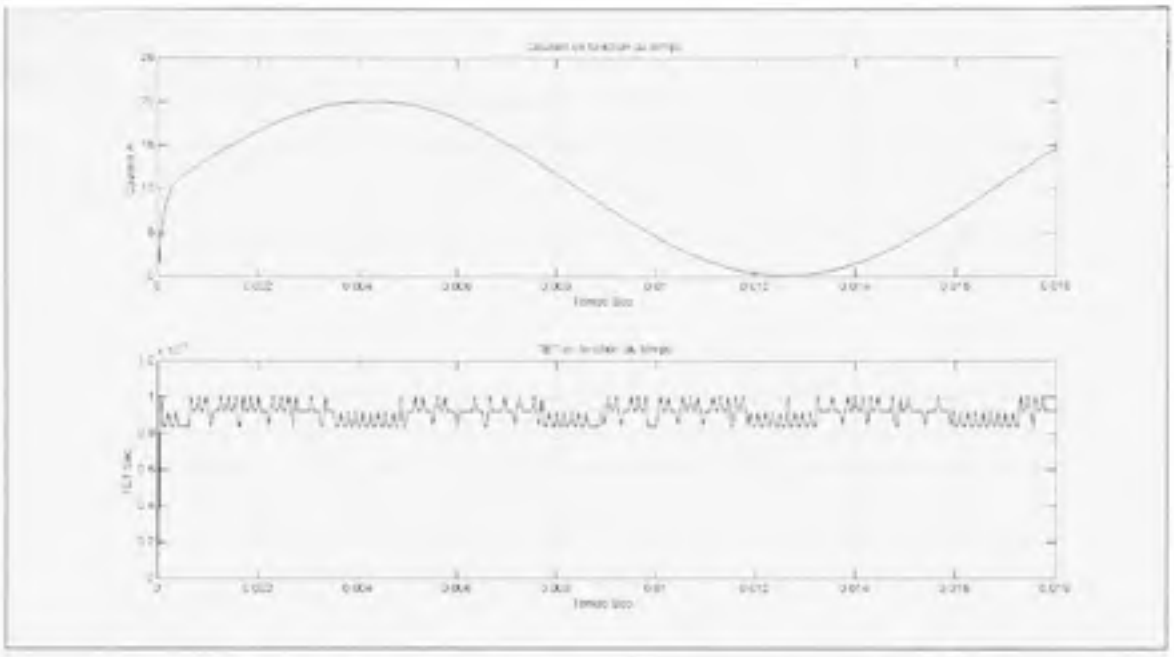

Figure 7.2 Simulation du circuit sans discontinuité sous *xPC Target.*

7.3.2 Simulation sans méthode de localisation

Pour la simulation sans algorithme de localisation, nous avons utilisé la méthode du changement de signe pour la détection des discontinuités. Tel qu'indiqué dans les chapitres précédents, cette méthode est implémentée simplement en ajoutant une évaluation du signal de sortie avant de calculer le pas suivant.

Les résultats obtenus pour la simulation exhibant une discontinuité simple sont illustrés sur la Figure 7.3. Cette méthode a détecté la discontinuité à 12,5 millisecondes. Ce grand écart avec la localisation réelle de la discontinuité est normal puisque si nous n'utilisons pas de méthode particulière pour faire la localisation, les discontinuités seront uniquement détectées au pas suivant et c'est exactement ce qui se passe dans ce cas précis. Ensuite, le temps moyen de la simulation est de 9.42799 microsecondes et le temps maximal est de 12.572 microsecondes. Encore une fois, ces résultats sont logiques, puisque nous avons effectué de légères modifications sur le code, ce qui a eu pour effet d'ajouter une petite charge de travail

sur le processeur. C'est pour cette raison que les temps nécessaires pour effectuer un pas sont légèrement plus grands que dans le cas précédent.

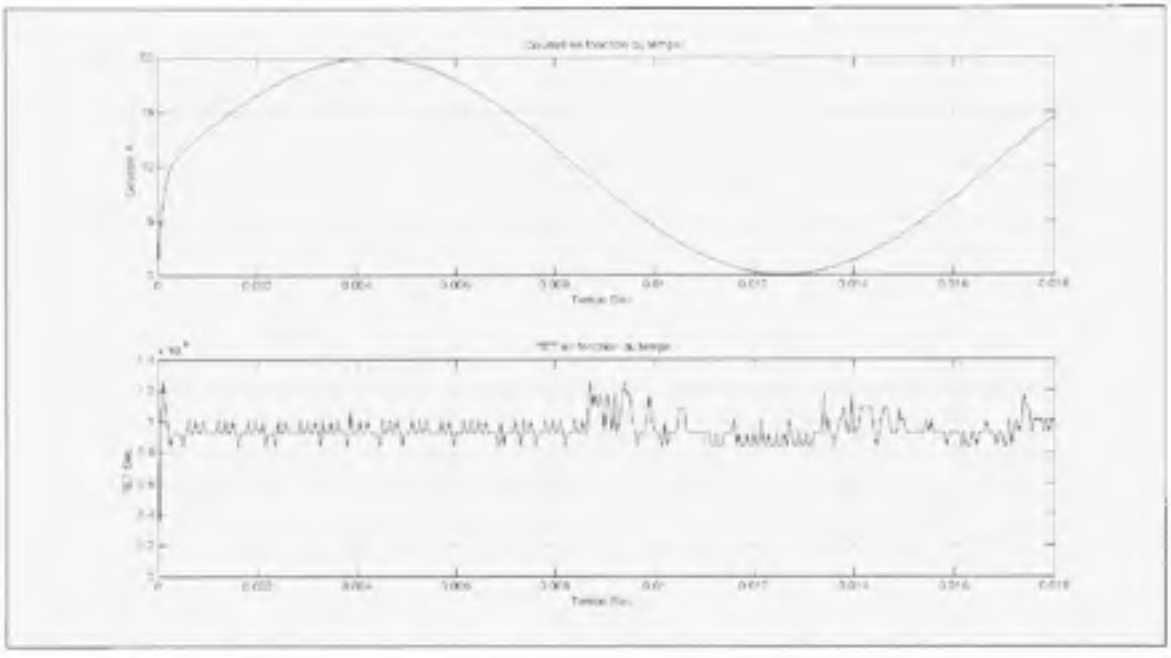

Figure 7.3 Simulation sans méthode de localisation sous *xPC Target*.

