ÉCOLE DE TECHNOLOGIE SUPÉRIEURE
UNIVERSITÉ DU QUÉBEC

THÈSE DE DOCTORAT PRÉSENTÉE À
L'ÉCOLE DE TECHNOLOGIE SUPÉRIEURE

COMME EXIGENCE PARTIELLE
À L'OBTENTION DU
DOCTORAT EN GÉNIE
PH.D.

PAR
ROGER CHAMPAGNE

SIMULATION EN TEMPS RÉEL À L'AIDE DE LA REPRÉSENTATION D'ÉTAT: APPLICATION À UN ENTRAÎNEMENT ÉLECTRIQUE BASÉ SUR UNE MACHINE ASYNCHRONE

MONTRÉAL, LE 16 JUILLET 2001

CETTE THÈSE A ÉTÉ ÉVALUÉE

PAR UN JURY COMPOSÉ DE:

- M. Louis-A. Dessaint, directeur de thèse
  Département de génie électrique à l'École de technologie supérieure

- Mme Ouassima Akhrif, professeure
  Département de génie électrique à l'École de technologie supérieure

- M. Kamal Al-Haddad, professeur
  Département de génie électrique à l'École de technologie supérieure

- M. Jean Mahseredjian, chercheur
  Institut de recherche d'Hydro-Québec

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

LE 22 JUIN 2001

À L'ÉCOLE DE TECHNOLOGIE SUPÉRIEURE

SIMULATION EN TEMPS RÉEL À L'AIDE DE LA REPRÉSENTATION D'ÉTAT: APPLICATION À UN ENTRAÎNEMENT ÉLECTRIQUE BASÉ SUR UNE MACHINE ASYNCHRONE

Roger Champagne

(Sommaire)

Les machines électriques sont omniprésentes dans nos vies. Il y en a dans nos ordinateurs et appareils ménagers, elles entraînent les machines-outils et les robots dans nos usines et déplacent trains et navires. Suite aux progrès importants en électronique de puissance ces dernières années, les entraînements à vitesse variable ont aussi connu une popularité croissante. Cependant, leur utilisation à grande échelle pollue le réseau électrique avec des harmoniques indésirables qui troublent le fonctionnement d'équipements sensibles, tels les ordinateurs et les systèmes de télécommunications. L'impact des entraînements à vitesse variable sur le réseau électrique qui les alimente doit donc être analysé à l'aide d'outils de simulation. De plus, la conception des entraînements de grande puissance bénéficierait aussi d'un outil permettant de développer les prototypes des contrôleurs associés à ces entraînements.

Le but de cette thèse est donc de développer l'outil en question, soit un simulateur d'entraînements électriques entièrement numérique en temps réel. Ce simulateur permettrait aux ingénieurs chargés de la conception des entraînements d'effectuer des batteries de tests sur un prototype de contrôleur, sans avoir besoin dès le départ du véritable convertisseur et de la véritable machine. Ces premiers essais pourraient donc se faire dans des installations beaucoup plus modestes et de façon plus sécuritaire.

Notre travail est basé sur la modélisation des entraînements électriques à l'aide de l'approche par variables d'état. Nous décrivons d'abord une méthode permettant d'obtenir automatiquement les équations d'état de tout système électrique linéaire, les composantes non-linéaires étant simulées à l'extérieur de la représentation d'état. La méthode est basée sur la théorie des graphes linéaires et comporte beaucoup de calcul matriciel, lequel est réalisé efficacement dans l'environnement Matlab. Une technique originale de mise à jour des équations d'état suite à un changement d'état d'interrupteurs est utilisée, ainsi qu'une méthode permettant d'obtenir une représentation d'état unique de tout l'étage de puissance. Cette dernière méthode permet une solution simultanée de toutes les équations dynamiques du système. Une implantation de la discrétisation trapézoïdale adaptée aux systèmes variant dans le temps est ensuite décrite et comparée à une méthode

d'intégration récemment développée pour la simulation en temps réel des systèmes rigides. Enfin, les diverses techniques exposées sont implantées afin de permettre la simulation en temps réel d'un entraînement industriel sur un ordinateur parallèle. D'excellents résultats sont obtenus avec un pas de calcul de l'ordre de 60 µs, incluant les communications interprocesseur et les acquisitions des entrées et sorties.

ABSTRACT

Electric machinery is widely used in our daily lives. There are electric motors in our computers and appliances at home, in tools and robots in manufacturing plants, and in vehicles such as cars, trains and ships. The past few decades have seen major advancements in the field of variable speed drives, consequent to new technologies in power electronics. However, widespread use of such drives pollutes the power grid with undesirable harmonics which compromise the normal operation of sensitive devices such as computers and telecommunication systems. The impact of the variable speed drives on the power grid must therefore be analyzed with simulation tools. Moreover, designing high power drives could also benefit from a simulation tool that would allow prototyping of the drive's controller.

Our goal in this thesis is to develop such a tool, a fully digital real-time simulator dedicated to electric drives. Such a simulator would allow engineers responsible for the design of large drives to prototype the drive's controller and initially test it using a simulated power converter and machine. Such tests would require limited space and equipment, and could be carried out safely without any high power devices.

Our work is based on modeling the drive using the state variable approach. We first describe a method which automatically computes the state equations of any linear electric system, nonlinear components being simulated outside the state-space representation. The method is based on linear graph theory and uses matrix computations extensively. This task is performed efficiently in the Matlab environment. An original technique, used to update the state equations when a switching device changes state, is then described. We then describe a method which yields a unique state-space representation for the entire power stage of a drive. This method allows a simultaneous delay-free solution of all the drive's dynamic equations. An implementation of the trapezoidal integration rule, adapted for time-varying systems, is then described and compared to an integration method recently developed specifically for real-time simulation of stiff systems. Finally, the various methods discussed thus far are implemented in order to yield a real-time simulation of an industrial drive on a parallel computer. Good results are obtained with timesteps of the order of 60 μs, which includes interprocessor communications and inputs and outputs acquisition and conversion.

AVANT-PROPOS

Le travail décrit dans cette thèse est le fruit d'une collaboration entre le G.R.É.P.C.I., le Laboratoire de Simulation de Réseaux (LSR) de l'IREQ, et TransÉnergie Technologies. Plus spécifiquement, nos travaux ont été réalisés dans le cadre de deux subventions industrie-université du CRSNG. Le G.R.É.P.C.I. et ses partenaires industriels collaborent dans différents projets de recherche liés à la simulation des réseaux électriques.

J'aimerais d'abord remercier mon directeur de thèse, le professeur Louis-A. Dessaint, avec qui je travaille depuis plus de six ans maintenant. Le dynamisme et la grande disponibilité du professeur Dessaint sont des éléments clés dans le succès de mes études de maîtrise et de doctorat. Il a toujours su bien m'encadrer, tout en me donnant beaucoup de latitude pour exprimer mes initiatives.

J'aimerais également remercier les autres membres du jury, les professeurs Ouassima Akhrif et Kamal Al-Haddad, ainsi que Jean Mahseredjian, chercheur à l'IREQ. Les nombreuses discussions que j'ai eues avec eux, autant durant les cours qu'ils m'ont donnés que durant mes travaux de recherche, ont eu un impact appréciable sur mon travail. Ils ont également tous fait preuve d'une grande disponibilité à mon égard.

Je veux également remercier tous les gens qui travaillent chez nos partenaires industriels, et avec qui j'ai eu l'occasion d'échanger, notamment l'équipe du LSR de l'IREQ et l'équipe chez TransÉnergie Technologies. Pierre Mercier, ancien chargé de techniques de simulation à l'IREQ, est le principal instigateur de la collaboration continue entre le G.R.É.P.C.I. et le LSR et a été d'un support inconditionnel tout au long de mes travaux. Gilbert Sybille, chercheur au LSR et auteur principal du Power System Blockset (PSB), a généreusement partagé avec moi ses connaissances liées aux divers aspects de mes travaux. Je tiens également à remercier Michel Tétreault, chef du LSR, ainsi que Jean-Claude Soumagne et Silvano Casoria, tous deux chercheurs au LSR, qui ont supporté ces travaux par l'entremise des subventions sus-mentionnées. Je remercie également toute l'équipe du LSR pour avoir consenti à m'accueillir et pour m'avoir supporté durant l'été 2000. J'y ai passé plusieurs semaines à expérimenter avec leur ordinateur SGI et je tiens à reconnaître les contributions de Pierre Giroux, Van Qué Do et Christian Larose, chercheurs au LSR, ainsi que de Josée Plante, administratrice du réseau informatique, et plus particulièrement Sylvain Guérette, également du LSR. Sylvain m'a aidé à comprendre beaucoup de choses concernant les ordinateurs à mémoire partagée et son expérience m'a permis de sauver beaucoup de temps.

Chez TransÉnergie Technologies, je tiens à remercier Yves Carmel, ancien directeur général, ainsi que Alain Vallée, directeur général, et Bahram Khodabakhchian, ingénieur, pour leur collaboration. Merci également à Patrice Brunelle, responsable technique du développement du PSB, pour son ouverture d'esprit face à mes idées et suggestions,

ainsi que pour sa capacité à adapter rapidement le PSB à des exigences sans cesse changeantes.

Le développement d'outils de simulation est un sujet fort intéressant, mais peu utile sans usager. À cet effet, je tiens à remercier particulièrement mes collègues et amis du G.R.É.P.C.I., autant étudiants que professeurs, pour leur grande patience avec toutes les versions intermédiaires du PSB, et pour leurs nombreux commentaires et suggestions. J'ai eu plusieurs discussions fort enrichissantes avec bon nombre d'entre eux et j'ai été en mesure d'apprécier pleinement, grâce à eux, la force du travail en groupe.

Des études de doctorat se complètent difficilement sans support moral. Outre les personnes déjà mentionnées, je tiens à adresser mes remerciements les plus sincères à ma famille et à mes amis, qui ont enduré mon absence, ou ma présence a temps partiel, durant ces années et ce, tout en me soutenant dans mes efforts avec quantité d'encouragements.

Enfin, s'il y a une personne qui mérite son propre paragraphe dans ces remerciements, c'est ma copine Julie. Nos chemins se sont croisés quelque part au début de ma maîtrise, et elle est toujours là, ce qui en dit long en soi. Merci pour ta grande patience et tes encouragements soutenus. J'espère un jour pouvoir te rendre ça.

TABLE DES MATIÈRES

Page

LISTE DES TABLEAUX

Page

LISTE DES FIGURES

Page

LISTE DES ABRÉVIATIONS ET DES SIGLES

G.R.É.P.C.I.	Groupe de recherche en électronique de puissance et commande industrielle
IREQ	Institut de recherche d'Hydro-Québec
CRSNG	Conseil de recherches en sciences naturelles et en génie du Canada
PSB	Power System Blockset
PI	Proportionnel et intégrale
EMTP	Electromagnetic Transients Program
dq0	axes du référentiel dq arbitraire (d=direct, q=quadrature, 0=homopolaire)
t	temps, s indice indiquant une composante appartenant à l'arbre (branche)
l	indice indiquant une composante n'appartenant pas à l'arbre (lien)
$\frac{dx}{dt}$, $\dot{x}$	dérivée de la variable x par rapport au temps
x	vecteur des variables d'état
u	vecteur des entrées
y	vecteur des sorties
A	matrice des paramètres du système
B	matrice de couplage des entrées
C	matrice de couplage des sorties
D	matrice liant les entrées aux sorties

B_f	matrice des mailles fondamentales
Q_f	matrice des coupures fondamentales
i ou I	courant électrique, A. Indices utilisés:
	a,b,c grandeur de phase a, b ou c (référentiel fixe)
	d,q grandeur statorique d'axe d ou q (référentiel dq)
	m grandeur mécanique ou magnétisante
	e grandeur électrique
	n grandeur nominale
	b grandeur de base
	l grandeur de fuite
	s grandeur statorique
	r grandeur rotorique
v ou V	tension, V (mêmes indices que courant électrique)
E	source de tension, V
J	source de courant, A
RLC	acronyme indiquant un circuit passif contenant exclusivement les deux types de sources idéales ainsi que des résistances, inductances et condensateurs
S	matrice liant les variables d'état aux tensions et courants des éléments résistifs
T	matrice liant les entrées aux tensions et courants des éléments résistifs
P	matrice des chemins ("path matrix"). puissance active, W sous-matrice de la matrice d'état C

G	matrice de conductances (circuits sans transformateurs) matrice liant les tensions aux courants des éléments résistifs (circuits avec transformateur)
D, M	matrices intermédiaires utilisées dans le calcul des matrices d'état C et D
V	matrice liant les variables d'état aux tensions des branches inductives
W	matrice liant les entrées aux tensions des branches inductives
λ	flux magnétique, V.s (mêmes indices que courant électrique) valeurs propres d'une matrice
r ou R	résistance, Ω (mêmes indices que courant électrique)
L	inductance, H (mêmes indices que courant électrique)
f	fréquence, Hz
p	nombre de paires de pôles de la machine
n ou N	vitesse de rotation, r/min ou rad/s
ω	vitesse angulaire électrique du référentiel dq0, rad/s
θ	position angulaire électrique du référentiel dq0, degrés
ω_e	vitesse angulaire du champ tournant statorique, rad/s
θ_e	position angulaire du champ tournant statorique, degrés
ω_r	vitesse angulaire électrique du rotor, rad/s
θ_r	position angulaire électrique du rotor, degrés
ω_m	vitesse angulaire mécanique du rotor, rad/s
θ_m	position angulaire mécanique du rotor, degrés
K	matrice de transformation de référentiel. Indices utilisés: s: transformation appliquée à des circuits stationnaires

r: transformation appliquée à des circuits en rotation

M, N	matrices de transformation de référentiel
Q_1, Q_2	sous-matrices de la matrice d'état D
I_3	matrice identité 3 par 3
J	moment d'inertie, $kg.m^2$ matrice d'état A ou sa linéarisation
T	période d'échantillonnage
T_m	couple mécanique, N.m ou p.u.
T_e	couple électrique, N.m ou p.u.
F	coefficient de frottement visqueux, N.m.s ou p.u.
x'	variable x est une grandeur rotorique référée au stator
A_0, A_1, B_0, B_1	matrices des coefficients d'intégration de la méthode MSRP-2
UAL	unité arithmétique et logique

INTRODUCTION

Les machines électriques sont omniprésentes dans nos vies. De celles qui se trouvent dans nos ordinateurs et appareils ménagers aux moteurs dans les trains et navires en passant par ceux qui entraînent les convoyeurs, pompes, machines-outils et robots dans les usines, les machines sont partout. Les entraînements à fréquence variable ont connu ces dernières décennies une croissance étonnante, conséquence directe de progrès notables dans le domaine de l'électronique de puissance.

Cependant, l'utilisation à grande échelle d'entraînements à fréquence variable cause certains problèmes au niveau du réseau électrique. Les convertisseurs à électronique de puissance de ces entraînements génèrent des harmoniques qui polluent le réseau et perturbent le fonctionnement d'appareils sensibles tels les ordinateurs et équipements de télécommunication. Il est donc important de pouvoir étudier et analyser l'impact de ces entraînements dans un contexte donné et un bon outil de simulation devrait faciliter cette analyse. Ceci est d'autant plus important que les machines électriques représentent une part importante de la charge du réseau électrique.

La conception d'entraînements électriques de grande puissance est également une activité qui pourrait bénéficier selon nous d'un type particulier d'outil de simulation, c'est-à-dire un simulateur d'entraînements électriques en temps réel entièrement numérique. Un tel outil permettrait aux ingénieurs chargés de la conception de l'entraînement de développer un prototype de commande et de faire des essais avec cette commande en la raccordant à un simulateur en temps réel qui reproduit fidèlement le comportement de l'étage de puissance de l'entraînement (source, convertisseur, machine et charge). Bien que le prototype doive éventuellement être testé sur "le vrai" système, nous sommes

d'avis que les premières phases de conception bénéficieraient de l'utilisation d'un tel simulateur.

Les simulateurs en temps réel entièrement numériques existent depuis le début des années 1990. Ils sont principalement utilisés pour la simulation des grands réseaux électriques et les essais liés aux équipements qui s'y raccordent, par exemple les disjoncteurs de lignes de transport à haute tension, les relais de protection, et la commande pour les compensateurs statiques. La plupart de ces simulateurs comprennent des modèles de convertisseurs à électronique de puissance ainsi que quelques modèles de machines utilisées dans les entraînements modernes. Cependant, les contraintes imposées par le temps réel forcent l'utilisation de modèles discrets décrits par des équations aux différences et il n'est pas possible d'utiliser une technique itérative comme on le fait pour simuler ces systèmes en temps différé. La simulation d'un entraînement sur ces simulateurs exige l'ajout de composantes parasites afin d'assurer la stabilité de la simulation. Dans un contexte de réseau électrique où le comportement d'une usine complète est étudié, ces composantes ne sont pas parasites et consistent en des charges passives qui se retrouvent à proximité des entraînements. Cependant, lorsque l'on veut étudier le comportement d'un entraînement isolé, ces composantes parasites sont nécessaires et modifient la dynamique du système de façon appréciable et selon nous inacceptable.

Les simulateurs de réseaux électriques en temps réel sont basés sur la modélisation du réseau à l'aide de l'approche nodale. La plupart de ces simulateurs utilisent de plus la méthode de discrétisation trapézoïdale, aussi connue sous le nom de méthode de Tustin. Afin de palier au problème sus-mentionné, nos travaux de recherche ont été réalisés dans le cadre de deux mandats, pour le compte du Laboratoire de Simulation de Réseaux (LSR) de l'IREQ, visant spécifiquement à étudier la possibilité d'utiliser une autre approche de modélisation, soit l'approche par variables d'état, pour réaliser des simulations en temps réel. Ces mandats visaient également à explorer des méthodes d'intégration à pas fixe alternatives à la discrétisation trapézoïdale dans le but d'éliminer les principaux

inconvénients liés à cette méthode, notamment les oscillations numériques produites lors de discontinuités.

L'utilisation de l'approche par variables d'état peut se justifier de plusieurs façons. D'abord, les équations dynamiques de plusieurs classes de systèmes s'expriment naturellement sous forme d'équations différentielles ordinaires d'ordre un. C'est le cas des systèmes électriques comprenant des convertisseurs de puissance et c'est également le cas pour les machines électriques. De plus, il existe de nombreux outils mathématiques nous permettant d'analyser les propriétés d'un système modélisé sous forme d'équations d'état. Les notions de commandabilité et d'observabilité sont basées sur cette représentation. Connaissant les équations d'état d'un système, on peut également analyser sa stabilité puisque les valeurs propres de la matrice d'état A sont en fait les pôles du système. Un autre avantage de la représentation d'état est qu'elle permet de changer facilement de méthode d'intégration. Pour changer de méthode d'intégration avec l'approche nodale, il faut discrétiser chaque composante à nouveau avec une méthode différente. Comme une part de nos travaux consiste à étudier les algorithmes d'intégration, cet avantage est appréciable. Cette approche permet également d'implanter des algorithmes à pas variables plus facilement qu'avec l'approche nodale. Cependant, comme nous aspirons à réaliser des simulations en temps réel, cet avantage est moins marqué dans notre cas. Enfin, l'approche par variable d'état se généralise facilement aux systèmes non-linéaires et est souvent la méthode de choix dans ce cas.

Les travaux décrits dans cette thèse visent donc à éliminer certaines contraintes des simulateurs actuels, et ainsi permettre la simulation numérique en temps réel d'entraînements électriques. Notre objectif principal est de développer des modèles et/ou méthodes de simulation qui permettent de simuler le comportement dynamique d'un entraînement électrique isolé. Dans ce contexte, nous avons aussi comme objectif d'étudier les algorithmes d'intégration à pas fixe, dans le but d'utiliser une technique d'intégration à la fois assez simple pour être applicable en temps réel, et assez robuste pour simuler de façon

stable et précise les systèmes particulièrement exigeants que sont les entraînements électriques.

Nous tenons à mentionner que nous nous concentrons exclusivement sur les techniques de modélisation et de simulation des entraînements dans cette thèse. Bien que notre but ultime soit de développer un simulateur permettant de connecter une commande réelle à un entraînement simulé, nous ne décrivons pas d'expérimentation où une telle interconnection est réalisée. Cette tâche dépasse le cadre de notre thèse.

Nous avons commencé par nous familiariser avec la génération automatique des équations d'état pour un système électrique général. En tant que co-auteur du Power System Blockset (PSB) [1], une librairie dédiée à la simulation des réseaux électriques et des entraînements dans l'environnement Matlab/Simulink, nous étions familiers avec les principes liés à l'obtention de ces équations, mais cette partie a été développée par un chercheur de l'IREQ, Gilbert Sybille, dans la version 1 du PSB. Notre contribution à cette première version a consisté à participer au développement des modèles de machines électriques et de régulateurs du PSB [2]. Lors de notre étude, nous nous sommes familiarisés avec une méthode d'obtention des équations d'état basée sur la théorie des graphes linéaires [3]. Cette méthode s'est avérée très performante dans l'environnement Matlab et notre implantation, un petit programme que nous avons écrit et que nous utilisions pour nos besoins personnels dans le cadre de la présente thèse, a grandi. La technique utilisée a été implantée dans la version 2 du PSB et est décrite en détail dans notre premier chapitre pour un système électrique général. Quelques exemples simples sont utilisés tout au long du développement afin de faciliter la compréhension de la méthode. Le premier chapitre décrit le point de départ de nos travaux, soit l'obtention des équations d'état du système.

La technique décrite au chapitre 1 s'applique pour un système électrique linéaire. Toutes les non-linéarités (interrupteurs, machines) sont considérées comme des modèles externes à la représentation d'état et apparaissent comme des entrées, sous forme de sources de courant, du point de vue de cette dernière. Nous discutons donc au chapitre 2 de la

modélisation des interrupteurs et des machines électriques. Une technique originale permettant d'inclure les interrupteurs dans les matrices d'état du système est d'abord décrite. Un modèle de machine asynchrone est ensuite défini et une attention particulière est portée aux différentes transformations de référentiel qui permettent de simplifier le modèle. Notre étude porte exclusivement sur la machine asynchrone, étant donnée que c'est la machine la plus couramment utilisée dans l'industrie. Les équations associées à la machine sont typiquement solutionnées séparément du reste du système. Or, nous avons adopté une méthode qui permet d'inclure les équations d'état de la machine dans la représentation d'état du reste du système, permettant une solution simultanée sans délai du système entier. Cette méthode est décrite à la fin du chapitre 2, lequel se termine par une validation en temps différé des techniques décrites à date.

Au chapitre 3, nous nous attardons à la question de la discrétisation du processus d'intégration, notre entraînement étant décrit par des équations différentielles qui doivent être solutionnées en temps réel. Un entraînement constitue un système d'une classe particulièrement délicate à discrétiser et ces particularités imposent des restrictions quant au choix de la méthode de discrétisation. Une méthode d'intégration matricielle récemment publiée est analysée et comparée à la discrétisation trapézoïdale.

Enfin, la thèse se termine par un chapitre portant sur l'implantation en temps réel des techniques de simulation et d'intégration exposées aux chapitres antérieurs. Pour ce faire, un système consistant en un entraînement industriel courant est étudié. La complexité du système à simuler couplée aux contraintes sévères du temps réel forcent l'utilisation d'ordinateurs parallèles, dont nous décrivons ensuite les deux principales architectures. La séparation des tâches sur plusieurs processeurs à l'aide de techniques de découplage est discutée, puis les principaux facteurs que nous avons eus à considérer pour réduire le temps de calcul de nos programmes sont énumérés. Quelques optimisations apportées à nos algorithmes sont enfin décrites, après quoi la simulation de l'entraînement en temps réel est analysée.

Avant d'attaquer le corps de notre sujet, nous allons faire une brève revue de la littérature liée aux différents aspects de notre sujet de recherche.

REVUE DE LA LITTÉRATURE

Au début de ce projet, nous avions un mandat bien clair qui consistait à étudier la possibilité de simuler, en temps réel et à l'aide de la représentation d'état, des entraînements électriques. Nous avons donc fait une recherche bibliographique en ce sens mais n'avons trouvé aucun article qui couvrait tous ces concepts. Nous avons donc été contraints de faire plusieurs recherches avec un nombre réduit de concepts et avons ainsi trouvé plusieurs publications intéressantes. Nous présentons ici les principaux résultats de ces recherches.

D'abord, l'obtention des équations d'état d'un système électrique est un sujet qui a été abondamment couvert par la littérature de la fin des années 1960 par, entre autres, Balabanian et Bickart [3] et durant les années 1970 par de nombreux auteurs, dont Rohrer [4], Gille [5], et Chua et Lin [6]. Le sujet a également été repris dans quelques livres plus modernes, dont celui de Rajagopalan [7].

Nous avons développé une certaine expertise en modélisation de machines électriques lors du développement du Power System Blockset (PSB) [1]. Ce projet a également fait l'objet de notre mémoire de maîtrise [2]. Cependant, les modèles développés pour la première version du PSB sont des modèles dans le domaine continu, destinés à une utilisation dans l'environnement Simulink. La version 1 du PSB résout les équations différentielles à l'aide d'algorithmes d'intégration itératifs à pas et à ordre variables. Les modèles ne sont donc pas directement utilisables dans le contexte d'une simulation en temps réel. Les modèles utilisés dans la version 2, laquelle permet les simulations discrètes à pas fixe, ont simplement été obtenus en substituant au bloc d'intégration de Simulink "1/s"

une fonction de transfert discrète, en l'occurrence celle de la méthode de discrétisation Euler avant.

Par conséquent, nous avons cherché des modèles alternatifs dans la littérature. Lauw et Meyer [8] présentent le modèle de machine universelle qui est utilisé dans le très populaire logiciel de simulation de réseaux électriques EMTP (Electromagnetic Transients Program) [22]. Ce modèle permet de simuler 12 types de machines électriques différentes, incluant les machines à courant continu et les machines à courant alternatif monophasées et triphasées. À cause de la façon dont EMTP résout les équations d'un réseau (approche nodale et discrétisation trapézoïdale), le modèle de machine universelle est scindé en quatre parties distinctes et la solution des équations de la machine est un processus itératif. La méthode de solution proposée s'adapte donc plutôt mal à une simulation en temps réel.

De leur côté, Vainio et al. [9] proposent un modèle de machine asynchrone monophasée destiné à une simulation en temps réel. Cependant, leur modèle est implémenté sur plusieurs ASIC ("Application Specific Integrated Circuit") de fabrication maison. La période d'échantillonnage requise n'est pas discutée et la discrétisation du modèle est faite selon deux approches (Euler avant et trapézoïdale), les deux étant comparées.

L'article de Gehlot et Alsina [10] présente un modèle de machine asynchrone triphasée, destiné à de la commande en temps réel. Ils utilisent un modèle d'état avec comme variables électriques les courant du stator et les flux du rotor. Ce choix est dicté par l'application visée (commande vectorielle) qui exige une estimation précise des flux au rotor. La topologie simulée est basée sur un onduleur MLI (Modulation de largeur d'impulsion, en anglais "PWM"). Cependant, le convertisseur est approximé de façon simpliste en considérant que la tension présente aux bornes de la machine est constante durant une période d'échantillonnage et égale à la valeur de la source à courant continu. La discrétisation du modèle est basée sur une méthode de prédicteur-correcteur et la simulation démontre d'excellents résultats pour une période d'échantillonnage de 555 µs.

La simulation de convertisseurs de puissance a été l'objet de nombreuses publications scientifiques et se retrouve aussi dans plusieurs manuels de références [6,7]. Cependant, nous avons limité notre recherche à la simulation en temps réel de convertisseurs de puissance et nous n'avons trouvé que quelques publications.

D'abord, Haskew et Byakod [11] proposent une méthode de simulation d'un redresseur à thyristors à 12 impulsions. Leur méthode est basée sur l'approche par variables d'état et la discrétisation est réalisée à l'aide de la méthode d'Euler arrière. La technique proposée exige cependant que les 4096 états possibles du redresseur (12 interrupteurs, deux états possible chacun, $2^{12} = 4096$ possibilités) soient stockés en mémoire. De plus, leur simulation est implémentée en langage Fortran sur un IBM RISC 6000.

Dans le but de nous rapprocher du sujet de notre projet, nous avons ensuite entrepris une recherche sur la simulation d'entraînements électriques en général, en omettant les concepts "représentation d'état" et "temps réel".

Notre première trouvaille est l'article de Kleinhans et al. [12]. On y présente un logiciel permettant la simulation, en temps différé, d'entraînements électriques de configurations variées. On démontre l'utilisation du logiciel à l'aide d'une application de commande à flux orienté d'une machine asynchrone triphasée à cage d'écureuil. L'approche de simulation est modulaire, mais les auteurs ne discutent pas la stratégie d'interface entre les divers modules, ce qui est la principale difficulté dans ce type de simulation. De plus, on ne donne que les équations générales du modèle de machine et aucune mention n'est faite de l'algorithme de solution utilisé.

Une autre publication, celle de Chhaya et Bose [13] présente le développement d'un système expert servant à concevoir, simuler et ajuster des entraînements électriques variés. Le système semble superviser une simulation en temps réel de l'ensemble convertisseur-machine, mais nous n'en sommes pas certains (très peu de détails sur cette partie

dans l'article). Cette simulation est réalisée à l'aide du logiciel SIMNON et sert essentiellement à ajuster les paramètres du contrôleur, lui-même implémenté sur un DSP. Cet article offre une excellente récapitulation des étapes requises pour la conception d'un entraînement électrique.

Les deux publications suivantes sont les seules que nous ayons trouvées qui discutent de simulation en temps réel d'entraînements électriques. Dans celle de Matuonto et al. [14], il n'y a aucune mention quant au convertisseur (qui est un cycloconvertisseur) ce qui nous porte à croire que son fonctionnement est idéalisé et est simulé de façon élémentaire. Les auteurs simulent une machine synchrone à pôles saillants à l'aide de l'algorithme de Runge-Kutta d'ordre 4 (intégration itérative), avec une période d'échantillonnage de 1 ms. Dans l'article de Dezza et al. [15], la topologie simulée est un onduleur MLI, semblable à un cas mentionné plus haut, et le fonctionnement du convertisseur est encore une fois idéalisé. Cependant, la simulation utilise l'algorithme d'Adams-Bashforth d'ordre 3 avec une période d'échantillonnage plus raisonnable de 200 µs. La simulation est réalisée sur un DSP TMS320C30 et l'interface avec la simulation est réalisée grâce au logiciel Labview, qui est exécuté sur un ordinateur Macintosh. Le logiciel Labview permet d'entrer les paramètres de la simulation à priori et de visualiser les résultats sur des graphiques à posteriori.

Nous résumons maintenant les résultats de diverses recherches faites à l'aide de concepts autres que ceux mentionnés à date. Tout d'abord, nous avons gardé en tête tout au long de nos travaux la possibilité de faire de la simulation hybride. Ce que nous entendons par simulation hybride est la simulation d'un système dont une partie est modélisée avec l'approche nodale et l'autre partie est modélisée à l'aide de l'approche par variables d'état. Nous avons recueilli deux publications traitant de simulation hybride. Dans celle de Zavahir et al. [16], on simule un convertisseur HTCC (haute tension courant continu) à l'aide de la représentation d'état et cette simulation s'intègre dans un réseau plus large qui est simulé à l'aide de l'approche nodale. Dans l'article de Kang et Lavers [17], on utilise les deux approches afin d'obtenir un système d'équation sous une forme assez origi-

nale, qui est cependant itérative. Les auteurs prétendent conserver les avantages des deux méthodes tout en éliminant les inconvénients propres à chacune. L'efficacité de la méthode proposée est illustrée à l'aide d'un onduleur triphasé à thyristors.

Un autre papier intéressant, que nous n'arrivons pas à placer dans une catégorie particulière, est celui de Wasynczuk et Sudhoff [18]. Ce papier nous intéresse puisqu'on y décrit une méthode systématique d'obtention de la représentation d'état d'un système complet. En guise d'exemple, on illustre l'application de la méthode développée à l'aide d'une topologie consistant en une machine synchrone raccordée à un redresseur à diodes. Le tout est implémenté dans l'environnement ACSL, un langage symbolique permettant de solutionner les équations différentielles. La procédure décrite est cependant de nature itérative et n'est donc pas applicable en temps réel.

La discrétisation du processus d'intégration est le dernier sujet que nous discutons dans cette revue de la littérature. Nous avons étudié durant un certain temps une publication de De Abreu-Garcia et Hartley [19] qui décrit une technique d'intégration discrète, baptisée "Matrix Stability Region Placement" (MSRP). Il s'agit d'un opérateur d'intégration matriciel qui permet de fixer la période d'échantillonnage non pas en fonction de la stabilité de la simulation, mais en fonction du degré de précision désiré. De plus, cette méthode a été spécifiquement conçue pour simuler, en temps réel, les systèmes rigides. Elle exige cependant que le système à simuler se présente sous forme de représentation d'état (linéaire ou non). Enfin, nous avons analysé les méthodes d'intégration discrètes à l'aide de deux volumes de référence, soir celui de Chua et Lin [6] et celui de Hartley et al. [20].

Cette recherche bibliographique nous permet de tirer trois grandes conclusions (la troisième étant une conséquence des deux autres):

a. beaucoup de publications traitent de l'un des sujets suivants: simulation de machines électriques, de convertisseurs de puissance ou d'entraînements électriques;

b. seulement quelques unes de ces publications traitent de simulation en temps réel d'entraînements électriques et dans ces publications, la dynamique du convertisseur est idéalisée; par conséquent, le convertisseur n'est pas vraiment simulé;

c. à date, personne ne semble avoir publié d'article où l'on simule en temps réel un ensemble convertisseur-machine électrique (un entraînement) en mode isolé.

Cette dernière conclusion nous encourage et nous angoisse un peu en même temps. Elle nous encourage dans l'optique où un projet de doctorat doit apporter une contribution originale. Il semble que ce soit le cas. Cependant, le fait de n'avoir trouvé aucune publication traitant spécifiquement de notre sujet est intrigant, en ce sens où il s'agit de quelque chose de difficile à réaliser, ou alors il n'y a pas d'intérêt (à date) à poursuivre un tel but. Dans l'optique de cette dernière remarque, il est fort possible que les limites de la technologie aient empêché la réalisation de simulations en temps réel d'entraînements électriques. Or ces limites sont constamment repoussées.

CHAPITRE 1

MODÉLISATION DES SYSTÈMES ÉLECTRIQUES À L'AIDE DE LA REPRÉSENTATION D'ÉTAT

Afin d'étudier la possibilité d'utiliser l'approche par variables d'état pour faire des simulations en temps réel, il est important de pouvoir générer systématiquement les équations d'état d'un système quelconque. Ce premier chapitre a pour but de présenter une synthèse des notions associées à la modélisation des systèmes électriques à l'aide de la représentation d'état.

Nous avons constaté qu'il existe essentiellement deux méthodes pour obtenir les équations d'état d'un réseau électrique. Une première méthode consiste en l'analyse de réseaux résistifs multiports [6,3]. Avec cette méthode, on extrait sous forme de ports toutes les composantes non-résistives du circuit, puis on substitue ces ports par une source appropriée (tension ou courant) et on obtient ainsi la contribution de chacune des composantes sur l'ensemble du circuit. La seconde méthode est basée sur la théorie des graphes [6,3]. C'est cette dernière approche qui est décrite dans le présent chapitre. Nous comparons les performances des deux méthodes à la fin du chapitre.

Nous présentons d'abord les notions de base de la théorie des graphes linéaires. Ces notions sont ensuite appliquées à l'obtention des équations d'état de systèmes simples, comportant uniquement des sources idéales et des composantes passives (résistances, condensateurs, inductances). Nous ajoutons ensuite la possibilité d'inclure dans les systèmes à modéliser les transformateurs et les inductances couplées. Nous discutons des

capacités de notre méthode à détecter les erreurs qui peuvent découler de mauvaises données d'entrée et concluons ce premier chapitre en comparant les performances de notre méthode d'obtention des équations d'état avec une méthode alternative.

De façon générale, un système est décrit avec l'approche par variables d'état par les deux équations

$$\begin{aligned} \dot{x} &= Ax + Bu \\ y &= Cx + Du \end{aligned}, \tag{1-1}$$

où x est le vecteur des variables d'état, u est le vecteur des entrées et y est le vecteur des sorties. Pour ce qui est des matrices, A est la matrice des paramètres du système, tandis que B et C sont les matrices de couplage des entrées et des sorties, respectivement. Enfin, D lie directement les sorties aux entrées et est souvent nulle.

Il est important de noter que les systèmes électriques pour lesquels la procédure décrite ici est appliquée sont des systèmes linéaires. Les éléments non-linéaires comme les machines électriques et les interrupteurs sont considérés à ce stade comme des sources de courant externes et sont des entrées du point de vue de la représentation d'état. Nous décrivons au chapitre suivant des méthodes permettant d'inclure ces composants dans la représentation d'état.

1.1 Théorie des graphes linéaires

Afin d'expliquer notre algorithme de calcul de représentation d'état, nous devons d'abord rappeler quelques notions fondamentales sur la théorie des graphes.

1.1.1 Définitions

Le graphe correspondant à un réseau électrique donné est un ensemble de points (les noeuds du réseau) reliés ensemble par des segments (les éléments du réseau). Un noeud de référence doit être choisi et il est pratique courante d'assigner à ce noeud le numéro zéro. Dans le cas de systèmes électriques, le graphe est toujours orienté, c'est-à-dire qu'une direction est assignée à chaque segment. Cette direction correspond au sens de circulation du courant dans l'élément que représente le segment en question. Une exception ne respecte pas cette règle: les sources de tension indépendantes, auxquelles on doit assigner la direction contraire à celle du courant. La figure 1-1 représente un circuit simple et son graphe. À ce stade-ci, la nature des éléments formant le réseau n'est d'aucune importance et cette information n'est pas contenue dans le graphe.

Un arbre est défini comme étant un sous-graphe connecté comprenant tous les noeuds du graphe à l'étude ainsi qu'un certain nombre de ses segments, lesquels sont choisis de telle sorte qu'aucune maille ne soit formée. Tous les noeuds doivent être touchés par au moins un segment. Les segments faisant partie de l'arbre portent de nom de *branches* de l'arbre, alors que les segments ne faisant pas partie de l'arbre sont nommés les *liens* de l'arbre. Si un graphe comprend n+1 noeuds, l'arbre doit contenir n branches.

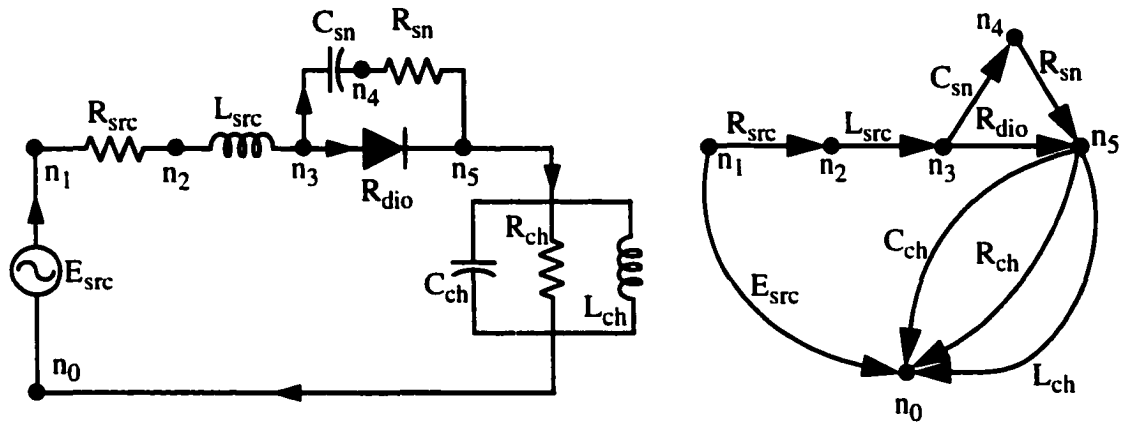


Figure 1-1 Un circuit simple et son graphe.

La figure 1-2 illustre un arbre correspondant au graphe de la figure 1-1. Nous disons *un arbre* plutôt que *l'arbre* parce qu'il n'est pas unique pour un graphe donné.

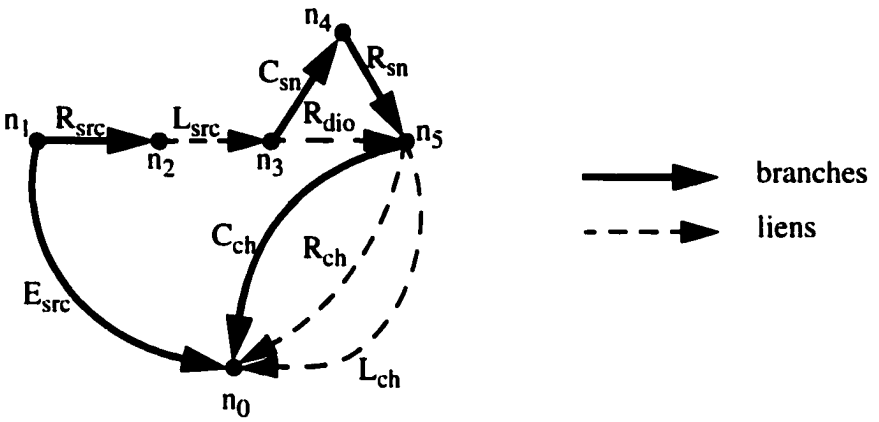


Figure 1-2 Un arbre correspondant au graphe de la figure 1-1.

1.1.2 Matrices fondamentales

Il existe trois matrices fondamentales lorsque la théorie des graphes est utilisée pour développer les équations d'état d'un système électrique, soit:

a. la matrice d'incidence nodale;
b. la matrice des mailles fondamentales;
c. la matrice des coupures fondamentales.

1.1.2.1 Matrice d'incidence nodale

Cette matrice est souvent symbolisée par A et ne doit pas être confondue avec la matrice d'état A. La matrice d'incidence nodale A décrit la topologie du système en faisant abstraction de la nature des éléments qui le composent. Pour un graphe comportant n

noeuds et s segments, la matrice A est formée de n rangées et s colonnes. Ses éléments peuvent prendre les valeurs 0, -1 ou 1, selon les conditions décrites en (1-2).

$$A_{ij} = \begin{cases} 0: & \text{si le segment } j \text{ n'est pas incident au noeud } i \\ -1: & \text{si le segment } j \text{ est incident au noeud } i \text{ et sort de ce dernier} \\ 1: & \text{si le segment } j \text{ est incident au noeud } i \text{ et entre dans ce dernier} \end{cases} \quad (1\text{-}2)$$

La matrice d'incidence nodale n'est pas vraiment utile dans le développement qui suit. On utilise plutôt la matrice d'incidence nodale réduite, qui est simplement la matrice d'incidence nodale de laquelle on retire la rangée correspondant au noeud de référence. La matrice d'incidence nodale réduite correspondant au graphe de la figure 1-1 est montrée en (1-3).

$$A = \begin{array}{c} \\ n_1 \\ n_2 \\ n_3 \\ n_4 \\ n_5 \end{array} \begin{array}{c} \begin{array}{ccccccccc} E_{src} & C_{sn} & C_{ch} & R_{src} & R_{sn} & R_{ch} & R_{dio} & L_{src} & L_{ch} \end{array} \\ \begin{bmatrix} -1 & 0 & 0 & -1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & -1 & 0 \\ 0 & -1 & 0 & 0 & 0 & 0 & -1 & 1 & 0 \\ 0 & 1 & 0 & 0 & -1 & 0 & 0 & 0 & 0 \\ 0 & 0 & -1 & 0 & 1 & -1 & 1 & 0 & -1 \end{bmatrix} \end{array} \quad (1\text{-}3)$$

1.1.2.2 Matrice des mailles fondamentales

La seconde matrice importante pour nos développements ultérieurs est la matrice des mailles fondamentales, identifiée par B_f. Elle comporte autant de rangées qu'il y a de liens dans l'arbre et autant de colonnes qu'il y a de segments dans le graphe. On obtient cette matrice, une fois l'arbre choisi, en ajoutant un à un les liens dans le graphe. Une maille fondamentale (une ligne de la matrice B_f) est ainsi formée. L'orientation de la maille est celle du lien qui la définit. La maille fondamentale définie par le lien L_{src} est illustrée à la figure 1-3.

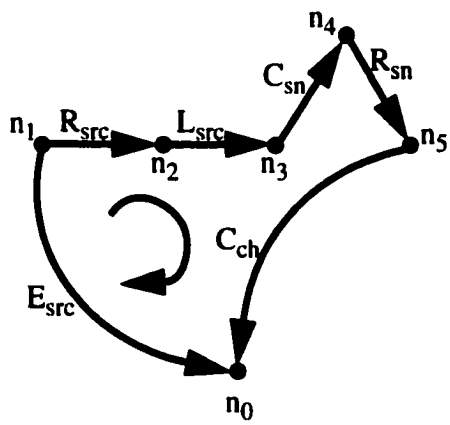


Figure 1-3 Maille fondamentale définie par le lien L_{src}.