Dans le cas de la discontinuité double, les résultats n'ont apporté aucune surprise. En effet, l'algorithme a complètement échoué à localiser les discontinuités. Les résultats obtenus pour les temps moyens et maximaux ont été similaires à ceux présentés plus haut, soit un temps moyen de 9.74619 microsecondes et un temps maximal de 13.41 microsecondes.

7.3.3 Simulation avec la méthode de commutation précise

Dans un troisième cas, la méthode de la commutation précise a été implémentée. Les résultats obtenus sont présentés à la Figure 7.4. Dans le cas de la discontinuité simple, la localisation est à 0.012497185398240 59 seconde. Ce résultat est légèrement plus précis que le cas précédent, car des calculs ont été effectués pour interpoler de façon précise la localisation de la discontinuité. Le temps moyen pour cette simulation a été de 9.69738 microsecondes et le temps maximal a été de 13.41 microsecondes. Ces temps de calcul sont

légèrement plus élevés que dans le cas précédent et ceci concorde avec la théorie, car dans le cas où une discontinuité est détectée, la charge de travail est considérablement plus élevée que dans le cas précédent, autrement elle demeure sensiblement identique.

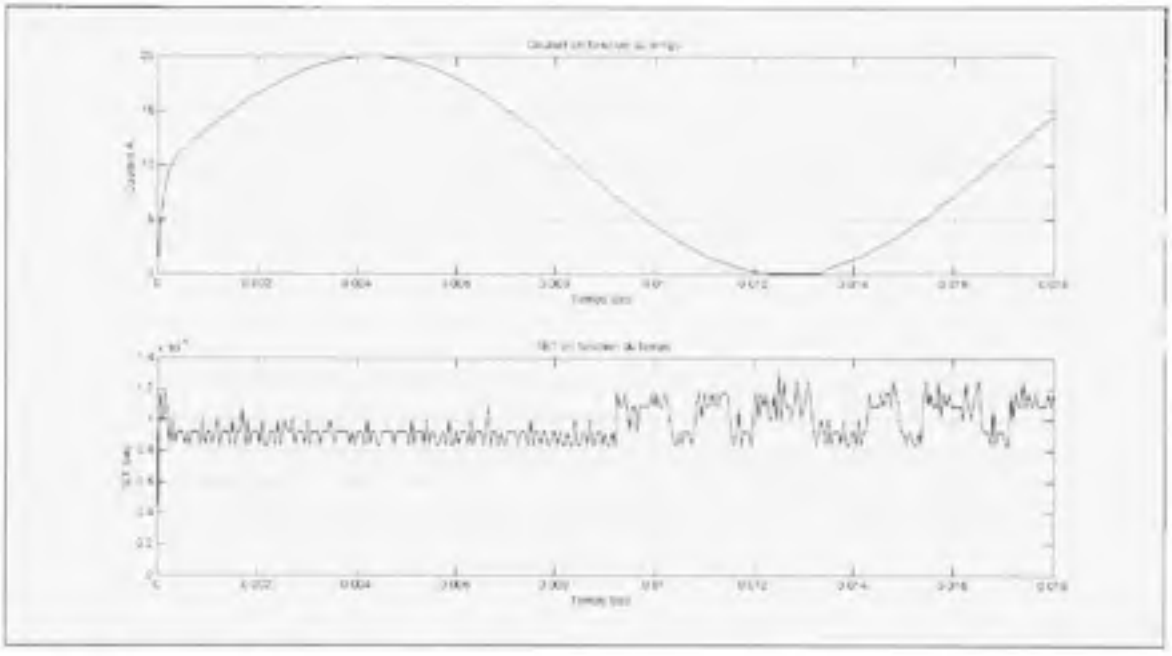

Figure 7.4 Simulation avec la méthode de la commutation précise sous *xPC Target.*

Encore une fois, dans le cas de la double discontinuité, l'algorithme a complètement échoué à localiser ces dernières. Les résultats obtenus pour les temps moyens et maximaux ont été similaires à ceux présentés plus haut soit un temps moyen de 9.47507 microsecondes et un temps maximal de 13.409 microsecondes.

7.3.4 Simulation avec la nouvelle méthode de détection

Finalement, la quatrième et dernière simulation a été effectuée avec la nouvelle méthode de détection. Nous pouvons voir les résultats obtenus sur la Figure 7.5. Pour cette simulation, la discontinuité a été localisée à 0.01248811822175501 seconde. Encore une fois nous gagnons en précision. Par contre, cette précision a un coup en temps de calculs. Le temps moyen pour cette simulation est passé à 10.61211 microsecondes et le temps maximal est

maintenant de 14.248 microsecondes. Dans ce cas aussi il est permis de constater une augmentation du temps de calcul par rapport aux simulations précédentes ce qui est normal. car les calculs des racines sont des opérations très coûteuses en terme de temps de calcul.