Comme la matrice d'admittance nodale, les éléments de la matrice des mailles fondamentales ne peuvent prendre que les valeurs 0, -1 et 1 selon les conditions énoncées en (1-4).

$$B_{f_{ij}} = \begin{cases} 0: \text{ si la maille définie par le lien } i \text{ ne comprend pas le segment } j \\ 1: \text{ si la maille définie par le lien } i \text{ comprend le segment } j \\ \quad \text{et que les orientations coïncident} \\ -1: \text{ si la maille définie par le lien } i \text{ comprend le segment } j \\ \quad \text{et que les orientations ne coïncident pas} \end{cases} \tag{1-4}$$

La matrice des mailles fondamentales B_f correspondant à l'arbre de la figure 1-2 est donnée en (1-5). La partie encadrée correspond à la maille fondamentale illustrée à la figure 1-3.

$$B_f = \begin{array}{c} \begin{matrix} E_{src} & C_{sn} & C_{ch} & R_{src} & R_{sn} & R_{ch} & R_{dio} & L_{src} & L_{ch} \end{matrix} \\ \left[\begin{array}{ccccccccc} 0 & 0 & -1 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & -1 & 0 & 0 & 1 & 0 & 1 & 0 & 0 \\ \hdashline -1 & 1 & 1 & 1 & 1 & 0 & 0 & 1 & 0 \\ \hdashline 0 & 0 & -1 & 0 & 0 & 0 & 0 & 0 & 1 \end{array}\right] \end{array} \begin{matrix} R_{ch} \\ R_{dio} \\ \hdashline L_{src} \\ \hdashline L_{ch} \end{matrix} \tag{1-5}$$

La matrice des mailles fondamentales permet aussi d'exprimer une généralisation de la loi de Kirchhoff des tensions. En considérant que v est le vecteur des tensions aux bornes de tous les segments d'un graphe, on obtient l'expression (1-6). Pour l'arbre de la figure 1-2, le résultat est l'expression (1-7).

$$B_f v = 0 \tag{1-6}$$

$$\begin{bmatrix} 0 & 0 & -1 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & -1 & 0 & 0 & 1 & 0 & 1 & 0 & 0 \\ -1 & 1 & 1 & 1 & 1 & 0 & 0 & 1 & 0 \\ 0 & 0 & -1 & 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} v_{Esrc} \\ v_{Csn} \\ v_{Cch} \\ v_{Rsrc} \\ v_{Rsn} \\ v_{Rch} \\ v_{Rdio} \\ v_{Lsrc} \\ v_{Lch} \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \end{bmatrix} \tag{1-7}$$

En partitionnant correctement l'expression (1-6), il est possible d'exprimer les tensions des liens de l'arbre en fonction des tensions aux bornes des branches de l'arbre. En effet, en utilisant les indices t et l pour identifier respectivement les grandeurs liées aux branches et aux liens de l'arbre et en désignant la matrice identité par I, on obtient (1-8). En ré-arrangeant cette expression, on obtient la relation (1-9).

$$\begin{bmatrix} B_t & I \end{bmatrix} \begin{bmatrix} v_t \\ v_l \end{bmatrix} = B_t v_t + v_l = 0 \tag{1-8}$$

$$v_l = -B_t v_t \tag{1-9}$$

L'équation (1-9) nous permet de conclure que pour un système comprenant s segments et n branches, il suffit de calculer les tensions aux bornes des n branches (vecteur v_t) et les tensions des s-n liens (vecteur v_l) seront obtenues à partir des tensions de branches par combinaison linéaire.

1.1.2.3 Matrice des coupures fondamentales

La dernière matrice nécessaire pour les développements qui suivent est la matrices des coupures (en anglais "cutsets") fondamentales, que nous identifions par le symbole Q_f. Une coupure est créée lorsque l'on retire certains segments d'un graphe et que ce retrait a pour effet de couper le graphe en deux sous-graphes disjoints. Une coupure fondamentale est associée à chaque branche d'un arbre. En effet, si une branche est retirée de l'arbre, on obtient nécessairement deux graphes disjoints. Une coupure fondamentale est donc définie par une branche de l'arbre et par tous les liens de l'arbre qui transitent entre les deux sous-arbres obtenus lors du retrait de la branche. L'orientation de la coupure est la même que celle de la branche qui définit la coupure. Si on prend par exemple l'arbre de la figure 1-2 et que l'on retire la branche correspondant à C_{ch} (entre les noeuds n_5 et n_0), on obtient deux arbres tel que montré à la figure 1-4.

On remarque que cette coupure fondamentale est orientée de l'arbre 2 vers l'arbre 1 (orientation de la branche retirée, C_{ch}) et que les liens transitant entre les deux arbres sont R_{ch}, L_{ch} et L_{src}. Cette coupure fondamentale est donc définie par la branche C_{ch} et les liens R_{ch}, L_{ch} et L_{src}. La matrice des coupures fondamentales permet de définir les cou-

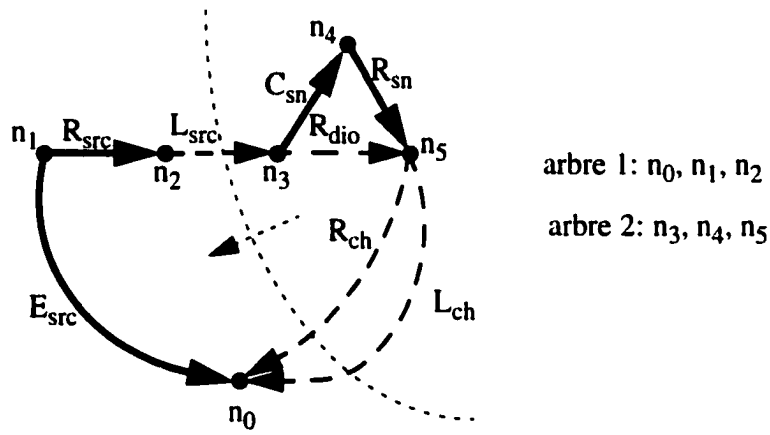


Figure 1-4 Coupure fondamentale formée par le retrait de la branche C_{ch}.

pures de façon concise. Elle comporte autant de rangées qu'il y a de branches dans l'arbre et autant de colonnes qu'il y a de segments dans le graphe. Comme pour la matrice d'admittance nodale et la matrice des mailles fondamentales, les éléments de la matrice des coupures fondamentales peuvent prendre les valeurs 0, -1 ou 1 selon les conditions énoncées en (1-10).

$$Q_{fij} = \begin{cases} 0: \text{ si la coupure définie par la branche } i \text{ n'est pas traversée par le segment } j \\ 1: \text{ si la coupure définie par la branche } i \text{ est traversée par le segment } j \text{ et que les orientations coïncident} \\ -1: \text{ si la coupure définie par la branche } i \text{ est traversée par le segment } j \text{ et que les orientations ne coïncident pas} \end{cases} \quad (1\text{-}10)$$

La matrice des coupures fondamentales obtenue pour l'arbre de la figure 1-2 est montrée en (1-11). La partie encadrée montre la rangée associée à la coupure fondamentale causée par la branche C_{ch}, décrite plus haut.

$$Q_f = \begin{array}{c} \begin{matrix} E_{src} & C_{sn} & C_{ch} & R_{src} & R_{sn} & R_{ch} & R_{dio} & L_{src} & L_{ch} \end{matrix} \\ \left[\begin{array}{ccccccccc} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 1 & -1 & 0 \\ \hdashline 0 & 0 & 1 & 0 & 0 & 1 & 0 & -1 & 1 \\ \hdashline 0 & 0 & 0 & 1 & 0 & 0 & 0 & -1 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 1 & -1 & 0 \end{array}\right] \begin{matrix} E_{src} \\ C_{sn} \\ \hdashline C_{ch} \\ \hdashline R_{src} \\ R_{sn} \end{matrix} \end{array} \qquad (1\text{-}11)$$

La matrice des coupures fondamentales est en quelque sorte le dual de la matrice des mailles fondamentales et permet d'exprimer une généralisation de la loi de Kirchhoff des courants:

$$Q_f i = 0 \ , \qquad (1\text{-}12)$$

où i est le vecteur des courants de tous les segments de l'arbre. Pour notre exemple, la relation (1-13) est obtenue.

$$\begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 1 & -1 & 0 \\ 0 & 0 & 1 & 0 & 0 & 1 & 0 & -1 & 1 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & -1 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 1 & -1 & 0 \end{bmatrix} \begin{bmatrix} i_{Esrc} \\ i_{Csn} \\ i_{Cch} \\ i_{Rsrc} \\ i_{Rsn} \\ i_{Rch} \\ i_{Rdio} \\ i_{Lsrc} \\ i_{Lch} \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix} \qquad (1\text{-}13)$$

La matrice des coupures fondamentales permet donc de lier les courants des branches de l'arbre aux courants des liens. Il est possible de partitionner la relation (1-12) comme nous l'avons fait plus haut pour la matrice des mailles fondamentales. L'expres-

sion (1-12) devient alors (1-14). En ré-arrangeant cette expression, on obtient la relation (1-15).

$$\begin{bmatrix} I & Q_l \end{bmatrix} \begin{bmatrix} i_t \\ i_l \end{bmatrix} = i_t + Q_l i_l = 0 \tag{1-14}$$

$$i_t = -Q_l i_l \tag{1-15}$$

Le résultat (1-15) nous permet donc de conclure qu'une fois calculés les courants des liens, les courants des branches s'obtiennent par combinaison linéaire de ces derniers.

Il existe une relation entre la matrice des coupures fondamentales et la matrice des mailles fondamentales (le symbole ' indique la transposée d'une matrice):

$$Q_f(B_f)' = (B_f)'Q_f = 0 \tag{1-16}$$

À partir de cette relation, il découle que

$$B_t = -(Q_l)' \tag{1-17}$$

Il est donc possible d'exprimer les deux lois de Kirchhoff (1-9) et (1-15) à partir de la matrice des coupures fondamentales seulement (partie associée aux liens), tel qu'exprimé en (1-18). Nous exploiterons ce résultat un peu plus loin.

$$\begin{aligned} i_t &= -Q_l i_l \\ v_l &= (Q_l)' v_t \end{aligned} \tag{1-18}$$

1.1.3 Choix des variables d'état

À date, notre discussion de la théorie des graphes est de nature purement topologique, en ce sens que nous faisons abstraction de la nature des composantes. En effet, l'obtention de l'arbre et le calcul des trois matrices fondamentales ont été présentés sans parler de la nature des composantes. Nous allons maintenant considérer ces composantes.

L'obtention des équations d'état à l'aide de la théorie des graphes est basée sur les trois catégories d'équations suivantes:

a. l'application de la loi de Kirchhoff des courants;
b. l'application de la loi de Kirchhoff des tensions;
c. les relations courant/tension de chacune des composantes du système.

Pour des systèmes passifs ne comprenant que des sources idéales et les trois éléments de base, les relations courant/tension et tension/courant requises sont les suivantes:

$$\begin{aligned} v_R &= Ri_R & i_R &= \frac{v_R}{R} \\ v_C &= \frac{1}{C}\int i_C dt + v_C(0) & i_C &= C\frac{dv_C}{dt} \\ v_L &= L\frac{di_L}{dt} & i_L &= \frac{1}{L}\int v_L dt + i_L(0) \end{aligned} \qquad (1\text{-}19)$$

Outre l'approche par variables d'état, il existe deux autres méthodes couramment utilisées pour solutionner les systèmes électriques:

a. méthode des noeuds;
b. méthode des mailles.

Nous allons discuter brièvement de ces deux méthodes afin de justifier le choix des variables dans l'approche par variables d'état. D'abord, ces deux méthodes découlent de l'application d'une des deux lois de Kirchhoff (courants ou tensions) suivie d'une substitution des relations courant/tension ou tension/courant des diverses composantes puis d'une élimination des variables dépendantes à l'aide de l'autre loi de Kirchhoff (tensions ou courants).

Dans le cas de la méthode des mailles, on applique d'abord la loi de Kirchhoff des tensions dans toutes les mailles indépendantes, puis on substitue dans les équations obtenues les relations tension/courant (colonne de droite de (1-19)) des composantes passives. On applique ensuite la loi de Kirchhoff des courants dans le but d'éliminer les variables dépendantes et le résultat est un ensemble d'équations intégro-différentielles, comportant à la fois des dérivées et des intégrales des courants de mailles.

Si l'on inverse l'ordre d'application des deux lois de Kirchhoff, on obtient alors la méthode des noeuds. On applique d'abord la loi de Kirchhoff des courants, et on substitue dans le résultat les relations courant/tension des composantes passives (colonne de gauche de (1-19)) puis on élimine les tensions dépendantes en appliquant la loi de Kirchhoff des tensions. Le résultat final est encore un ensemble d'équations intégro-différentielles, mais comportant cette fois-ci des dérivées et des intégrales des tensions de noeuds.

L'application de la méthode des noeuds ou de la méthode des mailles entraîne donc un ensemble d'équations intégro-différentielles. Il est possible d'éliminer les intégrales présentes en dérivant les équations, mais ceci élève l'ordre des équations finales. Il s'avère que les intégrales des variables proviennent d'une part de la substitution de la relation courant/tension des inductances dans le cas de la méthode des noeuds et d'autre part de la substitution de la relation tension/courant des condensateurs dans le cas de la méthode des mailles.

Si les tensions de condensateurs et les courants d'inductances sont conservées comme variables, le résultat de l'application successive des deux lois de Kirchhoff sera un ensemble d'équations différentielles du premier ordre, ou équations d'état, avec comme variables les tensions des condensateurs et les courants des inductances du système. Dans le développement qui suit, nous allons donc utiliser comme variables d'état les tensions des condensateurs et les courants des inductances.

Ceci complète la description des notions de la théorie des graphes requises pour continuer le développement de notre méthode d'obtention des équations d'état d'un système électrique. Nous allons d'abord commencer avec les systèmes comportant exclusivement des sources de tension et de courant idéales et des éléments passifs, soit des résistances, inductances et condensateurs.

1.2 Circuits passifs RLC

L'obtention des équations d'état d'un système électrique à l'aide de la théorie des graphes comporte les grandes étapes suivantes:

a. obtention de l'arbre topologique du système à l'étude;
b. calcul des matrices d'incidence nodale et des coupures fondamentales;
c. construction des matrices d'éléments passifs (*RLC*);
d. calcul des matrices d'état A et B ($\dot{x} = Ax + Bu$);
e. calcul des matrices d'état C et D ($y = Cx + Du$).

Ces étapes seront décrites dans les sections subséquentes, d'abord pour des circuits *RLC* simples, puis pour des circuits comprenant en plus des transformateurs ou des inductances couplées (mutuelles).

1.2.1 Obtention de l'arbre topologique

Lorsque l'on considère des systèmes électriques, les éléments du réseau doivent d'abord être triés dans un certain ordre selon la nature des composantes. Cet ordonnancement déterminera quels composantes sont des branches de l'arbre et lesquelles sont des liens de l'arbre. Nous avons déjà mentionné que les variables d'état sont les tensions des condensateurs et les courants des inductances du système. Il s'agit maintenant de déterminer comment classifier les composantes réactives et déterminer lesquelles il serait préférable de conserver dans l'arbre.

Lorsque nous avons discuté plus haut des lois de Kirchhoff et de leurs liens avec les matrices des mailles et des coupures fondamentales, nous avons conclu que les tensions des branches de l'arbre forment une base à partir de laquelle il est possible d'obtenir par combinaison linéaire les tensions aux bornes de toutes les composantes du système. De façon analogue, les courants des liens de l'arbre forment une base à partir de laquelle il est possible d'obtenir par combinaison linéaire les courants circulant dans toutes les composantes du système. Ceci implique que si nous voulons avoir les tensions de condensateurs et les courants d'inductances comme variables d'état, nous devrions favoriser la présence des condensateurs dans l'arbre et défavoriser la présence des inductances dans l'arbre. Autant que possible, les condensateurs doivent donc être des branches et les inductances des liens.

Si tous les condensateurs du système sont dans l'arbre et qu'aucune inductance ne s'y trouve, l'arbre est dit propre. Dans ce cas, les variables d'état sont toutes indépendantes. Si un condensateur n'a pu être inclus dans l'arbre, c'est que le système comporte une maille capacitive et la tension du condensateur exclu n'est pas indépendante. De la même manière, si une inductance se retrouve dans l'arbre, c'est que notre réseau comprend une coupure inductive, c'est-à-dire qu'il y a un noeud auquel ne sont raccordés que des seg-

ments inductifs. Le courant de l'inductance qui se retrouve dans l'arbre n'est donc pas indépendant.

En appliquant un raisonnement similaire à celui des éléments réactifs, on peut déterminer si les sources indépendantes de tension et de courant doivent être des branches ou des liens. Par définition, une source de tension impose sa tension, peu importe comment on la raccorde. Cette source ne peut donc en aucun cas être un lien puisque si c'était le cas, la tension à ses bornes seraient dictée par les branches de l'arbre. De la même manière, une source de courant, qui impose son courant, ne peut être une branche puisque dans cette éventualité, le courant qui la traverse serait déterminé par les courants des liens. Les sources de tension doivent donc être des branches et les sources de courant doivent être des liens.

Enfin, les seuls éléments dont nous n'avons pas discuté dans le choix des branches sont les résistances. Celles-ci peuvent faire partie de l'arbre ou non. Elles doivent être considérées après les condensateurs dans le choix des branches et on en prend juste assez pour tenter de compléter l'arbre. Si une fois toutes les résistances considérées il manque encore des branches, on doit choisir certaines inductances. Les composantes doivent donc être triées dans l'ordre suivant:

a. les sources de tension indépendantes E (doivent être des branches);
b. les condensateurs C;
c. les résistances R;
d. les inductances L;
e. les sources de courant indépendantes J (doivent être des liens).

Une fois le tri des composantes complété, nous passons à la construction de l'arbre proprement dit. On considère chacune des composantes dans l'ordre établi précédemment. Pour un système comportant n+1 noeuds, l'arbre doit contenir n branches. Les

branches ne doivent pas former de maille. L'arbre de la figure 1-2 a été construit en respectant ces règles et en considérant la diode comme une résistance.

La construction de l'arbre associé à un système donné peut se faire à la main si le système est relativement petit. Cependant, pour un circuit plus grand, la tâche se complique et il devient pratique de programmer un algorithme pour accomplir ce travail. Nous avons implémenté l'algorithme des ensembles de segments ("edge sets") [7]. Il s'agit de créer initialement au plus $n/2$ ensemble de segments vides pour un système comportant $n+1$ noeuds. On considère ensuite un à un les segments (triés selon l'ordre établi ci-dessus) du graphe. Selon que chacun des deux noeuds associés à chaque segment fait partie d'un ensemble de segments ou non, ces noeuds sont ajoutés dans un ensemble spécifique ou les ensembles sont concaténés selon le cas. À la fin du processus, tous les noeuds du graphe sont dans le premier ensemble et tous les autres ensembles sont vides.

Nous avons programmé cet algorithme en langage Matlab. Les cellules ("cell arrays"), un nouveau type de données dans Matlab 5, ont été d'une grande utilité pour accomplir cette tâche.

1.2.2 Calcul des matrices d'incidence nodale et des coupures fondamentales

Une fois que l'on a trouvé un arbre topologique correspondant à notre réseau, on peut construire la matrice d'incidence nodale réduite A. Les segments sont préalablement re-triés, de façon à avoir d'abord les branches de l'arbre, suivies des liens. La matrice d'incidence nodale réduite A est finalement partitionnée ($A = [\, A_t \, A_l \,]$) pour usage ultérieur; A_t contient les colonnes de A correspondant aux branches de l'arbre, alors que A_l contient les colonnes de A correspondant aux liens de l'arbre. La matrice d'incidence nodale réduite correspondant à notre exemple est montrée en (1-3).

La matrice des coupures fondamentales est ensuite obtenue en effectuant une réduction de Gauss-Jordan sur la matrice d'incidence nodale réduite. On obtient une matrice dont la partie de gauche est une matrice identité et la partie de droite est la matrice de coupures des liens, symbolisée par Q_l (Q_f = [1 Q_l]). La matrice des coupures fondamentales correspondant à la matrice d'incidence nodale (1-3) est présentée en (1-11).

Pour des besoins ultérieurs, la matrice des coupures fondamentales des liens Q_l est ensuite partitionnée tel que montré en (1-20). Les zéros présents dans la première colonne et la dernière rangée de (1-20) s'expliquent de la façon suivante. D'abord, si un condensateur du système est un lien (présence de la première colonne de (1-20)), c'est parce qu'il y a une maille capacitive dans le système. Or, une telle maille ne peut inclure de résistance ou d'inductance. Il ne peut donc y avoir de termes non-nuls dans les deux dernières rangées de la première colonne de (1-20). Rappelons que d'après la relation entre Q_l et B_t (1-17), la première colonne de (1-20) est aussi la première rangée de B_t. En appliquant un raisonnement similaire, si une inductance est une branche (présence de la dernière rangée de (1-20)), c'est qu'il y a nécessairement une coupure inductive. Or, une telle coupure ne peut comprendre de branche capacitive ou résistive. Par conséquent, il ne peut donc y avoir de termes non-nuls dans les deux premières colonnes de la dernière rangée de (1-20).

$$
\begin{array}{l}
\text{liens} \longrightarrow \quad C \quad R \quad L \quad J \quad \text{branches} \\
Q_l = \begin{bmatrix} Q_{EC} & Q_{ER} & Q_{EL} & Q_{EJ} \\ Q_{CC} & Q_{CR} & Q_{CL} & Q_{CJ} \\ 0 & Q_{RR} & Q_{RL} & Q_{RJ} \\ 0 & 0 & Q_{LL} & Q_{LJ} \end{bmatrix} \begin{array}{c} \downarrow \\ E \\ C \\ R \\ L \end{array}
\end{array}
\qquad (1\text{-}20)
$$

Pour notre exemple, le partitionnement obtenu est montré en (1-21).

$$Q_l = \begin{array}{c} \\ E \\ C \\ R \end{array} \overset{\begin{array}{cc} \underbrace{\quad R \quad} & \underbrace{\quad L \quad} \end{array}}{\left[\begin{array}{cc|cc} 0 & 0 & 1 & 0 \\ 0 & 1 & -1 & 0 \\ \hline 1 & 0 & -1 & 1 \\ \hline 0 & 0 & -1 & 0 \\ 0 & 1 & -1 & 0 \end{array}\right]} \qquad \begin{array}{lll} Q_{ER} = \begin{bmatrix} 0 & 0 \end{bmatrix} & & Q_{EL} = \begin{bmatrix} 1 & 0 \end{bmatrix} \\ Q_{CR} = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix} & & Q_{CL} = \begin{bmatrix} -1 & 0 \\ -1 & 1 \end{bmatrix} \\ Q_{RR} = \begin{bmatrix} 0 & 0 \\ 0 & 1 \end{bmatrix} & & Q_{RL} = \begin{bmatrix} -1 & 0 \\ -1 & 0 \end{bmatrix} \end{array} \tag{1-21}$$

$$Q_{EC} = Q_{EJ} = Q_{CC} = Q_{CJ} = Q_{RJ} = Q_{LL} = Q_{LJ} = [\,]$$

1.2.3 Matrices des éléments passifs

Cette étape est la dernière avant le calcul des matrices d'état. Il s'agit en fait d'assembler les différentes matrices contenant les valeurs des composantes du système de façon ordonnée. Les composantes faisant partie de l'arbre sont dans une matrice séparée de celle du même type de composantes ne faisant pas partie de l'arbre. Si l'on considère les systèmes ne contenant que des éléments *RLC* (pas d'inductance couplée ni de transformateur), les matrices suivantes sont définies et elles sont toutes diagonales:

a. R_t: matrice des branches résistives;
b. R_l: matrice des liens résistifs;
c. G_t: matrice des conductances faisant partie de l'arbre (R_t^{-1});
d. G_l: matrice des conductances ne faisant pas partie de l'arbre (R_l^{-1});
e. C_t: matrice des branches capacitives;
f. C_l: matrice des liens capacitifs;
g. L_{tt}: matrice des branches inductives;
h. L_{ll}: matrice des liens inductifs.