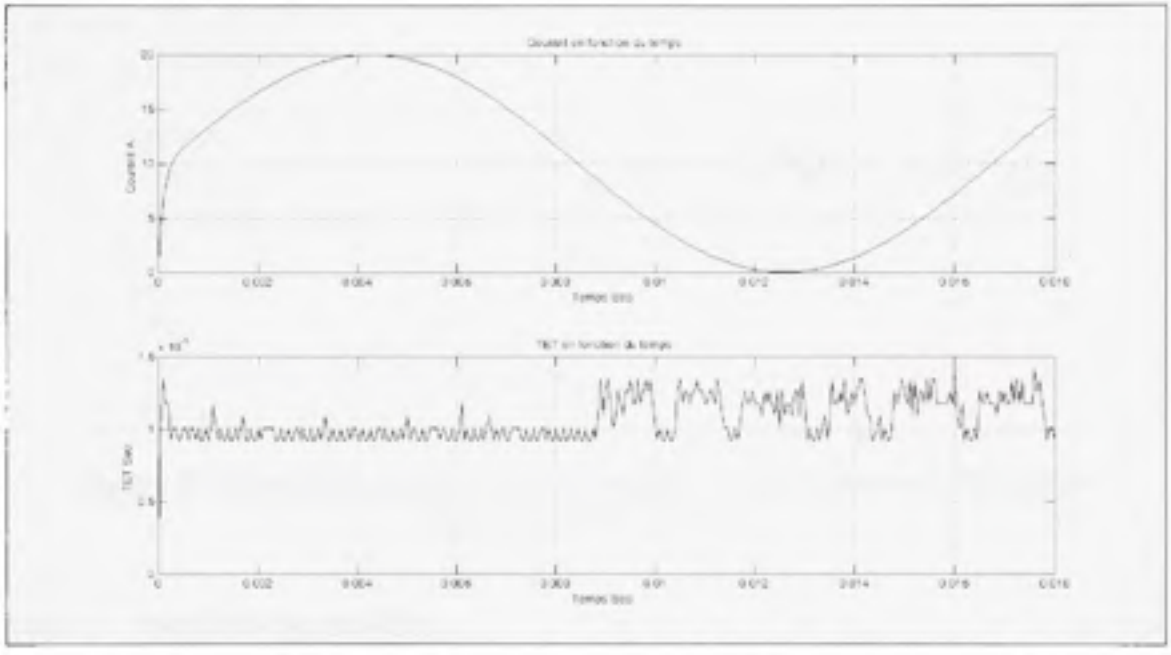

Figure 7.5 Simulation avec la nouvelle méthode de détection sous *xPC Target* (discontinuité simple).

Dans le cas de la double discontinuité. l'algorithme à été en mesure de les détecter avec une précision remarquable. Les résultats sont illustrés sur la Figure 7.6. En effet, l'algorithme a effectué la localisation de la première des deux discontinuités à 0.01665480373488435 seconde. Le temps moyen pour cette simulation à été de 10.75371 microsecondes et le temps maximal à été de 14.24800 microsecondes.

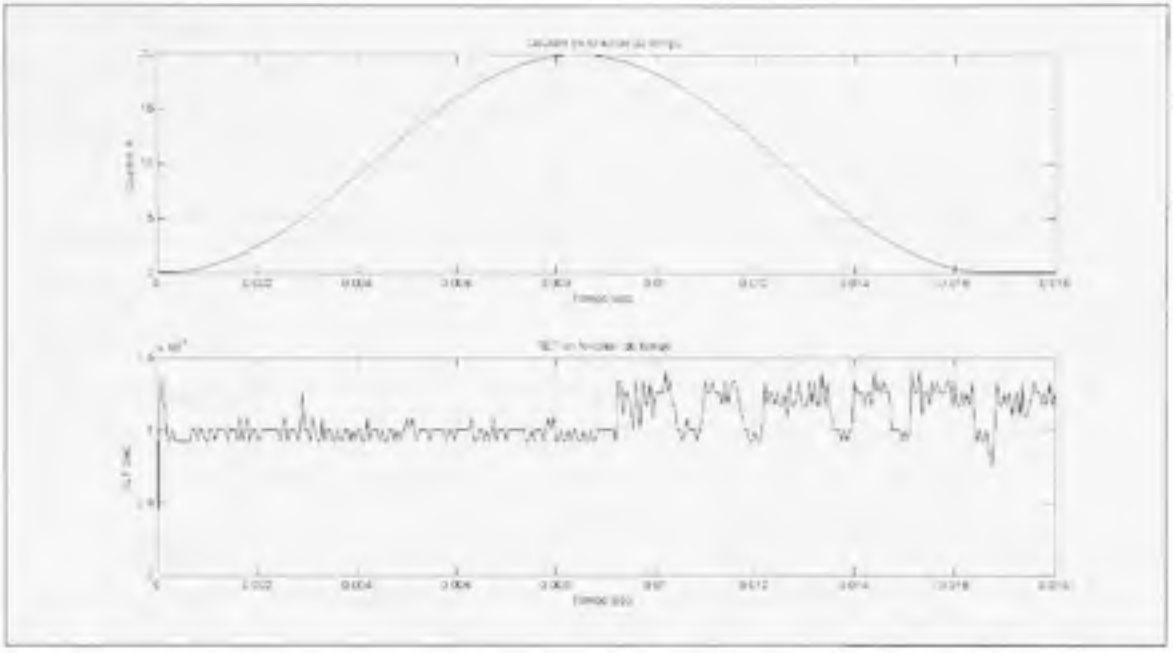

Figure 7.6 Simulation avec la nouvelle méthode de détection sous *xPC Target* (discontinuité double).

7.4 Synthèse des résultats

Les résultats obtenus lors de ces 7 simulations ont été résumés dans le Tableau 7.1 pour la localisation d'une discontinuité simple, dans le Tableau 7.2 pour une discontinuité double et dans le Tableau 7.3 pour le TET des simulations à discontinuité simple (le TET pour les simulations à discontinuité double a été volontairement omis). Pour les trois premières simulations, les résultats concordent parfaitement avec la théorie. La localisation des discontinuités est meilleure en fonction de la méthode utilisée et les temps de simulation augmentent en conséquence. Par exemple, la commutation précise localise plus précisément une discontinuité (différence de 9.048117×10^{-6} secondes avec la localisation réelle de la discontinuité) que le simple fait de ne pas faire de détection (différence 1.186271×10^{-5} secondes avec la localisation réelle de la discontinuité) et le temps moyen augmente de $0,26939 \times 10^{-6}$ secondes soit 0.53872 % du pas de simulation. Pour ce qui est de la nouvelle méthode, les résultats sont encore plus intéressants, car la différence n'est que de 1.905930×10^{-8} secondes et le TET moyen augmente uniquement de $0,91473 \times 10^{-6}$ secondes. De plus, la

nouvelle méthode a été en mesure de détecter une discontinuité double avec une précision très intéressante (une différence d'uniquement 6.7665047×10^{-7} secondes avec la localisation réelle de la première discontinuité).

À la lumière de ces résultats, il est permis de dire que le nouvel algorithme est une option très viable pour effectuer des simulations temps réel. En effet, la surcharge de travail reste très faible et la précision de la localisation est grandement améliorée. De plus, le fait de pouvoir localiser des discontinuités doubles contrebalance l'effet négatif de cette surcharge.

Tableau 7.1 Résumé des résultats pour la localisation de la discontinuité simple

Méthode	**Localisation (sec)**	**Différence (sec)**	**% pas de calcul**
S.M.	0,012500000	1.186271×10^{-5}	23.7254
C.P.	0.0124971853982405	9.048117×10^{-6}	18.0962
N.M.	0.012488118221755	1.905930×10^{-8}	0.038119

Tableau 7.2 Résumé des résultats pour la localisation de la discontinuité double

Méthode	**Localisation (sec)**	**Différence (sec)**	**% pas de calcul**
S.M.	-	-	-
C.P.	-	-	-
N.M.	0. 01665480373488435	6.7665047×10^{-7}	1.353

Tableau 7.3 Résumé des résultats pour les TET

	T. moyen (sec)	**ΔT. moyen (sec)**	**T. maximal (sec)**	**ΔT. maximal (sec)**
S.D.	9.06579×10^{-6}	-	10.058×10^{-6}	-
S.M.	9.42799×10^{-6}	0.3622×10^{-6}	12.572×10^{-6}	2.514×10^{-6}
C.P.	9.69738×10^{-6}	$0,26939 \times 10^{-6}$	$13,41 \times 10^{-6}$	$0,838 \times 10^{-6}$
N.M.	$10,61211 \times 10^{-6}$	$0,91473 \times 10^{-6}$	$14,248 \times 10^{-6}$	$0,838 \times 10^{-6}$

CONCLUSION

Dans une HILS, trois problèmes majeurs sont introduits sur lesquels nous avons un contrôle plus ou moins important. Le premier est relié à la précision des méthodes de résolution d'équations différentielles, le second concerne les retards de commutation et le troisième est introduit lorsqu'un nombre pair de discontinuités se produit entre deux pas de simulation.

Au niveau de la précision des méthodes de résolution, nous avons une influence très limitée. Par contre, sur les deux autres problèmes, nous pouvons grandement influencer les effets qu'ils peuvent avoir sur les résultats d'une simulation. En effet, en localisant convenablement une discontinuité et en s'assurant que toutes les discontinuités sont détectées, les erreurs introduites par ces deux problèmes sont gardées sous un seuil acceptable.

Jusqu'à présent, deux familles d'algorithmes étaient utilisées pour pallier à ces problèmes. Le premier type était les algorithmes aux pas variables introduits notamment par Osterby (Osterby, 1984) et Brankin (Brankin, 1991) et le second type était les algorithmes aux pas fixes introduits, entre autres, par Kuffel, Kent et Irwin (Kuffel, 1997).

Dans le premier groupe, deux problèmes majeurs persistaient. En premier, le fait de faire varier le pas de simulation faisait aussi varier le temps nécessaire pour effectuer chaque pas de simulation. Or dans une simulation temps réel, il faut impérativement tenir ce temps sous contrôle afin d'éviter tout débordement. De plus, dans une HILS, les circuits externes utilisés pour effectuer la simulation sont grandement influencés par la grandeur de ce pas. Il est donc important de le garder fixe.

Pour le deuxième groupe, bien qu'ils aient l'avantage d'avoir un pas de simulation fixe, ils n'étaient pas sans problème. En effet, les deux complications les plus récurrentes pour ces méthodes étaient le fait que le point suivant une discontinuité devait impérativement être

calculé afin de pouvoir détecter cette dernière et puisque la méthode de détection utilisée était très rudimentaire, le nombre de discontinuité détectable était limité à une.

La question suivante a donc été posée : Est-il possible d'augmenter l'efficacité et la précision des algorithmes de localisation des discontinuités afin de palier aux problèmes typiques des méthodes actuelles et d'améliorer la qualité des simulations temps réel des systèmes hybrides?

Ce document a donc tenté de répondre à cette question en proposant un nouvel algorithme basé sur une méthode développée pour la simulation de systèmes robotiques qui utilisaient la théorie du contrôle des systèmes non linéaire. Cette méthode avait comme avantage, d'un point de vue théorique, de détecter à l'avance les discontinuités avant même le calcul du pas suivant cette dernière. De plus, si plusieurs discontinuités se trouvaient dans le même pas, l'algorithme avait la capacité de toutes les détecter. En fait, le nombre de discontinuité que pouvait localiser la méthode est fonction de l'ordre de cette dernière. Par contre, dans cette forme, l'algorithme de localisation était fonction de la méthode numérique de résolution des EDO utilisée pour effectuer la simulation ce qui était un problème majeur. Une seconde approche, basée sur la même méthode, a donc été proposée et cette dernière était complètement indépendante de la méthode de résolution utilisée pour effectuer la simulation.

Des tests ont permis de voir qu'avec ce nouvel algorithme de localisation la précision a grandement augmenté. Pour un circuit de base, simulé avec un pas de 50 microsecondes, l'erreur dans la localisation était passée de 18.1 % du pas de calcul pour la commutation précise à 0.0391 % pour le nouvel algorithme. Pour obtenir ces résultats, le temps d'exécution avait augmenté de seulement $0,91473 \times 10^{-6}$ seconde. De plus, des tests ont été effectués pour vérifier la capacité du nouvel algorithme à localiser des discontinuités multiples dans un même pas de simulation. Le pourcentage d'erreur dans la localisation était situé entre 0,009 % et 1.353 %. Finalement, tous ces tests ont été effectués à l'aide de différentes méthodes numériques de résolution des EDO utilisées pour la simulation et les

résultats sont restés inchangés. Que ce soit pour une discontinuité simple ou double, le pourcentage d'erreur est resté en dessous de 1.4 %.

Suite à l'analyse de ces résultats, il est possible de répondre par l'affirmative à la question préalablement posée. Un nouvel algorithme de localisation des discontinuités a été proposé et celui-ci permet une bien meilleure précision dans la localisation, sans pour autant augmenter de façon drastique les temps d'exécution de chaque pas. De plus, ce nouvel algorithme permet la localisation de plusieurs discontinuités dans un même pas et est complètement indépendant de la méthode de résolution utilisée pour effectuer la simulation proprement dite.

RECOMMANDATIONS ET TRAVAUX FUTURS

À la lumière de ces conclusions, il est clair que le nouvel algorithme à de bonnes bases pour être un très bon choix pour une simulation temps réel de système hybride. Les temps de calcul sont très intéressants et la localisation est grandement améliorée lorsque cet algorithme est utilisé. Ceci étant dit, ce document est plus proche d'une preuve de concept que d'une mise au point définitive. Plusieurs points qui devraient être étudiés afin dans faire une méthode générique pour la simulation temps réel de système hybride. Certains de ces points seront présentés dans ce qui suit.