Pour notre exemple, les matrices suivantes sont obtenues:

$$C_t = \begin{bmatrix} C_{sn} & 0 \\ 0 & C_{ch} \end{bmatrix} \qquad R_l = \begin{bmatrix} R_{ch} & 0 \\ 0 & R_{dio} \end{bmatrix}$$
$$R_t = \begin{bmatrix} R_{src} & 0 \\ 0 & R_{sn} \end{bmatrix} \qquad L_{ll} = \begin{bmatrix} L_{src} & 0 \\ 0 & L_{ch} \end{bmatrix} \qquad L_{tt} = C_l = [\,] \tag{1-22}$$

À partir de ces matrices de composantes élémentaires et des sous-matrices Q_{xy} définies plus haut, on obtient les matrices finales des composantes (1-23). Ces matrices seront définies un peu plus loin lorsque nous développerons les matrices d'état *A* et *B*.

$$\begin{aligned} R &= R_l + Q'_{RR} R_t Q_{RR} \qquad \underline{C} = C_t + Q_{CC} C_l Q'_{CC} \\ G &= G_t + Q_{RR} G_l Q'_{RR} \qquad \underline{L} = L_{ll} + Q'_{LL} L_{tt} Q_{LL} \end{aligned} \tag{1-23}$$

Pour notre exemple, les matrices suivantes sont obtenues:

$$R = \begin{bmatrix} R_{ch} & 0 \\ 0 & R_{dio} + R_{sn} \end{bmatrix} \qquad \underline{C} = \begin{bmatrix} C_{sn} & 0 \\ 0 & C_{ch} \end{bmatrix}$$
$$G = \begin{bmatrix} \frac{1}{R_{src}} & 0 \\ 0 & \frac{1}{R_{sn}} + \frac{1}{R_{dio}} \end{bmatrix} \qquad \underline{L} = \begin{bmatrix} L_{src} & 0 \\ 0 & L_{ch} \end{bmatrix} \tag{1-24}$$

1.2.4 Matrices d'état *A* et *B*

Dans cette section, nous développons les équations requises pour le calcul des matrices d'état de circuits ne comprenant que des composantes *RLC*, sans transformateur ni inductance couplée. Ces deux dernières classes de composantes seront traitées séparément dans les sections suivantes.

À ce stade, nous devons combiner les deux lois de Kirchhoff avec les relations courant/tension des composantes afin de définir les matrices de composantes finales. On définit en premier lieu les vecteurs de tensions et de courants suivants.

$$
\begin{aligned}
v &= \begin{bmatrix} v_t & v_l \end{bmatrix}' & i &= \begin{bmatrix} i_t & i_l \end{bmatrix}' \\
v_t &= \begin{bmatrix} v_E & v_{Ct} & v_{Rt} & v_{Llt} \end{bmatrix}' & i_t &= \begin{bmatrix} i_E & i_{Ct} & i_{Rt} & i_{Llt} \end{bmatrix}' \\
v_l &= \begin{bmatrix} v_{Cl} & v_{Rl} & v_{Lll} & v_J \end{bmatrix}' & i_l &= \begin{bmatrix} i_{Cl} & i_{Rl} & i_{Lll} & i_J \end{bmatrix}'
\end{aligned}
\tag{1-25}
$$

Ensuite, on combine les équations (1-18) et (1-20) pour obtenir (1-26) à (1-33).

$$i_E = -Q_{EC}i_{Cl} - Q_{ER}i_{Rl} - Q_{EL}i_{Lll} - Q_{EJ}i_J \tag{1-26}$$

$$i_{Ct} = -Q_{CC}i_{Cl} - Q_{CR}i_{Rl} - Q_{CL}i_{Lll} - Q_{CJ}i_J \tag{1-27}$$

$$i_{Rt} = -Q_{RR}i_{Rl} - Q_{RL}i_{Lll} - Q_{RJ}i_J \tag{1-28}$$

$$i_{Llt} = -Q_{LL}i_{Lll} - Q_{LJ}i_J \tag{1-29}$$

$$v_{Cl} = Q'_{EC}v_E + Q'_{CC}v_{Ct} \tag{1-30}$$

$$v_{Rl} = Q'_{ER}v_E + Q'_{CR}v_{Ct} + Q'_{RR}v_{Rt} \tag{1-31}$$

$$v_{Lll} = Q'_{EL}v_E + Q'_{CL}v_{Ct} + Q'_{RL}v_{Rt} + Q'_{LL}v_{Llt} \tag{1-32}$$

$$v_J = Q'_{EJ}v_E + Q'_{CJ}v_{Ct} + Q'_{RJ}v_{Rt} + Q'_{LJ}v_{Llt} \tag{1-33}$$

Nous devons maintenant éliminer des équations ci-dessus toutes les variables que nous ne voulons pas conserver dans les équations finales. Rappelons que les variables d'état sont les tensions des condensateurs de l'arbre v_{Ct}, les courants des inductances qui ne sont pas dans l'arbre i_{Lll}, et que les entrées sont les sources de tension v_E et les sources de courant i_J.

Nous commençons par éliminer tous les courants des condensateurs (i_{Ct} et i_{Cl}) et les tensions des condensateurs qui ne sont pas dans l'arbre (v_{Cl}). Nous disposons à cet effet de la relation (1-34), valable pour les systèmes invariants dans le temps.

$$\begin{bmatrix} i_{Ct} \\ i_{Cl} \end{bmatrix} = \begin{bmatrix} C_t & 0 \\ 0 & C_l \end{bmatrix} \frac{d}{dt} \begin{bmatrix} v_{Ct} \\ v_{Cl} \end{bmatrix} \tag{1-34}$$

On peut reformuler (1-27) comme suit:

$$\begin{bmatrix} I & Q_{CC} \end{bmatrix} \begin{bmatrix} i_{Ct} \\ i_{Cl} \end{bmatrix} = -Q_{CR} i_{Rl} - Q_{CL} i_{Ll} - Q_{CJ} i_J \tag{1-35}$$

Nous introduisons ensuite (1-34) dans (1-35) en substituant v_{Cl} par (1-30). Le résultat est le suivant.

$$\begin{aligned} \frac{d}{dt} v_{Ct} &= (\underline{C})^{-1} \left(-Q_{CR} i_{Rl} - Q_{CL} i_{Ll} - Q_{CJ} i_J + \hat{C} \frac{d}{dt} v_E \right) \\ \underline{C} &= C_t + Q_{CC} C_l Q'_{CC} \\ \hat{C} &= -Q_{CC} C_l Q'_{EC} \end{aligned} \tag{1-36}$$

L'équation (1-36) contient encore une variable qui doit être éliminée, soit i_{Rl}. Nous y reviendrons plus loin. Nous allons d'abord éliminer les variables relatives aux inductances. Nous disposons à cet effet de la relation (1-37), valable encore une fois pour les systèmes invariants dans le temps.

$$\begin{bmatrix} v_{Ll} \\ v_{Lt} \end{bmatrix} = \begin{bmatrix} L_{ll} & 0 \\ 0 & L_{tt} \end{bmatrix} \frac{d}{dt} \begin{bmatrix} i_{Ll} \\ i_{Lt} \end{bmatrix} \tag{1-37}$$

On reformule ensuite (1-32) comme suit.

$$\begin{bmatrix} I & -Q'_{LL} \end{bmatrix} \begin{bmatrix} v_{Ll} \\ v_{Lt} \end{bmatrix} = Q'_{EL} v_E + Q'_{CL} v_{Ct} + Q'_{RL} v_{Rt} \tag{1-38}$$

On introduit ensuite (1-37) dans (1-38), tout en substituant i_{Llt} par (1-29). Le résultat est le suivant.

$$\begin{aligned} \frac{d}{dt} i_{Lll} &= (\underline{L})^{-1} \left(Q'_{EL} v_E + Q'_{CL} v_{Ct} + Q'_{RL} v_{Rt} + \hat{L} \frac{d}{dt} i_J \right) \\ \underline{L} &= L_{ll} + Q'_{LL} L_{tt} Q_{LL} \\ \hat{L} &= -Q'_{LL} L_{tt} Q_{LJ} \end{aligned} \tag{1-39}$$

Comme dans le cas de l'équation (1-36), il reste dans (1-39) une variable à éliminer, soit v_{Rt}. Pour éliminer i_{Rl} de (1-36) et v_{Rt} de (1-39), nous disposons des relations suivantes.

$$\begin{aligned} i_{Rl} &= G_l v_{Rl} \\ v_{Rt} &= R_t i_{Rt} \end{aligned} \tag{1-40}$$

En combinant (1-40) avec (1-28) et (1-31) et en ré-arrangeant les termes, on obtient un système de deux équations à deux inconnues (1-41).

$$\begin{aligned} i_{Rl} - G_l Q'_{RR} v_{Rt} &= G_l Q'_{CR} v_{Ct} + G_l Q'_{ER} v_E \\ R_t Q_{RR} i_{Rl} + v_{Rt} &= -R_t Q_{RL} i_{Lll} - R_t Q_{RJ} i_J \end{aligned} \tag{1-41}$$

La solution de ce système est (1-42). Les sous-matrices intermédiaires sont définies dans (1-43). Nous constatons avec (1-43) que l'existence des matrices S et T dépend de

l'existence des matrices inversées dans les expressions t_1 et t_2. Pour le type de système à l'étude, il est possible de démontrer que ces expressions sont toujours non-singulières. Cette démonstration se trouve dans [3].

$$\begin{bmatrix} v_{Rt} \\ i_{Rl} \end{bmatrix} = \begin{bmatrix} S_{11} & S_{12} \\ S_{21} & S_{22} \end{bmatrix} \begin{bmatrix} v_{Ct} \\ i_{Lll} \end{bmatrix} + \begin{bmatrix} T_{11} & T_{12} \\ T_{21} & T_{22} \end{bmatrix} \begin{bmatrix} v_E \\ i_J \end{bmatrix} \tag{1-42}$$

$$\begin{aligned} G &= G_t + Q_{RR} G_l Q'_{RR} & R &= R_l + Q'_{RR} R_t Q_{RR} \\ t_1 &= (R_t G)^{-1} R_t & t_2 &= (G_l R)^{-1} G_l \\ S_{11} &= t_1 Q_{RR} G_l Q'_{CR} & S_{21} &= t_2 Q'_{CR} \\ S_{12} &= t_1 Q_{RL} & S_{22} &= -t_2 Q'_{RR} R_t Q_{RL} \\ T_{11} &= t_1 Q_{RR} G_l Q'_{ER} & T_{21} &= t_2 Q'_{ER} \\ T_{12} &= t_1 Q_{RJ} & T_{22} &= -t_2 Q'_{RR} R_t Q_{RJ} \end{aligned} \tag{1-43}$$

L'étape suivante consiste à insérer le résultat (1-42) dans (1-36) et (1-39) et à ré-arranger les termes. Les matrices intermédiaires (1-44) sont définies et le résultat final est (1-45).

$$\begin{aligned} \tilde{Y} &= Q_{CR} R^{-1} Q'_{CR} & \tilde{H} &= -Q_{CL} + Q_{CR} R^{-1} Q'_{RR} R_t Q_{RL} \\ \tilde{F} &= Q'_{RL} G^{-1} Q_{RL} & \tilde{G} &= Q'_{CL} - Q'_{RL} G^{-1} Q_{RR} G_l Q'_{CR} \\ \hat{Y} &= Q_{CR} R^{-1} Q'_{ER} & \hat{H} &= -Q_{CJ} + Q_{CR} R^{-1} Q'_{RR} R_t Q_{RJ} \\ \hat{F} &= Q'_{RL} G^{-1} Q_{RJ} & \hat{G} &= Q'_{EL} - Q'_{RL} G^{-1} Q_{RR} G_l Q'_{ER} \end{aligned} \tag{1-44}$$

$$\frac{d}{dt}\begin{bmatrix} v_{Ct} \\ i_{Llt} \end{bmatrix} = A\begin{bmatrix} v_{Ct} \\ i_{Llt} \end{bmatrix} + B\begin{bmatrix} v_E \\ i_J \end{bmatrix} + B_2\frac{d}{dt}\begin{bmatrix} v_E \\ i_J \end{bmatrix}$$

$$A = \begin{bmatrix} \underline{C}^{-1} & 0 \\ 0 & \underline{L}^{-1} \end{bmatrix}\begin{bmatrix} -\tilde{Y} & \tilde{H} \\ \tilde{G} & -\tilde{F} \end{bmatrix} \qquad B = \begin{bmatrix} \underline{C}^{-1} & 0 \\ 0 & \underline{L}^{-1} \end{bmatrix}\begin{bmatrix} -\hat{Y} & \hat{H} \\ \hat{G} & -\hat{F} \end{bmatrix} \tag{1-45}$$

$$B_2 = \begin{bmatrix} \underline{C}^{-1} & 0 \\ 0 & \underline{L}^{-1} \end{bmatrix}\begin{bmatrix} \hat{C} & 0 \\ 0 & \hat{L} \end{bmatrix}$$

Un commentaire s'impose sur le résultat (1-45), quant à la présence des dérivées des entrées dans le résultat final. La matrice $\hat{C}$ sera non-nulle uniquement si le système à l'étude comporte au moins une maille formée uniquement de sources de tension idéales et de condensateurs. De la même manière, la matrice $\hat{L}$ sera non-nulle uniquement si le système à l'étude comporte au moins une coupure formée uniquement de sources de courant idéales et d'inductances. Ces deux situations sont normalement considérées comme des dégénérescences [6] et ne sont pas permises. Dans tous le développement qui suit, nous considérons donc la matrice B_2 nulle et par conséquent, les dérivées des entrées n'apparaissent pas dans les équations d'état finales. Les résultats suivants sont obtenus pour notre exemple.

$$\tilde{Y} = \begin{bmatrix} \frac{1}{R_{dio}+R_{sn}} & 0 \\ 0 & \frac{1}{R_{ch}} \end{bmatrix} \qquad \tilde{H} = \begin{bmatrix} 1-\frac{R_{sn}}{R_{dio}+R_{sn}} & 0 \\ 1 & -1 \end{bmatrix}$$

$$\tilde{F} = \begin{bmatrix} R_{src}+\frac{R_{sn}R_{dio}}{R_{dio}+R_{sn}} & 0 \\ 0 & 0 \end{bmatrix} \qquad \tilde{G} = \begin{bmatrix} -1+\frac{R_{sn}}{R_{dio}+R_{sn}} & -1 \\ 0 & 1 \end{bmatrix} \qquad (1\text{-}46)$$

$$\hat{Y} = \begin{bmatrix} 0 \\ 0 \end{bmatrix} \qquad \hat{H} = [\,]$$

$$\hat{F} = [\,] \qquad \hat{G} = \begin{bmatrix} 1 \\ 0 \end{bmatrix}$$

$$A = \begin{bmatrix} \frac{-1}{C_{sn}(R_{dio}+R_{sn})} & 0 & \frac{1-\frac{R_{sn}}{R_{dio}+R_{sn}}}{C_{sn}} & 0 \\ 0 & \frac{-1}{C_{ch}R_{ch}} & \frac{1}{C_{ch}} & \frac{-1}{C_{ch}} \\ \frac{-1+\frac{R_{sn}}{R_{dio}+R_{sn}}}{L_{src}} & \frac{-1}{L_{src}} & \frac{-R_{src}-\frac{R_{sn}R_{dio}}{R_{dio}+R_{sn}}}{L_{src}} & 0 \\ 0 & \frac{1}{L_{ch}} & 0 & 0 \end{bmatrix} \qquad B = \begin{bmatrix} 0 \\ 0 \\ \frac{1}{L_{src}} \\ 0 \end{bmatrix} \qquad (1\text{-}47)$$

1.2.5 Matrices d'état *C* et *D*

Il existe principalement deux types de sorties que l'on peut visualiser dans un système électrique, soit:

a. la tension entre deux noeuds, ce qui couvre toutes les possibilités de mesures de tensions;

b. le courant circulant dans un segment.

1.2.5.1 Sorties tension

Il existe une matrice P ("path matrix", ou matrice des chemins) qui définit un chemin fermé entre chacun des noeuds et le noeud de référence. Ce chemin est formé uniquement de branches de l'arbre. La matrice des chemins P s'obtient directement à partir de la matrice d'incidence nodale réduite selon la relation suivante.

$$P = ((A_t)^{-1})' \tag{1-48}$$

La matrice P est une matrice carrée où chaque rangée représente un noeud du graphe (sauf le noeud de référence) et chaque colonne représente une branche de l'arbre. Comme elle est obtenue à partir de la matrice d'incidence nodale, elle ne contient que des 0, 1 et -1 selon les conditions suivantes.

$$P_{ij} = \begin{cases} 0: \text{ si le chemin entre le noeud } i \text{ et le noeud de référence ne contient pas la branche } j \\ 1: \text{ si le chemin entre le noeud } i \text{ et le noeud de référence contient la branche } j \text{ et que les orientations coïncident} \\ -1: \text{ si le chemin entre le noeud } i \text{ et le noeud de référence contient la branche } j \text{ et que les orientations ne coïncident pas} \end{cases} \tag{1-49}$$

Si l'on connaît toutes les branches comprises entre n'importe quel noeud et le noeud de référence, on connaît alors les branches comprises entre n'importe quelle paire de noeuds. Nous avons donc besoin de ne considérer que les éléments faisant partie de l'arbre pour calculer les tensions entre deux noeuds:

a. les sources de tensions indépendantes v_E;
b. les condensateurs C_t;
c. les résistances R_t;
d. les inductances L_{tt}.

La tension aux bornes d'une source de tension est calculée de façon triviale, et il n'y a qu'à placer un "1" ou un "-1" au bon endroit dans la matrice *D*. La tension aux bornes d'un condensateur de l'arbre est une variable d'état, donc il s'agit simplement de placer un "1" ou un "-1" au bon endroit dans la matrice *C*. Pour ce qui est de la tension aux bornes d'une inductance de l'arbre, elle se calcule avec la relation suivante.

$$v_{Lt} = L_{tt}\frac{d}{dt}i_{Lt} \tag{1-50}$$

Comme i_{Ll} est variable d'état, on obtient i_{Lt} en appliquant la loi de Kirchhoff des courants sur les noeuds où il y a des coupures inductives.

$$i_{Lt} = -Q_{LL}i_{Ll} \tag{1-51}$$

On obtient finalement v_{Lt} en combinant (1-50) et (1-51).

$$v_{Lt} = -L_{tt}Q_{LL}\frac{d}{dt}i_{Ll} \tag{1-52}$$

Enfin, pour ce qui est des tensions aux bornes des résistances de l'arbre, elles sont données par la première équation de (1-42), obtenue plus haut lors du calcul des matrices *A* et *B*.

1.2.5.2 Sorties courant

Dans le cas des tensions, ce sont les éléments qui forment les branches de l'arbre qui forment la base pour le calcul de toutes les tensions. Pour les courants, c'est l'inverse c'est-à-dire que ce sont les liens qui forment la base pour le calcul des courants. Une fois que l'on connaît les courants des liens, on déduit les courant des branches par application

de la loi de Kirchhoff des courants sur les noeuds d'intérêt avec la première équation de (1-18). Nous devons calculer explicitement les courants des éléments suivants:

a. les condensateurs qui sont des liens;
b. les résistances qui sont des liens;
c. les inductances qui sont des liens;
d. les sources de courant indépendantes.

Le courant d'une source de courant s'obtient de façon triviale en plaçant dans la matrice D un "1" ou un "-1" au bon endroit. Le courant d'une inductance est une variable d'état et s'obtient aussi de façon triviale en plaçant dans la matrice C un "1" ou un "-1" au bon endroit. Le courant traversant un des condensateurs est calculé de façon analogue à la tension aux bornes d'une inductances, en appliquant la loi de Kirchhoff des tensions dans les mailles capacitives de circuit:

$$i_{Cl} = C_l Q'_{CC} \frac{d}{dt} v_{Cl} \tag{1-53}$$

Enfin, le courant dans une résistance qui est un lien de l'arbre s'obtient avec la seconde équation de (1-42), développée plus haut lors du calcul des matrices A et B.

1.3 Circuits contenant des transformateurs

Les équations décrites ci-haut sont valables seulement pour les cas où le réseau ne contient que des éléments passifs, sans transformateur ni inductance couplée. Si l'on veut considérer des éléments non-réciproques, comme des transformateurs, il faut reprendre le développement des équations en changeant un certain nombre de choses.

1.3.1 Matrices d'éléments passifs affectées

D'abord, le fait de considérer les transformateurs requiert l'ajout d'un élément important dans nos composantes: le transformateur idéal. C'est le traitement du transformateur idéal qui est crucial pour le bon fonctionnement de la méthode. Le transformateur idéal est un quadripôle, alors que tous les éléments considérés à date sont des bipôles. Chaque transformateur idéal aura donc deux segments lui correspondant dans le graphe. Pour des raisons de causalité, l'un des segments doit être dans l'arbre et l'autre non. Si un transformateur comporte plus de deux bobinages, tous les bobinages d'un côté du transformateur (primaire ou secondaire) devront faire partie de l'arbre et tous les bobinages de l'autre côté ne devront pas en faire partie. Pour des raisons pratiques, nous avons adopté la convention que le primaire d'un transformateur ne comporte qu'un seul bobinage et que le bobinage ou les bobinages du secondaire sont ceux qui feront partie de l'arbre. Le bobinage du primaire ne fera donc jamais partie de l'arbre. La figure 1-5 résume les relations de base requises pour le traitement des transformateurs.

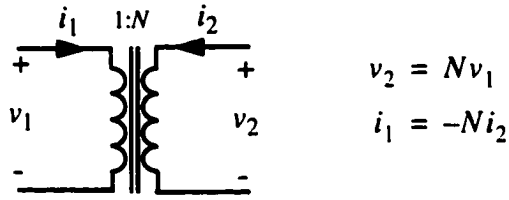


Figure 1-5 Relations de base pour le traitement des transformateurs.

Contrairement au cas des circuits *RLC*, nous n'utilisons plus R_t, R_l, G_t et G_l. Nous définissons de nouvelles relations tension-courant. L'équation (1-54) résume toutes les relations *v-i* requises, où la matrice G_{ll} est une matrice de conductances, G_{tt} est une matrice de résistances, G_{lt} est une matrice de transfert de courant alors que G_{tl} est une matrice de transfert de tension. Le rapport de transformation N doit être placé au bon

endroit dans les matrices G_{lt} et G_{tl}, selon les relations de base de la figure 1-5. On place une conductance nulle dans G_{ll} sur la diagonale, à l'endroit correspondant au lien de l'arbre du primaire du transformateur, et on place aussi une résistance nulle sur la diagonale de G_{tt}, à l'endroit correspondant à la branche de l'arbre du secondaire du transformateur.

$$\begin{bmatrix} i_{Rl} \\ v_{Rt} \end{bmatrix} = \begin{bmatrix} G_{ll} & G_{lt} \\ G_{tl} & G_{tt} \end{bmatrix} \begin{bmatrix} v_{Rl} \\ i_{Rt} \end{bmatrix} \tag{1-54}$$

1.3.2 Réseaux isolés

L'inclusion d'un transformateur idéal dans un système électrique a pour effet de créer deux réseaux électriquement isolés l'un de l'autre et ceci change l'allure du graphe. On obtient en fait deux graphes non-connectés. La figure 1-6 illustre un circuit simple où l'on retrouve un transformateur et le graphe associé. Il est possible d'obtenir les équations d'état d'un tel système à l'aide de la théorie des graphes. Il suffit d'apporter un petit changement dans la procédure à suivre. Il s'agit d'abord de définir un noeud de référence

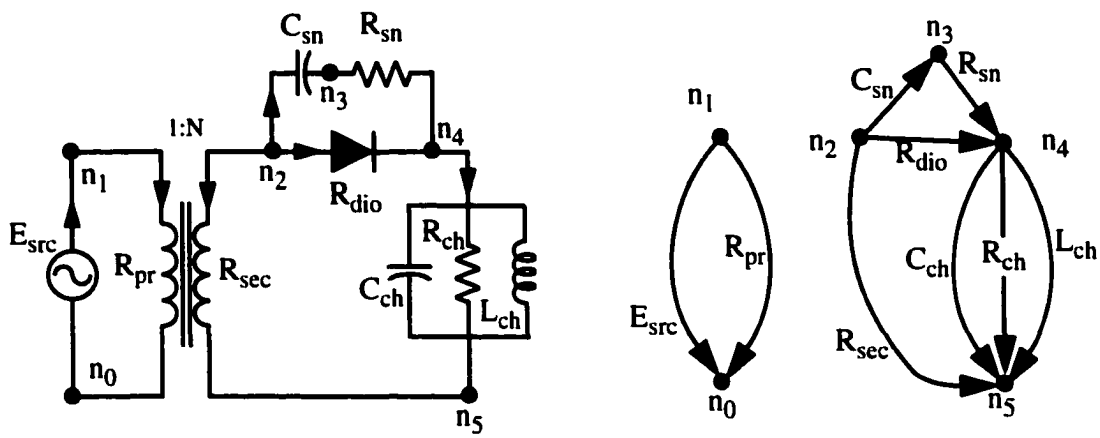


Figure 1-6 Un circuit simple comprenant un transformateur idéal et le graphe correspondant.

dans chacune des parties isolées, plutôt que de définir un seul noeud de référence. Au lieu de retirer une seule rangée de la matrice d'incidence nodale, on retire toutes les rangées associées aux noeuds de références. De plus, pour un système comprenant n+1 noeuds et r réseaux isolés, on aura r arbres comprenant au total n-r branches. Lorsqu'un graphe comprend plusieurs arbres, on parle d'une forêt. Pour l'exemple de la figure 1-6, il y a six noeuds et deux réseaux isolés, donc la forêt contiendra deux arbres et un total de quatre branches. Les branches seront choisies dans le même ordre que dans le cas sans transformateur, sauf que maintenant il faut considérer les enroulements primaire et secondaire de chaque transformateur. Selon la convention établie plus haut (chaque secondaire doit être une branche et chaque primaire doit être un lien), l'ordre de tri des composantes devient le suivant:

a. les sources de tension indépendantes E (doivent être des branches);
b. les secondaires des transformateurs idéaux R_{sec} (doivent être des branches);
c. les condensateurs C;
d. les résistances R;
e. les inductances L;
f. les primaires des transformateurs idéaux R_{sec} (doivent être des liens);
g. les sources de courant indépendantes J (doivent être des liens).

Une forêt correspondant à la figure 1-6 est illustrée à la figure 1-7. La matrice d'incidence nodale correspondant à la forêt choisie est montrée en (1-55), tandis que la matrice des coupures fondamentales des liens et ses sous-matrices sont montrées en (1-56).

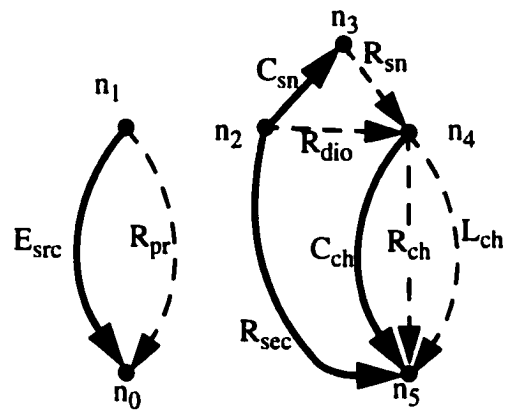


Figure 1-7 Forêt correspondant au système de la figure 1-6.

$$A = \begin{array}{c} \begin{matrix} E_{src} & C_{sn} & C_{ch} & R_{sec} & R_{sn} & R_{ch} & R_{dio} & R_{pr} & L_{ch} \end{matrix} \\ \left[\begin{array}{ccccccccc} -1 & 0 & 0 & 0 & 0 & 0 & 0 & -1 & 0 \\ 0 & -1 & 0 & -1 & 0 & 0 & -1 & 0 & 0 \\ 0 & 1 & 0 & 0 & -1 & 0 & 0 & 0 & 0 \\ 0 & 0 & -1 & 0 & 1 & -1 & 1 & 0 & -1 \end{array}\right] \begin{matrix} n_1 \\ n_2 \\ n_3 \\ n_4 \end{matrix} \\ \begin{matrix} \quad A_t & & & & & & & A_l & \end{matrix} \end{array} \tag{1-55}$$

$$Q_{fl} = \begin{array}{c} \begin{matrix} \quad R & & & & L \end{matrix} \\ \begin{matrix} E \\ C \\ R \end{matrix} \left[\begin{array}{ccccc} 0 & 0 & 0 & -1 & 0 \\ -1 & 0 & 0 & 0 & 0 \\ -1 & 1 & -1 & 0 & 1 \\ 1 & 0 & 1 & 0 & 0 \end{array}\right] \end{array} \qquad \begin{array}{ll} Q_{ER} = \begin{bmatrix} 0 & 0 & 0 & 1 \end{bmatrix} & Q_{EL} = \begin{bmatrix} 0 \end{bmatrix} \\ Q_{CR} = \begin{bmatrix} -1 & 0 & 0 & 0 \\ -1 & 1 & -1 & 0 \end{bmatrix} & Q_{CL} = \begin{bmatrix} 0 \\ 1 \end{bmatrix} \\ Q_{RR} = \begin{bmatrix} 1 & 0 & 1 & 0 \end{bmatrix} & Q_{RL} = \begin{bmatrix} 0 \end{bmatrix} \end{array} \tag{1-56}$$

$$Q_{EC} = Q_{EJ} = Q_{CC} = Q_{CJ} = Q_{RJ} = Q_{LL} = Q_{LJ} = [\,]$$

1.3.3 Matrices d'état *A* et *B*

Bien que l'inclusion des transformateurs change de façon subtile la procédure d'obtention de la forêt topologique, les calculs menant aux matrices d'état changent quant à eux complètement. Les enroulements du transformateur sont considérés comme

des résistances ou conductances, puisque leur présence modifie essentiellement les calculs de nature résistive, tel qu'exprimé en (1-54). Les résultats du travail d'élimination des variables capacitives et inductives réalisé plus haut, dans le cas où il n'y avait pas de transformateur, demeurent valables (équations (1-36) et (1-39)). Cependant, nous devons reprendre l'élimination de (1-36) et (1-39) des variables résistives non-désirées, v_{Rt} et i_{Rl}, en remplaçant (1-40) par (1-54). Les matrices A et B se calculent à partir des mêmes matrices que dans le cas précédent, mais les matrices intermédiaires deviennent (1-57) et (1-58).

$$\begin{aligned} \tilde{Y} &= -M_9 Q'_{CR} & \tilde{H} &= -Q_{CL} + M_{10} Q_{RL} \\ \tilde{F} &= M_6 Q_{RL} & \tilde{G} &= Q'_{CL} + M_5 Q'_{CR} \\ \hat{Y} &= -M_9 Q'_{ER} & \hat{H} &= -Q_{CJ} + M_{10} Q_{RJ} \\ \hat{F} &= M_6 Q_{RJ} & \hat{G} &= Q'_{EL} + M_5 Q'_{ER} \end{aligned} \tag{1-57}$$

I_1 = matrice identité, même dimension que G_{ll}

I_2 = matrice identité, même dimension que G_{tt}

$$\begin{aligned} M_1 &= I_1 + G_{lt} Q_{RR} & M_6 &= -M_4 (M_3 G_{lt} - G_{tt}) \\ M_2 &= I_2 - G_{tl} Q'_{RR} & M_7 &= G_{ll} Q'_{RR} (M_2)^{-1} \\ M_3 &= G_{tt} Q_{RR} (M_1)^{-1} & M_8 &= Q_{CR} (M_1 + M_7 G_{tt} Q_{RR})^{-1} \\ M_4 &= Q'_{RL} (M_2 + M_3 G_{ll} Q'_{RR})^{-1} & M_9 &= -M_8 (M_7 G_{tl} + G_{ll}) \\ M_5 &= -M_4 (M_3 G_{ll} + G_{tl}) & M_{10} &= M_8 (M_7 G_{tt} + G_{lt}) \end{aligned} \tag{1-58}$$

L'existence d'une représentation d'état pour un système comprenant un transformateur dépend du fait que plusieurs matrices doivent être non-singulières (calcul de M_3, M_4, M_7 et M_8 dans (1-58)). Contrairement au cas où il n'y a pas de transformateur, il n'est pas possible de prouver de façon certaine que ces matrices sont non-singulières. Si l'une d'elle est singulière, cela ne veut pas nécessairement dire qu'il n'existe pas de

représentation d'état pour le système. Cela signifie plutôt qu'il n'est pas possible de trouver une représentation d'état avec la procédure décrite ici.

Les matrices intermédiaires obtenues pour l'arbre de la figure 1-7 sont montrées en (1-59). Les matrices d'état *A* et *B* qui en découlent sont montrées en (1-60).

$$
\begin{aligned}
M_1 &= \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ -N & 0 & -N & 1 \end{bmatrix} \qquad M_7 = \begin{bmatrix} \frac{1}{R_{sn}} & 0 & \frac{1}{R_{dio}} & 0 \end{bmatrix} \qquad M_{10} = \begin{bmatrix} 0 \\ 0 \end{bmatrix} \\
M_2 &= \begin{bmatrix} 1 \end{bmatrix} \qquad M_8 = \begin{bmatrix} -1 & 0 & 0 & 0 \\ -1 & 1 & -1 & 0 \end{bmatrix} \\
M_3 &= \begin{bmatrix} 0 & 0 & 0 & 0 \end{bmatrix} \qquad M_9 = \begin{bmatrix} \frac{1}{R_{sn}} & 0 & 0 & \frac{N}{R_{sn}} \\ \frac{1}{R_{sn}} & \frac{-1}{R_{ch}} & \frac{1}{R_{dio}} & \frac{N}{R_{sn}} + \frac{N}{R_{dio}} \end{bmatrix} \\
M_4 &= M_6 = \begin{bmatrix} 0 \end{bmatrix} \\
M_5 &= \begin{bmatrix} 0 & 0 & 0 & 0 \end{bmatrix}
\end{aligned}
\tag{1-59}
$$

$$
A = \begin{bmatrix} \frac{-1}{C_{sn}R_{sn}} & \frac{-1}{C_{sn}R_{sn}} & 0 \\ \frac{-1}{C_{ch}R_{sn}} & \frac{\frac{-1}{R_{sn}} - \frac{1}{R_{ch}} - \frac{1}{R_{dio}}}{C_{ch}} & \frac{-1}{C_{ch}} \\ 0 & \frac{1}{L_{ch}} & 0 \end{bmatrix} \qquad B = \begin{bmatrix} \frac{N}{C_{sn}R_{sn}} \\ \frac{\frac{N}{R_{sn}} + \frac{N}{R_{dio}}}{C_{ch}} \\ 0 \end{bmatrix} \tag{1-60}
$$

1.3.4 Matrices d'état *C* et *D*

Étant donné que l'ajout des transformateurs affecte essentiellement les matrices de résistances et de conductances du système, seules les sorties tension aux bornes de bran-

ches résistives et courant des liens résistifs sont affectées. On définit d'abord six matrices intermédiaires comme suit.

$$\begin{aligned} D_1 &= (M_2 + M_3 G_{ll} Q'_{RR})^{-1} \\ D_2 &= (M_1 + M_7 G_{tt} Q_{RR})^{-1} \\ M_{11} &= D_1(-M_3 G_{ll} + G_{tl}) \\ M_{12} &= D_1(M_3 G_{lt} - G_{tt}) \\ M_{13} &= D_2(M_7 G_{tl} + G_{ll}) \\ M_{14} &= D_2(-M_7 G_{tt} - G_{lt}) \end{aligned} \tag{1-61}$$

L'équation (1-42) demeure valide. Cependant, (1-43) devient (1-62).

$$\begin{aligned} S_{11} &= M_{11} Q'_{CR} & S_{21} &= M_{13} Q'_{CR} \\ S_{12} &= M_{12} Q_{RL} & S_{22} &= M_{14} Q_{RL} \\ T_{11} &= M_{11} Q'_{ER} & T_{21} &= M_{13} Q'_{ER} \\ T_{12} &= M_{12} Q_{RJ} & T_{22} &= M_{14} Q_{RJ} \end{aligned} \tag{1-62}$$

1.3.5 Restrictions sur les paramètres

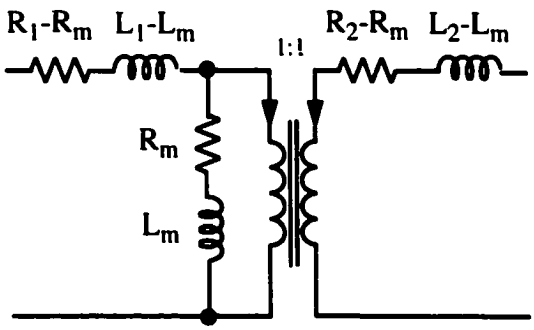
La technique de modélisation décrite ci-haut permet d'inclure dans un système un transformateur idéal, ce qui n'est pas le cas de la méthode qui a été utilisée dans le Power System Blockset version 1. Dans le cas de ce logiciel, les résistances de chaque enroulement et la résistance de la branche de magnétisation devaient avoir des valeurs finies non-nulles. Il était cependant permis de spécifier des inductances de fuite nulles et une inductance de magnétisation infinie.

Il y a un seul cas où l'on ne peut modéliser des enroulements idéaux avec la méthode proposée. Il s'agit du cas où les enroulements secondaires de plusieurs transformateurs forment une maille, par exemple dans une connexion triphasée en triangle. Comme tous les enroulements secondaires doivent faire partie de l'arbre, selon la convention établie plus haut, un arbre ne peut être construit dans ces conditions, puisque l'arbre ne doit contenir aucune maille. Il est donc nécessaire dans ce cas de spécifier une impédance de fuite (résistance, inductance ou les deux) non-nulle dans chacun des enroulements secondaires.

La technique de modélisation présentée ici permet aussi de modéliser un transformateur réel, avec impédances de fuites et de magnétisation. Ces impédances sont simplement considérées comme les autres composantes passives du système.

1.4 Circuits contenant des inductances couplées

Originalement, nous traitions une inductance mutuelle comme un transformateur ayant un rapport de transformation de un, tel qu'illustré à la figure 1-8.



R_1: résistance de l'enroulement 1.
R_2: résistance de l'enroulement 2.
R_m: résistance de la branche magnétisante.
L_1: self-inductance de l'enroulement 1.
L_2: self-inductance de l'enroulement 2.
L_m: inductance magnétisante.

Figure 1-8 Circuit équivalent d'une inductance mutuelle à deux enroulements.

Cependant, nous avons étudié certaines topologies où cette méthode donnait des résultats erronés. Nous avons donc éliminé le transformateur idéal et généré des matrices d'inductances et de résistances avec des termes hors-diagonale aux bons endroits. Pour l'exemple de la figure 1-8, les matrices suivantes sont obtenues.

$$L = \begin{bmatrix} L_1 & L_m \\ L_m & L_2 \end{bmatrix} \qquad R = \begin{bmatrix} R_1 & R_m \\ R_m & R_2 \end{bmatrix} \tag{1-63}$$

1.4.1 Matrices affectées

Comme il peut maintenant y avoir du couplage entre plusieurs inductances, les matrices L_{tt} et L_{ll} ne sont plus nécessairement diagonales. De plus, comme certaines de ces inductances peuvent être des branches alors que d'autres sont des liens, il y aura potentiellement du couplage entre les branches et les liens. La relation (1-37) ne tient plus et est remplacée par (1-64).

$$\begin{bmatrix} v_{Lll} \\ v_{Llt} \end{bmatrix} = \begin{bmatrix} L_{ll} & L_{lt} \\ L_{tl} & L_{tt} \end{bmatrix} \frac{d}{dt} \begin{bmatrix} i_{Lll} \\ i_{Llt} \end{bmatrix} \tag{1-64}$$

Nous utilisons les même relations résistives (1-54) que dans le cas où il y avait des transformateurs. Enfin, si l'on reprend le processus d'élimination des variables inductives non-désirées, on se retrouve avec la matrice d'inductances $\underline{L}$ suivante.

$$\underline{L} = L_{ll} - L_{lt}Q_{LL} - Q'_{LL}L_{tl} + Q'_{LL}L_{tt}Q_{LL} \tag{1-65}$$

Outre cette nouvelle définition de la matrice $\underline{L}$, le calcul des matrices d'état A et B demeure le même que dans le cas des circuits contenant des transformateurs. Le proces-

sus d'élimination des variables inductives non-désirées modifie également le calcul des parties des matrices d'état C et D associées aux sorties de tension aux bornes des inductances qui sont des branches de l'arbre, qui devient le suivant.

$$V_{Ltt} = \begin{bmatrix} V_{11} & V_{12} \end{bmatrix} \begin{bmatrix} v_{Ct} \\ i_{Lll} \end{bmatrix} + \begin{bmatrix} W_{11} & W_{12} \end{bmatrix} \begin{bmatrix} E \\ J \end{bmatrix}$$

$$t_3 = (L_{tl} - L_{tt} Q_{LL}) \underline{L}^{-1} \tag{1-66}$$

$$V_{11} = t_3 \tilde{G} \qquad W_{11} = t_3 \hat{G}$$

$$V_{12} = -t_3 \tilde{F} \qquad W_{12} = -t_3 \hat{F}$$

1.4.2 Restrictions sur les paramètres

Comme dans le cas des transformateurs, la méthode proposée permet d'assouplir certaines contraintes sur les paramètres, par rapport à la version 1 du Power System Blockset, où il n'était pas possible de spécifier des résistances nulles. De plus, l'inductance de magnétisation L_m devait aussi être non-nulle. La méthode de modélisation proposée ici permet de spécifier des résistances nulles et une inductance de magnétisation nulle. Ce dernier cas revient en fait à des inductances non-couplées.

1.5 Détection d'erreurs dans les données d'entrée

La modélisation des systèmes électriques basée sur la théorie des graphes permet également de détecter plusieurs types d'erreurs courantes. Par exemple, si lors de la construction de l'arbre une source de tension ne peut être incluse dans ce dernier, c'est qu'il y a une maille ne comportant que des sources de tension. En d'autres termes, il y a des sources de tension en parallèle, ce qui viole la loi de Kirchhoff des tensions. De façon analogue, si une source de courant doit être incluse dans l'arbre pour atteindre un certain noeud, c'est que nous sommes en présence de l'un des deux problèmes suivants:

a. la source de courant n'est pas raccordée à au moins une de ses extrémités;
b. il y a au moins deux sources de courant en série.

Dans un cas comme dans l'autre, la loi de Kirchhoff des courants est violée et une erreur doit être signalée. Enfin, nous avons mentionné plus haut que les situations suivantes provoquaient des dégénérescences non-tolérées:

a. mailles composées uniquement de sources de tension et de condensateurs (mailles *CE*);
b. coupures composées uniquement de sources de courant et d'inductances (coupures *LJ*).

Ces deux conditions sont détectées grâce à la matrice des coupures fondamentales Q_f. En effet, si l'on détecte un terme non-nul dans la sous-matrice Q_{EC} de (1-20), c'est qu'il y a une maille *CE*. Comme chaque rangée de Q_{EC} correspond à une source de tension particulière et chaque colonne de cette même matrice correspond à un condensateur en particulier, il est possible d'indiquer à l'usager quelle source de tension et quel condensateur causent cette dégénérescence.

De la même façon, un terme non-nul dans la sous-matrice Q_{LJ} de (1-20) indique la présence d'une coupure *LJ*. Encore une fois, nous connaissons les composantes qui causent le problème et elles peuvent être indiquées à l'usager.

1.6 Analyse de performance

Afin de confirmer la validité de la méthodologie proposée dans ce chapitre, les divers algorithmes ont été programmés en langage Matlab et des tests ont été effectués sur un grand nombre de circuits. Dans un premier temps, les données d'entrée devaient être fournies sous forme d'une matrice, où chaque rangée correspondait à une compo-

sante particulière et chaque colonne correspondait à un paramètre de la composante, soit entre autres:

a. un numéro de segment unique;
b. un noeud de source et un noeud de destination;
c. un type de composante (source de tension=1, condensateur=2, etc);
d. la valeur de la composante (R=10 Ω, C=4.7 μF, etc).

Cette façon de procéder convenait lorsque les systèmes à analyser étaient relativement petits, mais au fur et à mesure qu'ils grossissaient, l'entrée des données devenait laborieuse. Un filtre a donc été créé afin de permettre l'entrée des données à partir du Power System Blockset. Ce logiciel permet d'entrer graphiquement les données du circuit et les convertit en plusieurs matrices avant de faire l'analyse. Ce format étant incompatible avec celui requis pour notre méthode, une conversion s'avérait nécessaire, d'où le développement du filtre sus-mentionné.

Parmi les nombreux systèmes sur lesquels nos algorithmes ont été testés, il y en a un qui est particulièrement gros. Ce système comprend tout le réseau de transport d'un état américain avec des connexions inter-états. Il comprend 261 variables d'état, 147 entrées et 192 sorties. Le grand nombre d'entrées et de sorties est principalement dû au fait que le système comprend 17 lignes de transport à paramètres distribués, lesquelles sont modélisées par des sources de courant non-linéaires alimentées en tension. Chaque ligne triphasée requiert six sorties (trois tensions de phase à chaque extrémité de la ligne) et six entrées (trois courants de ligne à chaque extrémité). Ce réseau comporte aussi 48 transformateurs monophasés, dont 30 ont deux enroulements et 18 en ont trois. Enfin, le système comprend aussi 17 inductances mutuelles triphasées.

Au départ, le seul critère de performance de nos programmes était le temps requis pour obtenir la représentation d'état du système. On souhaite que ce temps soit le plus

petit possible. La principale raison pour laquelle nous avons choisi le réseau décrit ci-haut pour nos essais est que ce réseau est très long à analyser avec la version 1 du Power System Blockset, même sur des ordinateurs très performants. La nouvelle méthode permet de réduire radicalement le temps de calcul, comme le montre le tableau 1-1. Ces essais ont été réalisés sur un ordinateur SUN Ultra 10 équipé de 256 Mo de mémoire vive et dont le processeur UltraSparc IIi comporte une horloge de 333 MHz. Bien que les trois parties des calculs soient plus rapides qu'avant, notre travail affecte seulement la partie 2, le calcul des matrices d'état, du tableau 1-1. Le temps de calcul de cette partie passe de 6h40 à moins de six secondes, soit un gain en vitesse d'un facteur de plus de 4000.

Tableau 1-1

Temps requis pour obtenir les matrices d'état d'un grand réseau

Tâche	Temps requis avec PSB 1.0 (s)	Temps requis avec nouvelle méthode (s)
1 - Analyse graphique et conversion des données en matrices	360	28
2 - Calcul des matrices d'état *A*, *B*, *C* et *D*	24 200	5.8
3 - Calcul des conditions initiales permettant de démarrer la simulation en régime permanent	11.0	7.6

À la suite de ces essais, nous avons également conclu que la méthode proposée permet d'obtenir des matrices d'état beaucoup plus "propres" et creuses. En effet, la méthode employée pour le calcul de ces matrices dans la version 1 du Power System Blockset cause beaucoup d'erreurs numériques, de sorte que les matrices contiennent un grand nombre de termes négligeables. Pour comparer entre elles les diverses matrices

d'état obtenues avec différentes méthodes de calcul, nous considérons comme références les matrices obtenues avec la méthode proposée. En d'autres termes, tout élément qui est nul dans une matrice obtenue avec la méthode proposée et qui est non-nul dans la matrice équivalente obtenue avec une autre méthode est considéré comme un terme parasite. La figure 1-9 permet de constater que pour la matrice A, d'ordre 261 et pouvant contenir au maximum 68 000 termes, la méthode proposée donne une matrice creuse à plus de 93% alors que la matrice obtenue avec le PSB 1.0 contient presque dix fois plus de termes.