Dans un premier temps, au niveau de la validation, les tests ont été effectués avec un seul pas de calcul. Or qu'arrive-t-il lorsque le pas de simulation diminue ou augmente ? Il serait très intéressant d'étudier l'impact de la taille du pas de calcul sur la précision des résultats en le faisant varier pour un même circuit. De cette manière, il serait possible d'établir une règle de sélection d'un pas de calcul optimal pour une simulation donné. De plus, il serait intéressant d'étudier l'impact de la position de la discontinuité dans un pas de simulation sur l'efficacité de l'algorithme. Il a déjà été démontré qu'avec des méthodes rudimentaires, cette position pouvait grandement influencer les résultats. En est-il de même avec la nouvelle méthode ? Est-ce que la recherche des racines dans un polynôme est influencée par la proximité des racines aux points décrivant ce dernier ?

Ensuite, dans ce document, un seul circuit a été utilisé pour effectuer les tests. Ce circuit contenait un seul état et un nombre limité d'interrupteurs (les deux diodes). Une validation sur une plus grande gamme de circuits devrait être effectuée ainsi qu'une étude de l'impact du nombre d'états et d'interrupteurs du circuit sur la précision des résultats. Pour ce faire, des circuits d'électronique de puissance contenant un nombre élevé d'interrupteurs, mais un nombre limité d'états pourraient être utilisés dans un premier temps. Ensuite, des circuits d'entrainements électriques contenants un nombre moyen d'interrupteurs, mais un grand nombre d'états devraient être utilisés dans un second temps. Ainsi, avec le circuit utilisé dans ce document et les deux nouveaux, une comparaison pourrait être effectuée et des

conclusions pourraient être tirées quant à l'influence du nombre d'états et d'interrupteurs sur la précision de l'algorithme.

Aussi, étant donné que la fonction gardienne ne dépend pas de la topologie du circuit, cette dernière reste invariable. Il pourrait donc être intéressant d'implémenter ces calculs dans une unité de calcul externe très puissante, autre que l'ordinateur utilisé pour effectuer la simulation. Un circuit intégré de type FPGA pourrait être utilisé à cette fin. Avec ce type de circuit, il serait possible de tirer profit des performances accrues dans le calcul de la fonction gardienne et même de la possibilité d'effectuer des traitements concurrents. Par contre, pour ce faire, une méthode efficace d'échange de données entre le circuit externe et le simulateur devra être mise au point. Cette méthode devra permettre au simulateur de mettre à jour l'état des interrupteurs dans le circuit externe et de permettre à ce dernier de transmettre la localisation d'une discontinuité au simulateur.

Dans un ordre d'idée complètement différent, il y a un point important de la simulation temps réel des systèmes hybrides qui n'a pas été touché dans ce document et qui mérite notre attention. En aucun cas il n'a été question des commutations simultanées. Ce sujet a été évité, car il n'était pas directement relié au sujet présenté dans ce document. Par contre, un bon algorithme de localisation doit tenir compte de ce phénomène qui survient lorsqu'une discontinuité en déclenche une autre qui en déclenche une autre et ainsi de suite. Des travaux devraient être effectués pour déterminer si l'algorithme pourrait aider à obtenir une solution robuste à ce problème.

Finalement, les bases du nouvel algorithme développé dans ce document sont tirées d'un article traitant de la robotique. Il est donc permis de penser que cet algorithme pourrait être utilisé dans d'autres domaines que celui de la simulation des systèmes électriques contenant des interrupteurs. Il serait intéressant de considérer son utilisation dans d'autres domaines d'application tels que celui de la mécanique ou celui de l'hydraulique. De plus, plusieurs systèmes physiques complexes décrits par des équations algébriques différentielles (EAD)

pourraient être simulés à l'aide de cette nouvelle méthode. Bref, une étude plus approfondie de l'application de l'algorithme pourrait être bénéfique.

Dans le même ordre d'idée, il pourrait être intéressant d'adapter le nouvel algorithme pour aider à la résolution de boucles algébriques discontinues. Nous avons une boucle algébrique lorsque la sortie d'un système $y(t)$ dépend directement de l'entrée ($u(t)$), des sorties aux temps précédents ($y(t-1)$, $y(t-2)$, $y(t-3)$, etc...) et de la sortie elle-même au temps présent ($y(t)$). En effet, il arrive que la modélisation d'un système dynamique contienne de telles boucles algébriques et celles-ci doivent être résolues par un travail itératif couteux retarde considérablement la simulation. Les choses se compliquent d'autant plus lorsque la boucle contient des discontinuités, puisque celles-ci perturbent le travail itératif dans sa recherche d'une solution à la boucle algébrique. Il serait donc intéressant de repenser le nouvel algorithme afin de l'adapter à ce travail itératif et de permettre une localisation plus rapide et plus précise des solutions qui déclenchent des discontinuités.

ANNEXE I

Démonstrations pour les méthodes Adams-Bashforth d'ordre supérieur

Dans cette annexe, les équations pour les ordres supérieurs de la méthode Adams-Bashforth seront présentées. Dans chaque cas, la formule complète sera présentée et par la suite, les termes β_i seront développés. Avant de continuer, il est important de stipuler que durant la simulation, le pas h reste constant. Il est vrai que nous cherchons un pas h_{k+1} différent de h mais celui-ci ne sera jamais utilisé pour faire évoluer la simulation. Il sera utilisé uniquement pour localiser les discontinuités. Donc, dans les équations suivantes, h_k (ainsi que tous les pas précédents) est toujours égal à h.

Les équations qui suivent sont le résultat obtenu pour la méthode Adams-Bashforth d'ordre 3.

$$x(t_{k+1}) = x(t_k + h_{k+1}) = x(t_k) + h_{k+1}\left[\beta_1 f_k + \beta_2 f_{k-1} + \beta_3 f_{k-2}\right] \tag{I-1}$$

$$\beta_1 = 1 + \frac{h_{k+1}}{2h_k} + \frac{h_{k+1}^2}{6h_k^2} \tag{I-2}$$

$$\beta_2 = \frac{-h_{k+1}}{2h_k} + \frac{-2h_{k+1}^2}{6h_k^2} \tag{I-3}$$

$$\beta_3 = \frac{h_{k+1}^2}{6h_k^2} \tag{I-4}$$

Les équations qui suivent sont le résultat obtenu pour la méthode Adams-Bashforth d'ordre 4.