Le plus petit terme non-nul obtenu avec la méthode proposée est de l'ordre de 10^{-5}. La même matrice obtenue avec le PSB 1.0 contient plus de 25 000 termes inférieurs à cette valeur et en compte près de 16 000 qui sont supérieurs à ce seuil, ce qui est près de quatre fois plus qu'avec la méthode proposée. Le tableau 1-2 montre les taux de remplissage pour les quatre matrices d'état, en comparant les deux méthodes d'obtention des équations d'état. On y remarque que la méthode proposée permet de réduire substantiellement les termes parasites dans les quatre matrices.

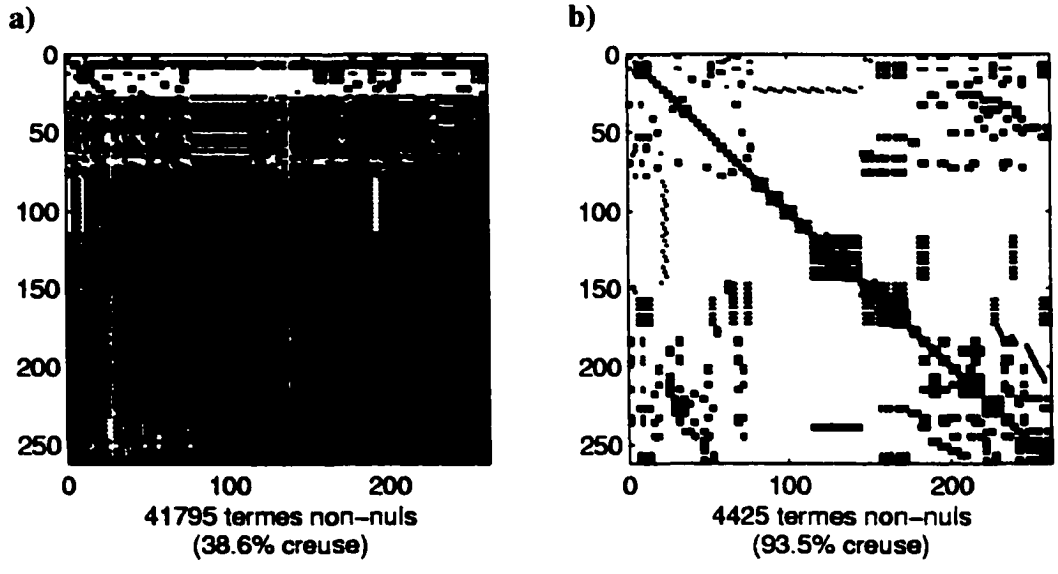


Figure 1-9 Patron de remplissage de la matrice d'état A. **a)** avec le PSB 1.0; **b)** avec la méthode proposée.

Tableau 1-2

Remplissage des matrices d'état selon la méthode de calcul.

Matrice	Nombre maximum d'éléments	Nombre de termes non-nuls	
		PSB 1.0	Méthode proposée
A	68 121	41 795	4 425
B	38 367	13 326	1 596
C	50 112	32 711	2 010
D	28 224	12 612	1 049

1.7 Conclusions

Ce chapitre décrit une méthodologie permettant d'obtenir systématiquement la représentation d'état d'un système électrique quelconque. Bien que la théorie et les algorithmes soient en grande partie connus et documentés, nous considérons que la combinaison de ces algorithmes et leur programmation, le tout résultant en un logiciel fonctionnel, est une contribution intéressante.

Étant donné les bonnes performances de cette méthode, les travaux décrits dans le présent chapitre ont été intégrés à la version 2.0 du Power System Blockset, commercialement disponible depuis l'été 2000.

Enfin, l'obtention de la représentation d'état d'un système électrique général n'est que la première étape dans nos travaux. Nous allons maintenant considérer plus spécifiquement une classe de systèmes électriques, à savoir les entraînements électriques.

CHAPITRE 2

MODÉLISATION DES ENTRAÎNEMENTS ÉLECTRIQUES À L'AIDE DE LA REPRÉSENTATION D'ÉTAT

Le chapitre précédent décrit une méthodologie permettant d'obtenir la représentation d'état de la partie linéaire d'un système électrique quelconque. Dans notre développement, nous avons très peu parlé des éléments non-linéaires présents dans de nombreux systèmes. En fait, à date, nous avons seulement mentionné que les modèles de lignes à paramètres distribués étaient modélisés par des sources de courant commandées en tension.

Cette approche de modélisation est celle employée dans le Power System Blockset pour toutes les composantes non-linéaires, incluant:

a. tous les types d'interrupteurs (disjoncteur, diode, thyristor, etc.);
b. tous les types de machines électriques (synchrone, asynchrone, etc.);
c. les lignes de transport à paramètres distribués;
d. le parafoudre;
e. la branche de magnétisation du transformateur saturable.

Dans le cadre de nos travaux, nous avons conservé une approche similaire. Le présent chapitre vise à décrire de façon plus spécifique comment nous modélisons les composantes non-linéaires présentes dans des entraînements électriques. Nous commençons par considérer les systèmes électriques comportant des interrupteurs. Ensuite, nous ajou-

tons la possibilité d'inclure les machines électriques dans ces mêmes systèmes et nous décrivons le modèle de machine asynchrone utilisé dans nos travaux. Enfin, nous décrivons une méthode qui permet d'obtenir une représentation d'état unique pour toute la partie électrique d'un entraînement, incluant la machine. Les performances de cette méthode sont analysées en temps différé.

2.1 Systèmes comprenant des interrupteurs

Nous considérons d'abord les systèmes électriques comportant des interrupteurs. Le travail décrit dans cette section s'applique à tous les types d'interrupteurs, puisque le macromodèle de la partie électrique d'un interrupteur demeure le même, qu'il s'agisse d'un disjoncteur, d'une diode, d'un transistor MOSFET ou autre. Seule la logique de commutation varie d'un type d'interrupteur à l'autre.

En premier lieu, nous discutons de la modélisation des interrupteurs. Nous décrivons ensuite une méthode innovatrice récemment mise au point pour mettre à jour les matrices d'état suite à la commutation d'interrupteurs.

2.1.1 Modélisation des interrupteurs

Il existe essentiellement deux grandes familles de modèles d'interrupteurs, soit les micromodèles et les macromodèles. Dans le premier cas, il s'agit de représenter fidèlement le comportement statique et dynamique de l'interrupteur. Les micromodèles sont principalement utilisés par les concepteurs de circuits électroniques et le niveau de détail avec lequel ils sont modélisés rend les calculs associés à leur simulation très longs. Cependant, lorsque le comportement détaillé de l'interrupteur lui-même importe peu et que l'on veut plutôt étudier le comportement d'un système comprenant des interrupteurs, un macromodèle suffit [21]. Nous considérons trois types de macromodèles d'interrupteurs, soit:

a. l'interrupteur idéal (impédance nulle à l'état passant, impédance infinie à l'état ouvert);
b. le modèle binaire résistif (faible résistance à l'état passant, très grande résistance à l'état ouvert);
c. le modèle binaire résistif avec élément(s) réactif(s).

L'interrupteur idéal est utilisé entre autres dans le logiciel EMTP [22], lequel est basé sur l'approche nodale et la discrétisation trapézoïdale. L'approche nodale se prête bien à la simulation d'interrupteurs idéaux puisque quand l'interrupteur est fermé, les deux noeuds auxquels il est raccordé deviennent un seul noeud et on retire une rangée et une colonne de la matrice d'admittance nodale. Cependant, la modélisation d'un interrupteur idéal n'est pas possible lorsque le système électrique est exprimé sous forme d'une représentation d'état puisqu'une résistance nulle conduit à une conductance infinie (et vice-versa) et les matrices R et G deviennent singulières. Nous sommes donc contraints d'utiliser au minimum un modèle d'interrupteur consistant en une résistance binaire. Quant au modèle binaire résistif avec composantes réactives, il s'agit d'ajouter dans le modèle un certain nombre de résistances, d'inductances et de condensateurs [7] afin de représenter plus fidèlement les caractéristiques de l'interrupteur à l'allumage et à l'extinction.

Nous avons utilisé deux de ces modèles dans nos travaux, à savoir le modèle binaire résistif et le modèle binaire résistif avec inductance en série. Ces deux modèles sont illustrés à la figure 2-1. Nous avons d'abord utilisé le modèle avec inductance en

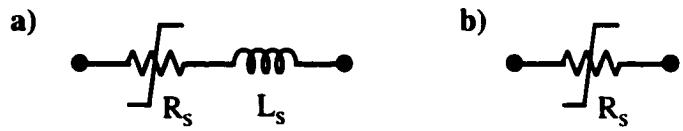


Figure 2-1 Macromodèles d'interrupteurs utlisés. **a)** modèles binaire résistif avec inductance série; **b)** modèle binaire résistif.

série. Cette inductance est typiquement de l'ordre du µH. L'inclusion de cette inductance dans le modèle se justifie de diverses façons. Dans le cas du modèle employé dans le Power System Blockset version 1, cette inductance sert à briser la boucle algébrique [23] formée lorsque le modèle d'interrupteur, simulé à l'extérieur de la représentation d'état, est placé dans une boucle de contre-réaction avec cette dernière, tel qu'illustré à la figure 2-2.

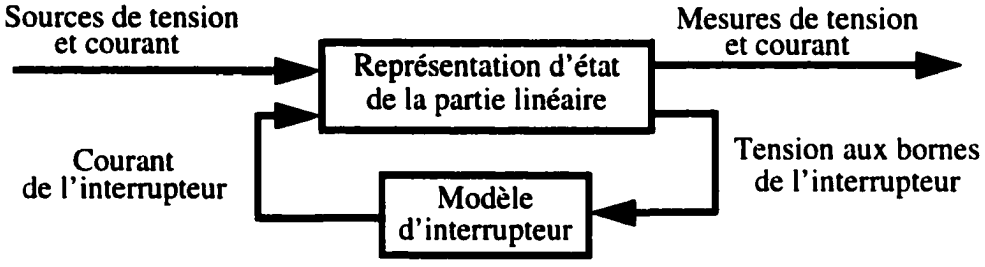


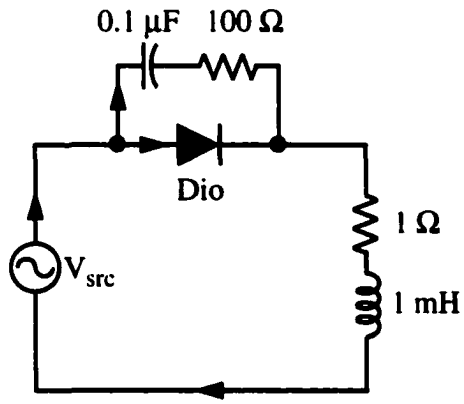
Figure 2-2 Interface de la partie linéaire du système avec le modèle d'interrupteur.

L'ajout de cette inductance peut aussi se justifier par le fait que l'on cherche à modéliser la caractéristique de l'interrupteur à l'allumage. En effet, lorsqu'un semiconducteur de puissance passe de l'état bloqué à l'état passant, le courant ne s'établit pas instantanément, mais croit graduellement comme dans une résistance en série avec une inductance.

Cependant, l'ajout de l'inductance dans le modèle a pour effet d'augmenter le nombre de variables d'état, et par conséquent la taille des matrices d'état. Pour un circuit comprenant seulement quelques interrupteurs, cette augmentation peut être sans conséquence. Cependant, pour des systèmes comportant un grand nombre d'interrupteurs, par exemple un système de transport d'énergie à courant continu, l'impact est loin d'être négligeable. Une autre conséquence de l'inclusion de l'inductance dans le modèle est qu'elle rend le système très rigide. Un système est dit rigide lorsqu'il comporte à la fois

des dynamiques très lentes et très rapides. Lorsqu'un système électrique est exprimé sous forme d'équations d'état, les valeurs propres de la matrice d'état *A* sont en fait les pôles du système. Le rapport, en valeur absolue, du pôle avec la plus grande valeur sur le pôle avec la plus petite valeur donne le coefficient de rigidité [20].

Pour illustrer le concept de rigidité, considérons le circuit simple illustré à la figure 2-3. Il s'agit d'un redresseur à diode simple alternance alimentant une charge inductive. Un circuit amortisseur, consistant en une résistance en série avec un condensateur, est raccordé aux bornes de l'interrupteur.



Diode: L = 10 µH (modèle inductif seulement)
R = 1 mΩ (interrupteur ouvert)
R = 1 MΩ (interrupteur fermé)

Figure 2-3 Circuit simple utilisé pour illustrer le concept de rigidité.

Les matrices d'état ont été calculées pour ce circuit dans quatre conditions différentes, soit en utilisant alternativement les deux modèles d'interrupteurs et dans chaque cas, avec l'interrupteur ouvert (résistance très élevée) puis fermé (résistance faible). Les valeurs propres de la matrice d'état *A* ont été calculées et le coefficient de rigidité obtenu. Le tableau 2-1 résume les résultats. En comparant les deux modèles pour un même état de l'interrupteur, on voit que le fait d'inclure une petite inductance série dans le modèle augmente substantiellement le coefficient de rigidité du système.

Tableau 2-1

Coefficients de rigidité obtenus dans diverses conditions

Modèle d'interrupteur	État de l'interrupteur	Coefficient de rigidité
Binaire résistif	ouvert	1
	fermé	100
Binaire résistif avec inductance série	ouvert	1 000 000
	fermé	10 000

Nous verrons au chapitre suivant qu'un système caractérisé par un coefficient de rigidité élevé pose certains problèmes au niveau de l'intégration numérique des équations.

2.1.2 Changements d'état d'interrupteurs

Nous avons décrit au chapitre précédent une méthode permettant d'obtenir la représentation d'état de la partie linéaire d'un système électrique. Nous avons aussi discuté à la section précédente de deux modèles d'interrupteurs. Nous allons maintenant voir comment le modèle d'interrupteur est interconnecté au reste du système.

Un interrupteur peut être simulé de diverses façons lorsque le système est modélisé sous forme d'équations d'état. Nous avons déjà discuté de la façon dont les modèles non-linéaires sont simulés dans la version 1 du Power System Blockset (figure 2-2). Une autre façon de faire serait de modifier la valeur de la résistance d'un interrupteur directement dans la matrice des résistances et de reprendre le calcul des matrices d'état de la même manière que le calcul initial (chapitre 1). Or, le nombre de calculs impliqués rend cette méthode peu attrayante pour une simulation en temps réel.

Une méthode combinant ces deux approches a récemment été développé par Gilbert Sybille, un chercheur de l'Institut de Recherche d'Hydro-Québec (IREQ). Nous avons participé à l'implantation et à l'optimisation de cette méthode dans la version 2 du Power System Blockset et allons maintenant la décrire.

D'abord, cette méthode n'est applicable que pour des systèmes où chaque interrupteur est modélisé comme une simple résistance binaire, sans inductance. En conservant la convention établie plus tôt, soit que chaque interrupteur est une source de courant alimenté en tension, on définit la relation suivante.

$$i_{sw} = G_{sw} v_{sw} \tag{2-1}$$

Dans (2-1), i_{sw} est le vecteur des courants circulant dans les n_{sw} interrupteurs, G_{sw} est une matrice diagonale dont les éléments représentent les conductances des interrupteurs, et v_{sw} est le vecteur des tensions aux bornes des n_{sw} interrupteurs. Fait à signaler, cette façon de modéliser les interrupteurs permet d'utiliser une résistance infinie (conductance nulle) pour un interrupteur à l'état bloqué. Les tensions v_{sw} forment une partie du vecteur des sorties y de la représentation d'état de la partie linéaire du système. Le reste du vecteur y contient différentes mesures de courants et tensions, que nous identifions par y_{mes}. Les courants i_{sw} forment quant à eux une partie du vecteur d'entrée u de cette même représentation d'état. Le reste du vecteur u contient les sources indépendantes de tension et de courant présentes dans le système. Ces sources sont identifiées par u_{src}. Notre système peut alors être défini tel qu'illustré à la figure 2-4, où les matrices d'état ont un suffixe o pour indiquer qu'il s'agit des matrices d'état originales du système. Il s'agit ensuite de développer les expressions définissant une nouvelle représentation d'état équivalente au schéma de la figure 2-4, tel qu'illustré à la figure 2-5.

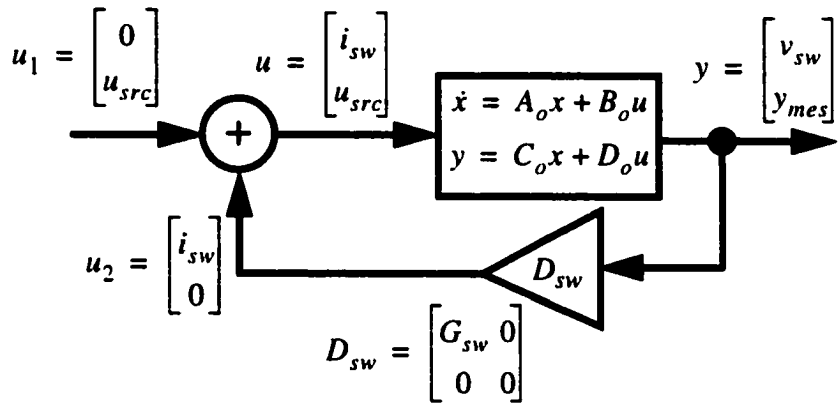


Figure 2-4 Schéma-bloc illustrant la méthode de mise à jour des matrices d'état.

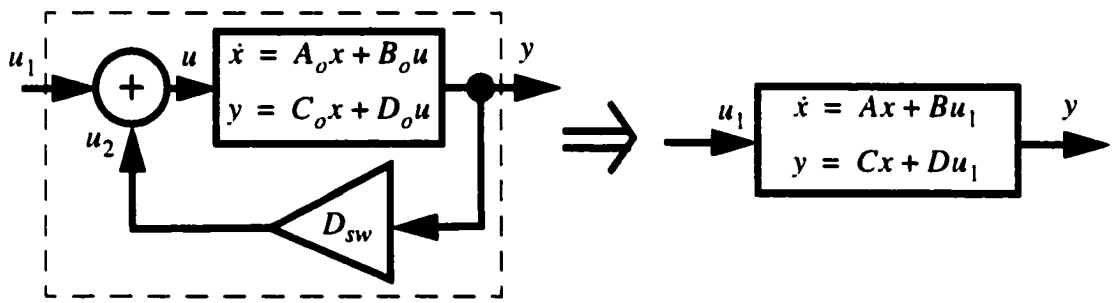


Figure 2-5 Représentation d'état équivalente (matrices A, B, C et D).

Les relations suivantes servent de base pour obtenir les expressions des nouvelles matrices d'état.

$$\dot{x} = A_o x + B_o u \tag{2-2}$$

$$y = C_o x + D_o u \tag{2-3}$$

$$u_2 = D_{sw} y \tag{2-4}$$

$$\begin{aligned} u &= u_1 + u_2 \\ &= u_1 + D_{sw} y \end{aligned} \tag{2-5}$$

En substituant (2-5) dans (2-3), on obtient (2-6).

$$y = C_o x + D_o u_1 + D_o D_{sw} y \tag{2-6}$$

En isolant y dans (2-6), et en ré-arrangeant les termes, on obtient les matrices C et D de la nouvelle représentation d'état. La matrice I dans (2-8) est une matrice identité de dimension appropriée.

$$y = Cx + Du_1 \tag{2-7}$$

$$D_x = (I - D_o D_{sw})^{-1} \tag{2-8}$$

$$C = D_x C_o \tag{2-9}$$

$$D = D_x D_o \tag{2-10}$$

Pour le calcul des matrices A et B, il s'agit d'abord de substituer (2-5) et (2-7) dans (2-2), puis d'arranger les termes. Le résultat est le suivant:

$$\dot{x} = Ax + Bu_1 \tag{2-11}$$

$$B_x = B_o D_{sw} \tag{2-12}$$

$$A = A_o + B_x C \tag{2-13}$$

$$B = B_o + B_x D \tag{2-14}$$

La mise à jour des matrices d'état A, B, C et D se résume donc de la façon suivante, lorsqu'un ou plusieurs interrupteurs changent d'état:

a. mettre à jour la matrice D_{sw}, qui contient sur sa diagonale les conductances de tous les interrupteurs;
b. calculer (2-8), (2-9) et (2-10);
c. calculer (2-12), (2-13) et (2-14).

2.2 Machines électriques

Nous abordons maintenant la modélisation de la composante centrale dans un entraînement, soit la machine électrique. Comme tous nos travaux ont été effectués avec une machine asynchrone ou machine d'induction, nous allons discuter uniquement de cette machine. Après avoir présenté le modèle utilisé et justifié notre choix de variables d'état, nous discutons de l'impact du choix de la transformation de référentiel sur la stabilité de la simulation des entraînements à haute fréquence de commutation, toujours en temps différé.

2.2.1 Modélisation de la machine asynchrone

La modélisation de la machine asynchrone est bien documentée dans la littérature. Cependant, il existe un grand nombre de modèles. Le niveau de détail du modèle dépend de l'application que l'on désire simuler. Par exemple, pour un manufacturier de machines, il est important de faire des simulations d'une grande précision si l'on souhaite reproduire fidèlement le comportement de la machine dans toutes les conditions. D'autre part, si l'on désire étudier l'impact de la dynamique d'une machine placée dans un grand réseau électrique, il n'est pas nécessaire de la simuler de façon très détaillée. Ceci dit, notre choix de modèle est dicté par le fait que l'on désire simuler cette machine en temps réel, ce qui impose une contrainte sévère au niveau du temps disponible pour résoudre les équations associées à notre modèle. Comme nous nous intéressons aux entraînements, nous souhaitons simuler le plus fidèlement possible le comportement d'une machine

réelle, tout en respectant la contrainte de temps de calcul. Nous avons donc choisi d'utiliser un modèle très semblable à celui développé dans le cadre du Power System Blockset.

Ce modèle utilise des transformations de référentiel afin de convertir les variables de phase en variables dans le référentiel *dq*. L'utilisation de transformations de référentiel est principalement motivée par le fait qu'elle simplifie grandement les équations de la machine. En effet, les matrices d'inductances de la machine varient dans le temps lorsque les variables de phase sont utilisées (référentiel *abc*). Certaines inductances sont des fonctions du sinus ou du cosinus de l'angle rotorique. Les transformations dans le référentiel *dq*, judicieusement choisies, rendent ces inductances constantes. Ces transformations sont cependant applicables uniquement si les enroulements de la machine sont équilibrés et distribués sinusoïdalement. Pour le moment, nous allons développer les équations de la machine dans le référentiel *dq* arbitraire et nous discuterons plus en détail des transformations de référentiel dans une section ultérieure.

Notre modèle est basé sur les hypothèses suivantes [24]:

a. l'entrefer est uniforme;
b. le circuit magnétique est linéaire (pas de saturation magnétique);
c. les enroulements du stator sont identiques et distribués de telle sorte qu'ils produisent une force magnétomotrice purement sinusoïdale, avec les phases disposées de telle sorte qu'une seule force magnétomotrice tournante est établie lorsque les courants du stator sont équilibrés;
d. les enroulements ou barres du rotor sont disposés de telle sorte que, à tout instant, la force magnétomotrice du rotor peut être considérée comme purement sinusoïdale et comportant le même nombre de pôles que la force magnétomotrice du stator;
e. les enroulements du rotor et du stator sont raccordés en étoile et le neutre n'est pas accessible (connexion à trois fils).