$$x(t_{k+1}) = x(t_k + h_{k+1}) = x(t_k) + h_{k+1}\left[\beta_1 f_k + \beta_2 f_{k-1} + \beta_3 f_{k-2} + \beta_4 f_{k-3}\right] \tag{I-5}$$

$$\beta_1 = 1 + \frac{h_{k+1}}{2h_k} + \frac{h_{k+1}^2}{6h_k^2} + \frac{h_{k+1}^3}{24h_k^3} \tag{I-6}$$

$$\beta_2 = \frac{-h_{k+1}}{2h_k} + \frac{-2h_{k+1}^2}{6h_k^2} + \frac{-3h_{k+1}^3}{24h_k^3} \tag{I-7}$$

$$\beta_3 = \frac{h_{k+1}^2}{6h_k^2} + \frac{3h_{k+1}^3}{24h_k^3} \tag{I-8}$$

$$\beta_4 = \frac{-h_{k+1}^3}{24h_k^3} \tag{I-9}$$

Les équations qui suivent sont le résultat obtenu pour la méthode Adams-Bashforth d'ordre 5.

$$x(t_{k+1}) = x(t_k + h_{k+1}) = x(t_k) + h_{k+1}\left[\beta_1 f_k + \beta_2 f_{k-1} + \beta_3 f_{k-2} + \beta_4 f_{k-3} + \beta_5 f_{k-4}\right] \tag{I-10}$$

$$\beta_1 = 1 + \frac{h_{k+1}}{2h_k} + \frac{h_{k+1}^2}{6h_k^2} + \frac{h_{k+1}^3}{24h_k^3} + \frac{h_{k+1}^4}{120h_k^4} \tag{I-11}$$

$$\beta_2 = \frac{-h_{k+1}}{2h_k} + \frac{-2h_{k+1}^2}{6h_k^2} + \frac{-3h_{k+1}^3}{24h_k^3} + \frac{-4h_{k+1}^4}{120h_k^4} \tag{I-12}$$

$$\beta_3 = \frac{h_{k+1}^2}{6h_k^2} + \frac{3h_{k+1}^3}{24h_k^3} + \frac{6h_{k+1}^4}{120h_k^4} \tag{I-13}$$

$$\beta_4 = \frac{-h_{k+1}^3}{24h_k^3} + \frac{-4h_{k+1}^4}{120h_k^4} \tag{I-14}$$

$$\beta_5 = \frac{h_{k+1}^4}{120h_k^4} \tag{I-15}$$

Le polynôme pour chacun de ces ordres est formé de la même manière qu'avec l'ordre 2. Premièrement nous substituons $\overline{g}_\beta$ dans l'équation 3.31 et nous formons le polynôme en regroupant tous les termes du même degré ensemble.

En effectuant ces étapes nous obtenons les équations qui suivent pour la méthode Adams-Bashforth d'ordre 3.

$$ah_{k+1}^3 + bh_{k+1}^2 + ch_{k+1} + d = 0 \tag{I-16}$$

Avec

$$a = \frac{\dot{g}(x(t_k))}{6h_k^2}(f_k - 2f_{k-1} + f_{k-2}) \tag{I-17}$$

$$b = \frac{\dot{g}(x(t_k))}{2h_k}(f_k - f_{k-1}) \tag{I-18}$$

$$c = \dot{g}(x(t_k))f_k \tag{I-19}$$

$$d = g(x(t_k)) \tag{I-20}$$

Les équations qui suivent sont le résultat de ces étapes pour la méthode Adams-Bashforth d'ordre 4.

$$ah_{k+1}^4 + bh_{k+1}^3 + ch_{k+1}^2 + dh_{k+1} + e = 0 \tag{I-21}$$

Avec

$$a = \frac{\dot{g}(x(t_k))}{24h_k^3}(f_k - 3f_{k-1} + 3f_{k-2} - f_{k-3}) \tag{I-22}$$

$$b = \frac{\dot{g}(x(t_k))}{6h_k^2}(f_k - 2f_{k-1} + f_{k-2}) \tag{I-23}$$

$$c = \frac{\dot{g}(x(t_k))}{2h_k}(f_k - f_{k-1}) \tag{I-24}$$

$$d = \dot{g}(x(t_k))f_k \tag{I-25}$$

$$e = g(x(t_k)) \tag{I-26}$$

Finalement, les équations qui suivent sont le résultat de ces étapes pour la méthode Adams-Bashforth d'ordre 5.

$$ah_{k+1}^5 + bh_{k+1}^4 + ch_{k+1}^3 + dh_{k+1}^2 + eh_{k+1} + f = 0 \tag{I-27}$$

Avec

$$a = \frac{\dot{g}(x(t_k))}{120h_k^4}(f_k - 4f_{k-1} + 6f_{k-2} - 4f_{k-3} + f_{k-4}) \tag{I-280}$$

$$b = \frac{\dot{g}(x(t_k))}{24h_k^3}(f_k - 3f_{k-1} + 3f_{k-2} - f_{k-3}) \tag{I-29}$$

$$c = \frac{\dot{g}(x(t_k))}{6h_k^2}(f_k - 2f_{k-1} + f_{k-2}) \tag{I-30}$$

$$d = \frac{\dot{g}(x(t_k))}{2h_k}(f_k - f_{k-1}) \tag{I-31}$$

$$e = \dot{g}(x(t_k))f_k \tag{I-32}$$

$$f = g(x(t_k)) \tag{I-33}$$

ANNEXE II

Code source des applications modifiées *xPC Target*

Dans cette annexe le code source des programmes *xPC Target* utilisés pour effectuer les simulations temps réel sera présenté. La totalité des fichiers ne sera pas présentée étant donné que ce qui nous intéresse est uniquement les modifications effectuées pour obtenir les algorithmes. Il sera donc, pour chaque cas, présenté uniquement les fonctions qui ont été modifiées.