Une caractéristique importante du modèle est la nature des variables d'état. Dans la plupart des modèles de machines électriques, les variables d'état sont généralement les courants circulant dans les divers enroulements de la machine ou les flux de ces mêmes enroulements. Krause et Thomas [24] mentionnent que le choix devrait être dicté par le système d'équations qui requiert le moins de calcul. Selon eux, ceci se produit lorsque les flux sont les variables d'état. Cependant, leur motivation à réduire les calculs provient du fait que leurs simulations sont faites sur des ordinateurs analogiques. Dans ce cas, un calcul plus complexe signifie un plus gros ordinateur analogique. Dans notre cas, nous verrons à la prochaine section que les équations de notre modèle de machine sont insérées dans un système d'équations plus général. Avec cette façon de procéder, les courants de la machine servent de variables d'interface et doivent être les variables d'état.

Les équations des tensions dans le référentiel *dq* arbitraire pour notre modèle sont données par (2-15). Les équations des flux sont quant à elles données par (2-16). Le schéma équivalent correspondant à ces équations est montré à la figure 2-6 [25].

$$\begin{aligned} v_{qs} &= r_s i_{qs} + \frac{d}{dt}\lambda_{qs} + \omega\lambda_{ds} \\ v_{ds} &= r_s i_{ds} + \frac{d}{dt}\lambda_{ds} - \omega\lambda_{qs} \\ v'_{qr} &= r'_r i'_{qr} + \frac{d}{dt}\lambda'_{qr} + (\omega - \omega_r)\lambda'_{dr} \\ v'_{dr} &= r'_r i'_{dr} + \frac{d}{dt}\lambda'_{dr} - (\omega - \omega_r)\lambda'_{qr} \end{aligned} \tag{2-15}$$

$$\begin{aligned} \lambda_{qs} &= L_s i_{qs} + L_m i'_{qr} \\ \lambda_{ds} &= L_s i_{ds} + L_m i'_{dr} \\ \lambda'_{qr} &= L'_r i'_{qr} + L_m i_{qs} \\ \lambda'_{dr} &= L'_r i'_{dr} + L_m i_{ds} \\ L_s &= L_{ls} + L_m \\ L'_r &= L'_{lr} + L_m \end{aligned} \tag{2-16}$$

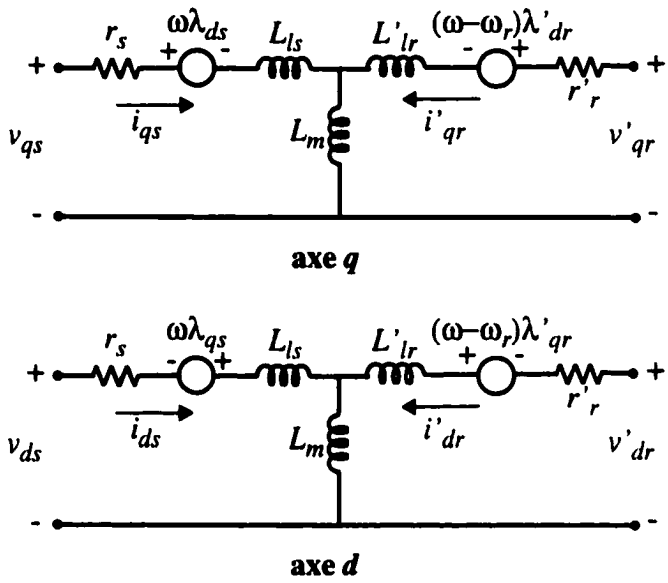


Figure 2-6 Schéma équivalent du modèle de machine asynchrone dans le référentiel *dq* arbitraire.

Dans (2-15), (2-16) et la figure 2-6, on remarque d'abord qu'il n'y a pas de composante homopolaire, conséquence du raccordement en étoile à trois fils. Les suffixes *s* et *r* identifient les grandeurs liées respectivement au stator et au rotor. Les suffixes *l* et *m* identifient respectivement les inductances de fuite (de l'anglais "leakage") et de magnétisation. Les "primes" servent à indiquer que toutes les grandeurs liées au rotor sont référées au stator. Enfin, ω et ω_r identifient respectivement la vitesse angulaire du référentiel et la vitesse angulaire électrique du rotor.

Les équations d'état de la partie électrique de notre modèle s'obtiennent en substituant les flux de (2-16) dans les équations des tensions (2-15), puis en isolant les dérivées de tous les courants. On obtient alors deux systèmes de deux équations à deux inconnues qu'il faut résoudre. Le résultat est le suivant.

$$\frac{d}{dt}\begin{bmatrix} i_{qs} \\ i_{ds} \\ i'_{qr} \\ i'_{dr} \end{bmatrix} = \frac{1}{D}\left[\begin{array}{cc:cc} -r_s L'_r & C_1 & r_r L_m & -\omega_r L_m L'_r \\ -C_1 & -r_s L'_r & \omega_r L_m L'_r & r'_r L_m \\ \hdashline r_s L_m & \omega_r L_m L_s & -r'_r L_s & C_2 \\ -\omega_r L_m L_s & r_s L_m & -C_2 & -r'_r L_s \end{array}\right]\begin{bmatrix} i_{qs} \\ i_{ds} \\ i'_{qr} \\ i'_{dr} \end{bmatrix} + \frac{1}{D}\left[\begin{array}{cc:cc} L'_r & 0 & -L_m & 0 \\ 0 & L'_r & 0 & -L_m \\ \hdashline -L_m & 0 & L_s & 0 \\ 0 & -L_m & 0 & L_s \end{array}\right]\begin{bmatrix} v_{qs} \\ v_{ds} \\ v'_{qr} \\ v'_{dr} \end{bmatrix}$$

$$\begin{aligned} D &= L'_r L_s - L_m^2 \\ C_1 &= L_m^2(\omega - \omega_r) - \omega L'_r L_s \\ C_2 &= \omega L_m^2 - L'_r L_s(\omega - \omega_r) \end{aligned} \tag{2-17}$$

Afin de modéliser le comportement de la machine complète, nous devons ajouter à (2-17) les équations relatives à la partie mécanique de la machine. D'abord, le couple électromagnétique T_e développé par la machine s'obtient à l'aide de la relation (2-18) [25], où p est le nombre de paires de pôles de la machine.

$$T_e = \frac{3}{2} p L_m (i_{qs} i'_{dr} - i_{ds} i'_{qr}) \tag{2-18}$$

La partie mécanique de la machine (le rotor) est modélisée par une masse cylindrique de densité uniforme tournant sur son axe. On considère que le rotor et la charge qu'il entraîne sont caractérisés par un moment d'inertie combiné J et un coefficient de frottement visqueux F. Les équations dynamiques associées au rotor sont les suivantes:

$$\begin{aligned} \frac{d}{dt}\omega_m &= \frac{1}{J}(T_e - T_m - F\omega_m) \\ \frac{d}{dt}\theta_m &= \omega_m \end{aligned} \tag{2-19}$$

Dans (2-19), ω_m et θ_m identifient respectivement la vitesse angulaire mécanique et la position angulaire mécanique du rotor, tandis que T_m est le couple mécanique de la charge.

2.2.2 Les transformations de référentiel

L'équation (2-17) ayant été développée dans le référentiel *dq* arbitraire, nous allons maintenant discuter brièvement des principales transformations de référentiel couramment utilisées dans les modèles de machines électriques et justifier le choix du référentiel retenu dans nos travaux.

2.2.2.1 Relations entre les variables de phase et le référentiel *dq* arbitraire

Il est important de faire la distinction entre deux types de transformations de référentiel, soit les transformations appliquées à des circuits stationnaires et celles appliquées à des circuits en rotation. Dans le cas de la machine asynchrone, ces deux types de transformation sont appliquées respectivement au stator et au rotor. La figure 2-7 illustre le passage du référentiel des variables de phase *abc* au référentiel *dq* arbitraire et ce, pour des circuits stationnaires. On remarque que les axes *d* (direct) et *q* (quadrature) sont orthogonaux et tournent à une vitesse arbitraire ω, d'où le nom de référentiel *dq* arbitraire. L'axe magnétique de la phase *a* a arbitrairement été placé à l'origine du cercle trigonométrique, qui sert de référence. L'axe *q* forme quant à lui un angle θ avec l'origine. Le symbole *f* identifie la variable électrique à laquelle on applique la transformation et

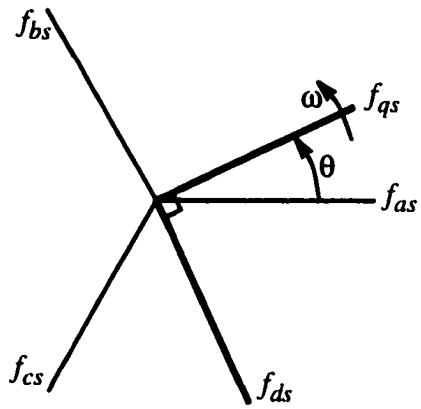


Figure 2-7 Représentation graphique de la transformation de référentiel pour des circuits stationnaires.

peut être soit un courant, une tension, un flux ou une charge électrique. Dans notre cas, la seule variable que nous transformons du référentiel *abc* vers le référentiel *dq* arbitraire est la tension statorique de la machine. Comme nous considérons une machine triphasée dont les enroulements sont raccordés en étoile et que le neutre n'est pas accessible (connexion à trois fils), il y a seulement deux tensions indépendantes qui peuvent être spécifiées. Nous avons arbitrairement choisi d'utiliser v_{ab} et v_{bc} comme tensions d'entrée et ce, au rotor comme au stator. Nous définissons donc une première matrice de transformation de référentiel en (2-20). Le symbole K sera utilisé pour identifier toutes les matrices de transformation et le suffixe s indique qu'il s'agit d'une transformation appliquée à des circuits stationnaires.

$$\begin{bmatrix} v_{qs} \\ v_{ds} \end{bmatrix} = K_s \begin{bmatrix} v_{abs} \\ v_{bcs} \end{bmatrix}$$

$$K_s = \frac{1}{3}\begin{bmatrix} 2\cos\theta & \cos\theta + \sqrt{3}\sin\theta \\ 2\sin\theta & \sin\theta - \sqrt{3}\cos\theta \end{bmatrix} \qquad (2\text{-}20)$$

Il est également important de définir la transformation complémentaire à (2-20), qui nous permettra le passage du référentiel *dq* arbitraire au référentiel *abc*. Dans notre cas, la seule variable à subir cette transformation au stator est le courant de phase. Encore une fois, à cause de la connexion en étoile à trois fils, seulement deux des trois courants de phase sont indépendants. Nous avons arbitrairement choisi de travailler avec i_{as} et i_{bs} et la matrice de transformation associée à ces courants est donnée en (2-21). Nous utiliserons le symbole K^{-1} pour identifier toutes les matrices de transformation permettant le passage du référentiel *dq* arbitraire au référentiel *abc*.

$$\begin{bmatrix} i_{as} \\ i_{bs} \end{bmatrix} = (K_s)^{-1} \begin{bmatrix} i_{qs} \\ i_{ds} \end{bmatrix}$$
$$(K_s)^{-1} = \frac{1}{2} \begin{bmatrix} 2\cos\theta & 2\sin\theta \\ -\cos\theta+\sqrt{3}\sin\theta & -\sqrt{3}\cos\theta-\sin\theta \end{bmatrix} \tag{2-21}$$

Il reste à définir les relations entres les variables de phase *abc* et les variables *dq*, mais cette fois du côté du rotor, ou de façon plus générale pour des circuits en rotation. La figure 2-8 permet de définir les principales relations. Le référentiel *dq* forme toujours

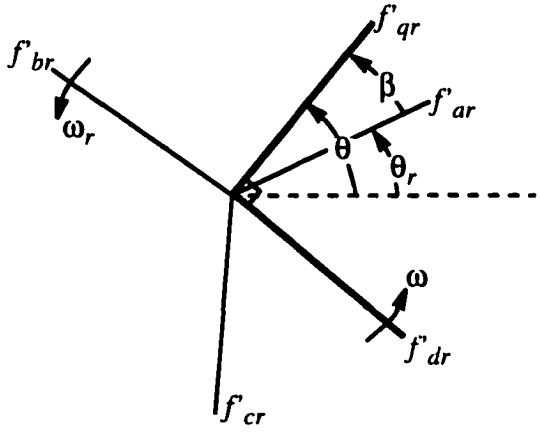


Figure 2-8 Représentation graphique de la transformation de référentiel pour des circuits en rotation.

un angle θ avec la référence (origine du cercle trigonométrique) et tourne toujours à une vitesse ω. Le référentiel *abc* quant à lui tourne aussi à une vitesse ω_r. De plus, il forme un angle θ_r avec l'origine. Enfin, la différence d'angle entre les deux référentiels est identifiée par le symbole β.

Les variables que nous devons transformer au rotor sont les mêmes qu'au stator, c'est-à-dire que nous devons transformer les tensions rotoriques du référentiel *abc* vers le référentiel *dq* arbitraire et les courants rotoriques en sens inverse, soit du référentiel *dq* arbitraire au référentiel *abc*. Les matrices de transformations requises sont identifiées en (2-22) et (2-23). Le suffixe *r* indique que les transformations sont appliqués à des circuits en rotation.

$$\begin{bmatrix} v'_{qr} \\ v'_{dr} \end{bmatrix} = K_r \begin{bmatrix} v'_{abr} \\ v'_{bcr} \end{bmatrix}$$

$$K_r = \frac{1}{3}\begin{bmatrix} 2\cos\beta & \cos\beta + \sqrt{3}\sin\beta \\ 2\sin\beta & \sin\beta - \sqrt{3}\cos\beta \end{bmatrix} \qquad (2\text{-}22)$$

$$\begin{bmatrix} i'_{ar} \\ i'_{br} \end{bmatrix} = (K_r)^{-1} \begin{bmatrix} i'_{qr} \\ i'_{dr} \end{bmatrix}$$

$$(K_r)^{-1} = \frac{1}{2}\begin{bmatrix} 2\cos\beta & 2\sin\beta \\ -\cos\beta + \sqrt{3}\sin\beta & -\sqrt{3}\cos\beta - \sin\beta \end{bmatrix} \qquad (2\text{-}23)$$

2.2.2.2 Transformations de référentiel courantes

Il y a trois transformations de référentiel qui sont couramment employées dans la simulation des machines électriques. Dans chaque cas, il s'agit d'assigner une vitesse particulière au référentiel pour obtenir une transformation donnée. Ces transformations se font dans les référentiels suivants:

a. référentiel *dq* fixé au stator ou stationnaire;

b. référentiel *dq* fixé au rotor;

c. référentiel *dq* synchrone.

Le choix optimal du référentiel dépend essentiellement de l'application. Dans chaque cas, il s'agit de rendre disponible les variables que l'on désire observer tout en minimisant la quantité de calculs [24]. Nous allons d'abord présenter les trois transformations, après quoi nous discuterons de l'utilisation de chacune.

Le transformation dans le référentiel *dq* fixé au stator ou stationnaire porte aussi le nom de transformation de Clarke ou transformation $\alpha\beta$. Ce référentiel étant fixe, la vitesse est nulle. Comme la vitesse est nulle, l'angle du référentiel demeurera constant et sa valeur est arbitraire. Nous avons choisi de travailler avec un angle nul. Pour obtenir les matrices de transformation dans le référentiel *dq* stationnaire, il suffit donc de substituer dans les matrices de transformations dans le référentiel *dq* arbitraire (2-20) à (2-23) les valeurs (2-24). Dans ces conditions, on obtient les matrices de transformation (2-25) à (2-28).

$$\begin{aligned} \omega &= 0 \\ \theta &= 0 \\ \beta &= -\theta_r \end{aligned} \tag{2-24}$$

$$K_s = \frac{1}{3}\begin{bmatrix} 2 & 1 \\ 0 & -\sqrt{3} \end{bmatrix} \tag{2-25}$$

$$(K_s)^{-1} = \frac{1}{2}\begin{bmatrix} 2 & 0 \\ -1 & -\sqrt{3} \end{bmatrix} \tag{2-26}$$

$$K_r = \frac{1}{3}\begin{bmatrix} 2\cos\theta_r & \cos\theta_r - \sqrt{3}\sin\theta_r \\ -2\sin\theta_r & -\sin\theta_r - \sqrt{3}\cos\theta_r \end{bmatrix} \tag{2-27}$$

$$(K_r)^{-1} = \frac{1}{2}\begin{bmatrix} 2\cos\theta_r & -2\sin\theta_r \\ -\cos\theta_r - \sqrt{3}\sin\theta_r & -\sqrt{3}\cos\theta_r + \sin\theta_r \end{bmatrix} \tag{2-28}$$

Les matrices de transformation dans le référentiel fixé au rotor s'obtiennent de la même manière. La vitesse du référentiel ainsi que sa position angulaire deviennent celles du rotor. L'angle β est alors nul. En substituant les valeurs (2-29) dans (2-20) à (2-23), on obtient les matrices (2-30) à (2-33).

$$\begin{aligned} \omega &= \omega_r \\ \theta &= \theta_r \\ \beta &= 0 \end{aligned} \tag{2-29}$$

$$K_s = \frac{1}{3}\begin{bmatrix} 2\cos\theta_r & \cos\theta_r + \sqrt{3}\sin\theta_r \\ 2\sin\theta_r & \sin\theta_r - \sqrt{3}\cos\theta_r \end{bmatrix} \tag{2-30}$$

$$(K_s)^{-1} = \frac{1}{2}\begin{bmatrix} 2\cos\theta_r & 2\sin\theta_r \\ -\cos\theta_r + \sqrt{3}\sin\theta_r & -\sqrt{3}\cos\theta_r - \sin\theta_r \end{bmatrix} \tag{2-31}$$

$$K_r = \frac{1}{3}\begin{bmatrix} 2 & 1 \\ 0 & -\sqrt{3} \end{bmatrix} \tag{2-32}$$

$$(K_r)^{-1} = \frac{1}{2}\begin{bmatrix} 2 & 0 \\ -1 & -\sqrt{3} \end{bmatrix} \tag{2-33}$$

Enfin, il nous reste à définir les matrices de transformations associées au référentiel *dq* synchrone. Ce référentiel tourne à la vitesse du champ tournant du stator, que l'on identifie par le symbole ω_e, tandis que sa position angulaire est identifiée par θ_e. L'angle β est calculé à chaque pas de simulation et consiste cette fois en la différence d'angle

identifiée à la figure 2-8. Les matrices de transformation pour ce référentiel s'obtiennent donc en substituant les valeurs (2-34) dans (2-20) à (2-23). Le résultat est (2-35) à (2-38).

$$\begin{aligned} \omega &= \omega_e \\ \theta &= \theta_e \end{aligned} \tag{2-34}$$

$$K_s = \frac{1}{3}\begin{bmatrix} 2\cos\theta_e & \cos\theta_e + \sqrt{3}\sin\theta_e \\ 2\sin\theta_e & \sin\theta_e - \sqrt{3}\cos\theta_e \end{bmatrix} \tag{2-35}$$

$$(K_s)^{-1} = \frac{1}{2}\begin{bmatrix} 2\cos\theta_e & 2\sin\theta_e \\ -\cos\theta_e + \sqrt{3}\sin\theta_e & -\sqrt{3}\cos\theta_e - \sin\theta_e \end{bmatrix} \tag{2-36}$$

$$K_r = \frac{1}{3}\begin{bmatrix} 2\cos\beta & \cos\beta + \sqrt{3}\sin\beta \\ 2\sin\beta & \sin\beta - \sqrt{3}\cos\beta \end{bmatrix} \tag{2-37}$$

$$(K_r)^{-1} = \frac{1}{2}\begin{bmatrix} 2\cos\beta & 2\sin\beta \\ -\cos\beta + \sqrt{3}\sin\beta & -\sqrt{3}\cos\beta - \sin\beta \end{bmatrix} \tag{2-38}$$

Nous avons déjà mentionné que le choix de transformation de référentiel est essentiellement dicté par les variables de phase que nous voulons observer ainsi que l'application. Quelques exemples sont fournis dans [24] et nous résumons ces exemples dans ce qui suit. Le premier cas cité est lorsque l'on utilise une machine asynchrone dans un réseau électrique équilibré. Si les tensions statoriques appliquées à une machine d'induction à cage sont équilibrées, les tensions v_{qs} et v_{ds} sont des constantes lorsque l'on utilise le référentiel synchrone tandis que pour une machine à cage, les tensions rotoriques v'_{qr} et v'_{dr} sont nulles. Si les variables de phases au stator et au rotor ne nous intéressent pas, la machine peut être simulée dans le référentiel synchrone sans qu'aucune transformation ne soit nécessaire. Si l'on désire simuler cette même machine dans des conditions équilibrées mais que cette fois, on désire observer les courants de phase statoriques, la transformation dans le référentiel stationnaire est celle qui requiert le moins de calculs. Si par

contre on s'intéresse aux courants de phase rotoriques, la transformation dans le référentiel fixé au rotor sera la plus avantageuse.

Dans le cas où il y a un déséquilibre, que ce soit au stator ou au rotor, aucun référentiel ne permet d'obtenir des valeurs constantes de tension comme c'est le cas dans l'exemple sus-mentionné. Si par exemple on simule une machine à rotor bobiné et que des résistances externes sont raccordées au rotor, il devient nécessaire d'obtenir les courants de phase rotoriques afin de modéliser l'interaction entre le modèle de machine et ces résistances, si ces dernières sont simulées dans le référentiel *abc*. Si nous nous intéressons aussi aux courants de phase statoriques, le référentiel fixé au rotor ou le référentiel stationnaire permettront une simulation avec un minimum de calculs. Si par contre on simule une machine à cage comportant un rotor équilibré et que les courants de phases rotoriques ne sont pas requis, le référentiel stationnaire permettra une simulation optimale puisque dans ces conditions, les matrices K_s et $(K_s)^{-1}$ sont constantes et les matrices K_r et $(K_r)^{-1}$ ne sont pas requises.