Premièrement, dans le cas de la méthode simple, sans localisation, la seule modification qui à été effectuée est l'évaluation du signal de sortie avant chaque pas. Cette évaluation est donc effectuée dans la fonction `rt_ertODEUpdateContinuousStates(...)` :

```
static void rt_ertODEUpdateContinuousStates(RTWSolverInfo *si )
{
  ...
  if (((mGD1m0 < 0) && (SwD1 == 1)) || ((mGD2m0 > 0) && (SwD2 == 0)))
  {
    CircuitDiodeTest_Y.Out3 = 1;
    SwD1 = 0;
    SwD2 = 1;
    CircuitDiodeTest_P.StateSpace_A = A1;
    CircuitDiodeTest_P.StateSpace_B = B1;
    mGaD1 = GaD1_1;
    mGbD1 = GbD1_1;
    mGaD2 = GaD2_1;
    mGbD2 = GbD2_1;
  }
  else if (((mGD1m0 > 0) && (SwD1 == 0)) || ((mGD2m0 < 0) && (SwD2 == 1)))
  {
    CircuitDiodeTest_Y.Out3 = 1;
    SwD1 = 1;
    SwD2 = 0;
    CircuitDiodeTest_P.StateSpace_A = A2;
    CircuitDiodeTest_P.StateSpace_B = B2;
    mGaD1 = GaD1_2;
    mGbD1 = GbD1_2;
    mGaD2 = GaD2_2;
    mGbD2 = GbD2_2;
  }
  ...
}
```

Ensuite, pour la commutation précise, c'est encore une fois la fonction qui met à jour les variables d'état qui a été modifiée. Lorsque le pas est calculé et que les sorties sont à jour, la

fonction gardienne est évaluée et si une discontinuité est détectée, les étapes de localisation sont effectuées.

```
static void rt_ertODEUpdateContinuousStates(RTWSolverInfo *si )
{
  CircuitDiodeTest_output(0);
  …
  /** S'il y a un changement de signe dans le garde, il y a eu une discontinuité */
  if((mGD1m0 * mGD1m1 < 0) || (mGD2m0 * mGD2m1 < 0))
  {
    CircuitDiodeTest_Y.Out3 = 1;
    /** Interpolation  pour trouver t(teta)*/
    if(mGD1m0 * mGD1m1 < 0)
    {
      lTeta = (t - h) + h *( (0 - mGD1m1) / (mGD1m0 - mGD1m1));
    }
    else
    {
      lTeta = (t - h) + h *( (0 - mGD2m1) / (mGD2m0 - mGD2m1));
    }

    /** u(teta)  = la limite !! */
    lUTeta = AMP * sin(FREQ * lTeta + PHI) + BIAS;

    /** Interpolation de y(teta) entre y(t) et y(t+h) */
    lYTeta = mYm1 + ( (lTeta - (t - h)) / (h) ) * (mYm0 - mYm1);

    /** Interpolation de x(teta) entre x(t) et x(t+h) */
    lXTeta = mXm1 + ( (lTeta - (t - h)) / (h) ) * (mXm0 - mXm1);

    /** Configurer convenablement les matrices A, B, C et D */
    if (((mGD1m0 < 0)&&(SwD1 == 1))||((mGD2m0 > 0)&&(SwD2 == 0)))
    {
      SwD1 = 0;
      SwD2 = 1;
      CircuitDiodeTest_P.StateSpace_A = A1;
      CircuitDiodeTest_P.StateSpace_B = B1;
      mGaD1 = GaD1_1;
      mGbD1 = GbD1_1;
      mGaD2 = GaD2_1;
      mGbD2 = GbD2_1;
    }
    else if (((mGD1m0 > 0)&&(SwD1 == 0))||((mGD2m0 < 0)&&(SwD2 == 1)))
    {
      SwD1 = 1;
      SwD2 = 0;
      CircuitDiodeTest_P.StateSpace_A = A2;
      CircuitDiodeTest_P.StateSpace_B = B2;
      mGaD1 = GaD1_2;
      mGbD1 = GbD1_2;
      mGaD2 = GaD2_2;
      mGbD2 = GbD2_2;
    }

    /** dx(teta) */
    lDxTeta = CircuitDiodeTest_P.StateSpace_A *
              lXTeta + CircuitDiodeTest_P.StateSpace_B * lUTeta;

    /** x(teta + h) */
    lX = lXTeta + h * lDxTeta;

    /** t(teta + h)*/
    lT = lTeta + h;
```

```
    /** y(teta + h)*/
    lY = CircuitDiodeTest_P.StateSpace_C * lX + CircuitDiodeTest_P.StateSpace_D * lT;

    /** Interpolation pour la synchronisation */
    /** Interpolation de x(t + h) entre x(teta) et x(teta + h) */
    mXm0 = lXTeta + ( ((t)  - lTeta) / (h) ) * (lX - lXTeta);

    for (i = 0; i < nXc; i++)
    {
      y[i] = mXm0; // il y a juste un état
    }

    /** Interpolation de y(t + h) entre y(teta) et y(teta + h) */
    mYm0 = lYTeta + ( ((t)  - lTeta) / (h) ) * (lY - lYTeta);
    CircuitDiodeTest_B.StateSpace = mYm0;
  }
  ...
}
```

Finalement, pour la nouvelle méthode, c'est aussi la méthode qui met à jour les variables d'état qui a été modifiée :

```
static void rt_ertODEUpdateContinuousStates(RTWSolverInfo *si )
{
  ...
  /** Avant de calculer le point au temps t(n+1), il faut rouler l'algo sur t(n) */
  /** Former les coefficients du polynôme */
  a = (mDGD1m0-mDGD1m1);
  b = ( 2*h*mDGD1m0);
  c =  -2*h*(Gamma-1.0)*mGD1m0;
  bCarre = b * b;
  quatreAC = 4 * a * c;
  deuxA = 2 * a;

  /** Voir si nous avons une racine complexe */
  if ((bCarre - quatreAC) < 0)
  {
    D1r1 = h + 1;
    D1r2 = h + 1;
  }
  else
  {
    sq = sqrt(bCarre - quatreAC);
    D1r1 = ((-b + sq) / deuxA);
    D1r2 = ((-b - sq) / deuxA);

    if (D1r1 < 0)
    {
      D1r1 = h + 1;
    }
    if (D1r2 < 0)
    {
      D1r2 = h + 1;
    }
  }

  if (D1r2 < D1r1)
  {
    tpr = D1r1;
```

```
  D1r1 = D1r2;
  D1r2 = tpr;
}

a = (mDGD2m0-mDGD2m1);
b = ( 2*h*mDGD2m0 );
c =  -2*h*(Gamma-1.0)*mGD2m0;

bCarre = b * b;
quatreAC = 4 * a * c;
deuxA = 2 * a;

/** Voir si nous avons une racine complexe */
if ((bCarre - quatreAC) < 0)
{
  D2r1 = h + 1;
  D2r2 = h + 1;
}
else
{
  sq = sqrt(bCarre - quatreAC);
  D2r1 = ((-b + sq) / deuxA);
  D2r2 = ((-b - sq) / deuxA);

  if (D2r1 < 0)
  {
    D2r1 = h + 1;
  }
  if (D2r2 < 0)
  {
    D2r2 = h + 1;
  }
}

if (D2r2 < D2r1)
{
  tpr = D2r1;
  D2r1 = D2r2;
  D2r2 = tpr;
}

if (D1r1 < D2r1)
{
  r1 = D1r1;
}
else
{
  r1 = D2r1;
}


if(t < 0.005)
{
  r1 = h + 1;
  r2 = h + 1;
}

// Est ce qu'il y a une disc entre t(n) et t(n+1)
if (r1 < h)
{
  // Évaluation de  t(teta)
  lTeta = t + r1;

  // Évaluation de x(teta)
  lXTeta = mXm0 + r1 * mDxm0;

  // u(teta)  = la limite !!
  lUTeta = AMP * sin(FREQ * lTeta + PHI) + BIAS;
```

```
    if(gFlag == 0)
    {
      CircuitDiodeTest_Y.Out2 = t;
      gFlag = 0;
    }

    // Configurer convenablement les matrices A, B, C et D
    if (((D1r1 == r1)&&(SwD1 == 1))||((D2r1 == r1)&&(SwD2 == 0)))
    {
      SwD1 = 0;
      SwD2 = 1;
      CircuitDiodeTest_P.StateSpace_A = A1;
      CircuitDiodeTest_P.StateSpace_B = B1;
      mGaD1 = GaD1_1;
      mGbD1 = GbD1_1;
      mGaD2 = GaD2_1;
      mGbD2 = GbD2_1;
    }
    else if (((D1r1 == r1)&&(SwD1 == 0))||((D2r1 == r1)&&(SwD2 == 1)))
    {
      SwD1 = 1;
      SwD2 = 0;
      CircuitDiodeTest_P.StateSpace_A = A2;
      CircuitDiodeTest_P.StateSpace_B = B2;
      mGaD1 = GaD1_2;
      mGbD1 = GbD1_2;
      mGaD2 = GaD2_2;
      mGbD2 = GbD2_2;
    }

    // dx(teta)
    lDxTeta = CircuitDiodeTest_P.StateSpace_A *
              lXTeta + CircuitDiodeTest_P.StateSpace_B * lUTeta;

    // x(n+1)
    mXm0 = lXTeta + (h-r1) * lDxTeta;
    for (i = 0; i < nXc; i++)
    {
      x[i] = mXm0; // il y a juste un état
    }
  }
  else
  {
    // Opération normale, car pas de disc.
    ...

  }
  ...
}
```

BIBLIOGRAPHIE

Brankin, L. F. Shampine and I. Gladwell and R. W. 1991. « Reliable solution of special event location problems for ODEs ». *ACM Trans. Math. Softw.*, vol. 17, n° 1, p. 11--25.

Dinavahi, V.R. and Reza Iravani, M. and Bonert, R. 2001. « Real-time digital simulation of power electronic apparatus interfaced with digital controllers ». *IEEE Transactions on Power Delivery*, vol. 16, n° 4, p. 775-781.

Esposito, Joel M. 2002. « Simulation and control of Hybrid systems with applications to mobile robotics ». University of Pennsylvania.

Esposito, Joel M., Vijay Kumar et George J. Pappas. 2001. « Accurate Event Detection for Simulating Hybrid Systems ». In *Hybrid Systems: Computation and Control : 4th International Workshop, HSCC 2001 Rome, Italy, March 28-30, 2001, Proceedings:* . p. 204. <http://www.springerlink.com/content/4hx8nlq1lwlm6mpb >.

Fortin, André. 2001. *Analyse numérique pour ingénieurs deuxième édition*. Montréal: Presses internationales Polytechnique, 487 p.

Kelper, Bruno De. 2002. « Simulation à pas fixe et en temps réel de circuits électriques contenant des interrupteurs ». Montréal, École de technologie supérieure, 160 p.

Kelper, V.-Q. Do and D. McCallum and P. Giroux and B. De. 2001. « A backward-forward interpolation technique for a precise modeling of power electronic in HYPERSIM ».

Kuffel, P. and Kent, K. and Irwin, G. 1997. « The implementation and effectiveness of linear interpolation within digit ». *International Journal of Electrical Power & Energy Systems*, vol. 19, n° 4, p. 221-227.

Marquez, Horacio J. 2003. *Nonlinear control systems : analysis and design*. Hoboken: J. Wiley, 352 p.

Mathworks, The. 2005. « Hitachi Accelerates the Development of Engine Knock-Reduction Systems ». *The Mathworks User Story*, p. 1-2.

Mathworks, The. 2006. « BAE Systems Achieves 80% Reduction in Software-Defined Radio Development Time with Model-Based Design ». *The Mathworks User Story*, p. 1-2.

Mathworks, The. 2007. « Cessna Enhances Antiskid Technology with Hardware in the Loop Testing ». *The Mathworks User Story*, p. 1-2.

Mathworks, The. 2008. *xPC Target™ Getting Started Guide*. Natick, 192 p.

Nakra, B. De Kelper and L.A. Dessaint and K. Al-Haddad and H. 2002. « A comprehensive approach to fixed-step simulation of switched circuits ». *IEEE Transactions on Power Electronics*, vol. 17, n° 2, p. 216-224.

Osterby, C. W. Gear and O. 1984. « Solving Ordinary Differential Equations with Discontinuities ». *ACM Trans. Math. Softw.*, vol. 10, n° 1, p. 23--44.

Rao, Singiresu S. 2002. *Applied Numerical Methods for Engineers ans Scientists*. Prentice Hall.

Strunz, K. 2004. « Flexible numerical integration for efficient representation of switching in real time electromagnetic transients Simulation ». *IEEE Transactions on Power Delivery*, vol. 19, n° 3, p. 1276-1283.

Tom T. Hartley, Guy O. Beale, Stephen P. Chicatelli. 1994. *Digital Simulation of Dynamic Systems - A control Theory Approach*. Prentice Hall.

Wikipedia. 2009. « Théorème de Sturm ». , En Ligne, <http://fr.wikipedia.org/wiki/Th%C3%A9or%C3%A8me_de_Sturm>.