Dans nos travaux, nous nous intéressons exclusivement à la machine asynchrone à cage d'écureuil. Comme nous tentons de faire de la simulation en temps réel, il est important d'utiliser l'approche la plus performante. C'est pour cette raison que nous allons utiliser la transformation dans le référentiel stationnaire dans tout le développement qui suit. En substituant $\omega = 0$ dans les équations d'état de la partie électrique de la machine asynchrone dans le référentiel arbitraire (2-17), on obtient le système d'équations d'état suivant.

$$\frac{d}{dt}\begin{bmatrix} i_{qs} \\ i_{ds} \\ i'_{qr} \\ i'_{dr} \end{bmatrix} = \frac{1}{D}\left[\begin{array}{cc:cc} -R_s L'_r & -L_m^2\omega_r & R'_r L_m & -L_m L'_r \omega_r \\ L_m^2\omega_r & -R_s L'_r & L_m L'_r \omega_r & R'_r L_m \\ \hdashline R_s L_m & L_m L_s \omega_r & -R'_r L_s & L_s L'_r \omega_r \\ -L_m L_s \omega_r & R_s L_m & -L_s L'_r \omega_r & -R'_r L_s \end{array}\right]\begin{bmatrix} i_{qs} \\ i_{ds} \\ i'_{qr} \\ i'_{dr} \end{bmatrix}$$

$$+ \frac{1}{D}\left[\begin{array}{cc:cc} L'_r & 0 & -L_m & 0 \\ 0 & L'_r & 0 & -L_m \\ \hdashline -L_m & 0 & L_s & 0 \\ 0 & -L_m & 0 & L_s \end{array}\right]\begin{bmatrix} v_{qs} \\ v_{ds} \\ v'_{qr} \\ v'_{dr} \end{bmatrix} \qquad (2\text{-}39)$$

$$D = L'_r L_s - L_m^2$$

2.3 Entraînement complet

Nous présentons maintenant une technique permettant de modéliser toutes les composantes d'un entraînement électrique dans un seul système d'équations d'état, permettant ainsi une simulation simultanée sans délai de la dynamique complète de l'entraînement. La technique en question est tirée d'une thèse de doctorat publiée en 1979 [26]. Outre le fait qu'elle permet la solution simultanée du système entier, cette méthode a le grand avantage d'utiliser un modèle de machine dans le référentiel *dq*, ce qui en simplifie grandement les équations. Nous utiliserons le système montré à la figure 2-9 en guise d'exemple. Il s'agit d'un onduleur à MOSFET alimenté par une source de tension idéale à courant continu et alimentant une machine d'induction à cage d'écureuil.

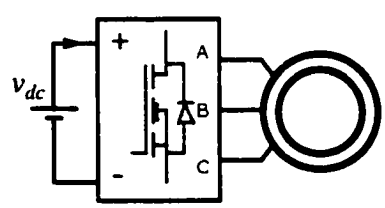


Figure 2-9 Schéma de l'exemple.

La première étape consiste à bien définir les différentes parties de notre simulation. Le système est d'abord coupé en deux, soit le convertisseur et la source de tension d'un côté et la machine de l'autre. Pour l'instant, la machine est considérée comme deux sources de courant dites fictives [26] qui représentent les courants statoriques d'une machine raccordée en Y à trois fils, tel que montré à la figure 2-10.

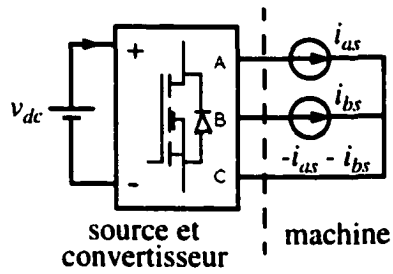


Figure 2-10 Séparation du schéma.

Le convertisseur voit donc deux ensembles d'entrées:

a. le vecteur $e = [v_{dc}]$, qui dans ce cas-ci ne contient qu'un élément;
b. le vecteur $i = [i_{as}\ i_{bs}]^T$, qui contient les sources de courant fictives.

Ensuite nous devons subdiviser le système à nouveau. Il s'agit cette fois de séparer la machine en deux parties, soit une partie commune avec le convertisseur (le stator) et une partie propre à la machine (le rotor). Notre système comporte maintenant trois parties distinctes, tel qu'illustré à la figure 2-11. Dans cette figure, l'indice *c* est utilisé pour identifier une variable propre au convertisseur et ce, dans le référentiel *abc*. L'indice *mm* identifie une variable propre à la machine dans le référentiel *dq*. Les tensions et courants rotoriques v_{mm} et i_{mm} sont ces variables. Enfin, l'indice *cm* identifie une variable commune aux deux parties, mais dans le référentiel *dq*. Les courants et tensions statoriques v_{cm} et i_{cm} formeront ce dernier groupe.

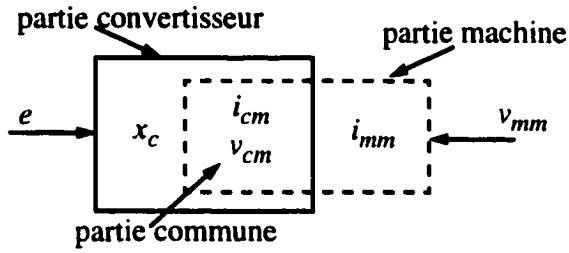


Figure 2-11 Séparation du système en trois parties distinctes.

Nous avons expliqué au chapitre précédent comment obtenir la représentation d'état du convertisseur à l'aide de la théorie des graphes. Cependant, nous devons scinder le vecteur des entrées u (et par conséquent la matrice B) en deux afin d'accommoder les étapes suivantes. L'équation d'état du convertisseur devient alors

$$\dot{x}_c = A_c x_c + B_{c1} e + B_{c2} i \tag{2-40}$$

On peut maintenant exprimer (2-17) de façon plus compacte:

$$\dot{x}_m = A_m x_m + B_m u_m \tag{2-41}$$

Les vecteurs des variables d'état et des entrées de la machine sont définis comme suit (référentiel dq):

$$\begin{aligned} x_m &= \left[i_{qs} \; i_{ds} \vdots i'_{qr} \; i'_{dr} \right]^T = \left[i_{cm} \vdots i_{mm} \right]^T \\ u_m &= \left[v_{qs} \; v_{ds} \vdots v'_{qr} \; v'_{dr} \right]^T = \left[v_{cm} \vdots v_{mm} \right]^T \end{aligned} \tag{2-42}$$

Comme nous sommes en présence d'une machine à cage, les tensions rotoriques v'_{qr} et v'_{dr} (figure 2-6, page 69) sont nulles, donc v_{mm} est zéro et ne sera plus considéré

dans le reste du développement. L'équation d'état de la machine s'exprime maintenant par:

$$\frac{d}{dt}\begin{bmatrix} i_{cm} \\ i_{mm} \end{bmatrix} = \begin{bmatrix} A_{m11} & A_{m12} \\ A_{m21} & A_{m22} \end{bmatrix}\begin{bmatrix} i_{cm} \\ i_{mm} \end{bmatrix} + \begin{bmatrix} B_{m11} \\ B_{m21} \end{bmatrix} v_{cm} \tag{2-43}$$

Il nous reste maintenant à définir les équations qui nous permettront d'interfacer la partie convertisseur dans le référentiel *abc* à la partie commune de la machine dans le référentiel *dq*. D'abord, comme les courants de la partie commune de la machine i_{cm} sont exprimés dans le référentiel *dq*, nous devons définir la transformation suivante afin de pouvoir calculer les injections de courant i_{as} et i_{bs} dans le convertisseur:

$$\begin{aligned} i = \begin{bmatrix} i_{as} \\ i_{bs} \end{bmatrix} &= \begin{bmatrix} 1 & 0 \\ -1/2 & -\sqrt{3}/2 \end{bmatrix}\begin{bmatrix} i_{qs} \\ i_{ds} \end{bmatrix} \\ &= M i_{cm} \end{aligned} \tag{2-44}$$

La matrice M est en fait la matrice $(K_s)^{-1}$ dans (2-26). Ensuite, les entrées de la partie commune (vues du convertisseur) sont les tensions v_{ab} et v_{bc} aux bornes des sources de courant fictives de la figure 2-10. Ces tensions sont obtenues dans le référentiel *abc* avec la relation suivante:

$$\begin{aligned} v_i = \begin{bmatrix} v_{abs} \\ v_{bcs} \end{bmatrix} &= P x_c + Q_1 e + Q_2 i \\ &= P x_c + Q_1 e + Q_2 M i_{cm} \end{aligned} \tag{2-45}$$

Dans cette équation, i est le vecteur des sources de courant fictives et e est le vecteur des sources de tension, tous deux définis plus haut. Les matrices P, Q_1 et Q_2 sont des

sous-matrices des matrices C et D ($y = Cx + Du$) du convertisseur. Les deux tensions v_{ab} et v_{bc} sont ensuite transformées dans le référentiel stationnaire avec la relation suivante:

$$\begin{aligned} v_{cm} &= \begin{bmatrix} v_{qs} \\ v_{ds} \end{bmatrix} = \begin{bmatrix} 2/3 & 1/3 \\ 0 & -\sqrt{3}/3 \end{bmatrix} \begin{bmatrix} v_{abs} \\ v_{bcs} \end{bmatrix} \\ &= N v_i \\ &= N(P x_c + Q_1 e + Q_2 M i_{cm}) \end{aligned} \tag{2-46}$$

La matrice N est en fait la matrice K_s dans (2-25). La combinaison des diverses expressions nous permet de définir le comportement du système complet, avec comme variables d'état les tensions aux bornes de condensateurs et les courants d'inductances du convertisseur (x_c, référentiel *abc*) et les courants statoriques et rotoriques de la machine i_{cm} et i_{mm} (référentiel *dq*):

$$\frac{d}{dt}\begin{bmatrix} x_c \\ i_{cm} \\ i_{mm} \end{bmatrix} = \begin{bmatrix} A_c & B_{c2}M & 0 \\ B_{m11}NP & A_{m11}+B_{m11}NQ_2M & A_{m12} \\ B_{m21}NP & A_{m21}+B_{m21}NQ_2M & A_{m22} \end{bmatrix} \begin{bmatrix} x_c \\ i_{cm} \\ i_{mm} \end{bmatrix} + \begin{bmatrix} B_{c1} \\ B_{m11}NQ_1 \\ B_{m21}NQ_1 \end{bmatrix} e \tag{2-47}$$

2.4 Validation de la méthode de simulation en temps différé

Bien que notre but ultime soit de simuler des entraînements électriques en temps réel, nous devons d'abord étudier la validité de la méthodologie proposée en temps différé.

La première étape de notre validation consiste à réaliser une simulation étalon du montage à étudier qui servira de référence. Nous avons choisi d'utiliser le Power System

Blockset pour faire cette simulation de référence. Ce choix est principalement motivé par le fait qu'en tant que co-auteur du logiciel, nous le connaissons bien.

Le système choisi est relativement simple et consiste en une machine asynchrone triphasée alimentée par un onduleur à MOSFET, lui même alimenté par une batterie. La commande directe du flux et du couple (Direct Flux and Torque Control, ou DFTC) [27,28] est utilisée pour contrôler l'onduleur. La machine entraîne une charge de type ventilateur. La figure 2-12 illustre le schéma du montage, tandis que la figure 2-13 illustre le schéma de la simulation réalisée avec le Power System Blockset et Simulink. Les principaux paramètres associés à cet entraînement sont donnés à l'annexe A. Quelques précisions s'imposent ici relativement aux deux figures. D'abord, ce système est la version simplifiée d'un système plus complexe. Le système d'origine comprend, au lieu d'une batterie, un redresseur triphasé. Comme ce système sera utilisé au dernier chapitre, nous conservons ici les composantes qui en feront partie, notamment le bus à courant continu ainsi que la résistance de décharge qui lui est raccordée. La seule composante de la figure 2-12 que l'on ne retrouve pas dans le système complet est la résistance en parallèle avec chacun des interrupteurs. Cette résistance est en fait requise à cause du fait que les interrupteurs sont vus par la représentation d'état comme des sources de courant. Or, sans la résistance, nous nous retrouverions avec des sources de courant en série, ce qui viole une des lois fondamentales d'interconnection des sources. La résistance placée en parallèle avec chaque interrupteur permet de contourner cette restriction, laquelle est uni-

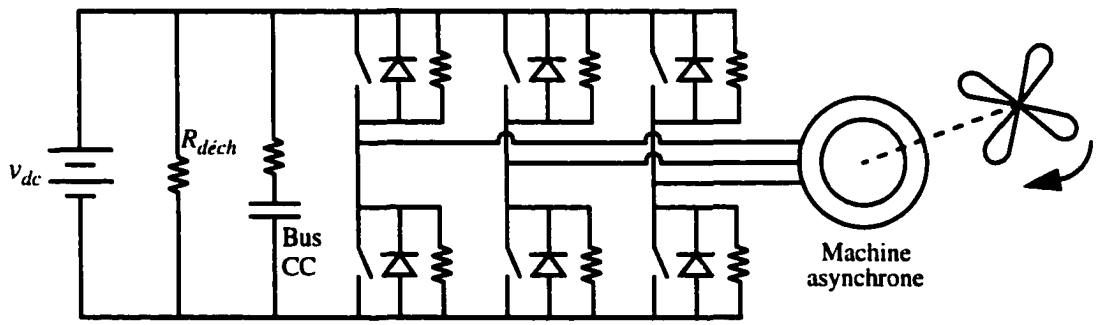


Figure 2-12 Schéma de l'entraînement étudié.

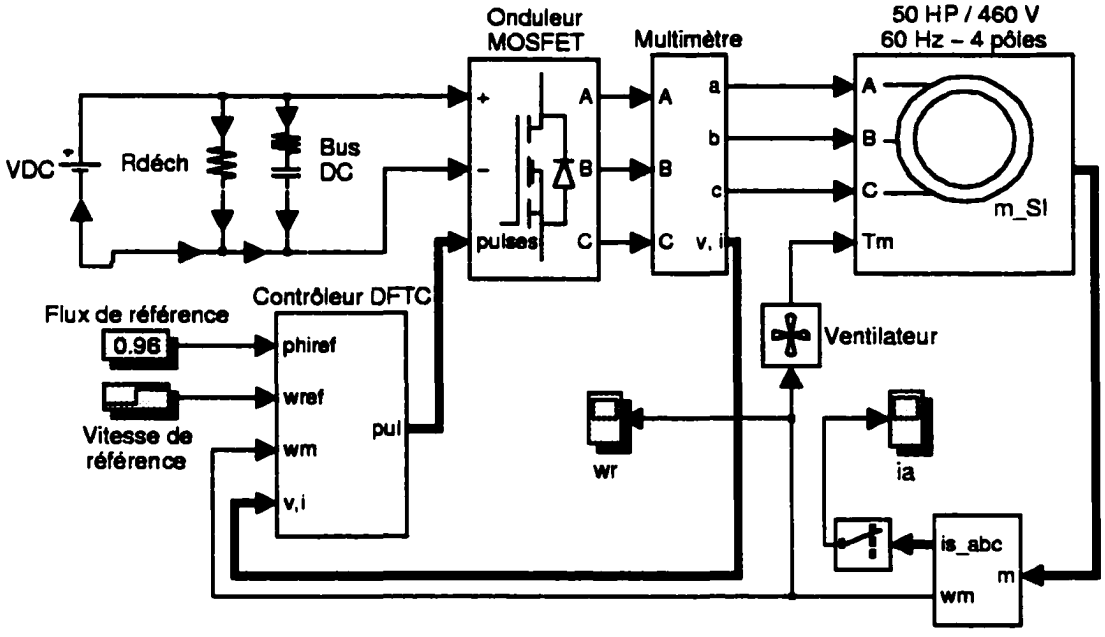


Figure 2-13 Schéma Simulink/PSB de la simulation de référence.

quement due à la modélisation utilisée. La résistance que nous utilisons a une valeur de 1 MΩ et n'influence pas le comportement dynamique du système étudié.

Nous avons programmé les équations de la méthode proposée à la section 2.3 dans une s-function. La s-function est un bloc Simulink qui peut être inséré dans un schéma-bloc et auquel est associé un programme. La raison d'être de la s-function est de permettre aux usagers de réaliser sous forme de programme des algorithmes difficilement réalisables avec les blocs standards de Simulink. Elle permet donc d'interconnecter des algorithmes complexes avec des schémas de commandes réalisés dans Simulink.

Cette s-function peut être programmée dans divers langages, notamment le langage Matlab, le C, et le Fortran. Dans un premier temps, nous avons réalisé notre s-function en langage Matlab, puisque celui-ci permet de réaliser simplement les calculs matriciels, lesquels sont nombreux dans nos équations. La programmation d'une s-function en langage Matlab est la plus simple, mais c'est aussi la moins performante en termes de vitesse de

simulation. Or, elle permet de développer rapidement des prototypes de simulation. La figure 2-14 illustre un schéma Simulink utilisant notre s-function en langage Matlab.

On voit que la s-function reçoit en entrée la tension de la batterie, les impulsions produites par la commande DFTC ainsi que le couple mécanique de la charge. La s-function transmet à sa sortie un vecteur de signaux comprenant entre autres les tensions statoriques et les courants de phase de la machine ainsi que la vitesse de rotation de la machine, signaux requis par la commande. On remarque aussi la présence de deux délais unitaires introduits entre l'entrée de la s-function et les sorties de la commande et du ventilateur. Ces délais sont requis pour briser des boucles algébriques, dont nous discuterons dans le prochain chapitre. Mentionnons simplement pour l'instant que deux boucles algébriques sont formées dans cette simulation parce que la sortie $y(n)$ de la s-function dépend de l'entrée $u(n)$ de la s-function et que la même situation se présente d'une part dans la commande DFTC, et d'autre part dans le modèle du ventilateur. Comme ces deux derniers modèles sont chacun dans une boucle de contre-réaction avec la s-function, les sorties de chacun des trois systèmes au moment n dépendent d'elles mêmes. La simulation du modèle mathématique du système dans son ensemble requiert, dans ces condi-

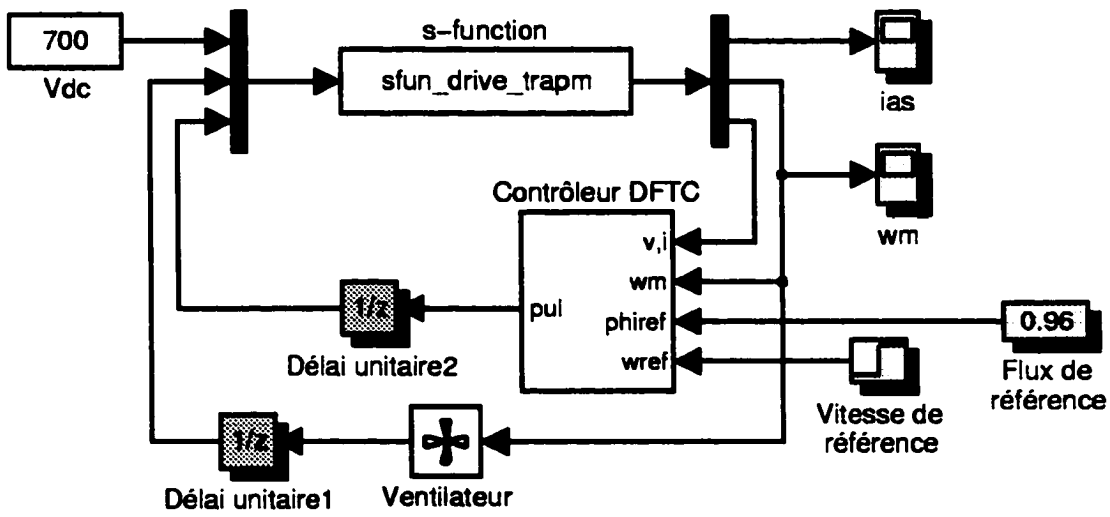


Figure 2-14 Schéma Simulink utilisant la s-function en langage Matlab.

tions, une solution itérative à chaque pas de calcul. Comme nous visons à faire de la simulation en temps réel, un processus itératif n'est pas toléré et des délais unitaires sont ajoutés pour briser les boucles algébriques.

La s-function requiert également un certain nombre de paramètres d'entrée, entre autres les valeurs des paramètres du convertisseur et de la machine, les valeurs initiales des matrices d'état, etc. Ces valeurs sont saisies à partir du schéma de la simulation de référence de la figure 2-13 à l'aide d'une version modifiée des routines d'analyse du Power System Blockset.

Une fois les paramètres d'entrée obtenus, le schéma de la figure 2-14 peut être simulé dans Simulink. Nous décrirons en détail les différentes tâches effectuées dans la s-function au chapitre suivant, après avoir discuté de la question de l'intégration numérique des équations d'état de notre système.

Les performances, en termes de vitesse d'exécution, de cette première s-function en langage Matlab se sont avérées décevantes. Nous avons donc converti la s-function en langage C afin d'en augmenter la vitesse d'exécution. Bien qu'il existe aujourd'hui un toolbox Matlab, le Matlab Compiler, qui permette de faire cette conversion automatiquement, ce produit n'était pas disponible au moment où nous avons converti notre s-function. Nous l'avons donc converti nous-même du langage Matlab au langage C. Cette conversion a nécessité une familiarisation avec l'interface de programmation (API, ou Application Program Interface) des s-function en langage C [30,31]. La tâche de conversion qui a exigé le plus d'effort consistait à écrire en langage C les principales routines d'algèbre matriciel requises (inversion, multiplication et addition de matrices).

Avant d'effectuer notre comparaison, nous devons préciser certains faits relatifs au montage étudié. Bien que notre but ne soit pas d'étudier le fonctionnement de la commande DFTC, nous devons au moins décrire les particularités de cette commande qui

nous concernent. D'abord, contrairement à une commande à modulation de largeur d'impulsions (MLI), la fréquence de commutation maximale n'est pas définie dans la commande DFTC, si on ne prend pas les moyens de la limiter. En d'autres termes, les interrupteurs peuvent changer d'état à chaque pas de calcul. Si notre simulation est réalisée à pas fixe, la fréquence maximale de commutation devient l'inverse du pas de calcul employé. Nous devons donc ajouter dans la commande un mécanisme qui limite la fréquence de commutation des interrupteurs à une valeur raisonnable, en fonction de la puissance de l'entraînement simulé.

Notre objectif étant de simuler des entraînements de moyenne et grande puissances afin de faciliter le prototypage des commandes, nous fixons la fréquence de commutation maximale des interrupteurs à quatre kHz, valeur raisonnable pour des entraînements de quelques dizaines de HP et plus. Cette fréquence de commutation maximale impose des contraintes sur le pas de calcul qui peut être utilisé pour la simulation d'un tel système. Selon le théorème d'échantillonnage, on peut utiliser un pas de calcul maximum T_{max} égal à

$$T_{max} = \frac{1}{2 \times f_{max}} , \tag{2-48}$$

où f_{max} est la fréquence du signal le plus rapide dans la simulation. Dans notre cas, cette fréquence correspond à la fréquence maximale de commutation, que nous limitons à quatre kHz. Ceci implique donc que le pas de calcul maximum que nous pouvons utiliser dans ces conditions est T_{max} = 125 µs. Dans la pratique, on utilise couramment des pas de calcul plus petits que le pas obtenu avec le théorème d'échantillonnage. Nous considérerons quand même 125 µs comme le pas maximum, afin d'évaluer et comparer la précision des résultats obtenus avec les deux méthodes de simulation.

Nous devons également fixer un pas de calcul minimum pour faire nos comparaisons, en gardant en tête que nous visons à faire de la simulation en temps réel. Les simulateurs en temps réel sont en fait des ordinateurs multiprocesseurs équipés d'entrées/sorties. Or, dans ces simulateurs, un temps minimum est requis à chaque pas de calcul afin d'effectuer l'acquisition et la conversion des entrées/sorties d'une part, et les communication interprocesseur d'autre part. Sur le simulateur Hypersim de l'É.T.S., ces deux tâches prennent environ 25 µs à chaque pas de calcul. Il est prévu que sur la prochaine génération de simulateurs, ces tâches totaliseront environ 15 µs. Ce temps s'ajoute au temps dédié au calcul et le total doit être inférieur au pas de calcul utilisé pour la simulation. Nous utilisons donc 25 µs comme limite inférieure de pas de calcul pour nos comparaisons.

Un dernier commentaire s'impose concernant le pas de calcul du montage à l'étude. Notre commande DFTC nous force à intégrer un mécanisme de limitation de la fréquence maximale de commutation. La façon dont nous nous y prenons pour implémenter cette fonctionnalité consiste d'abord à remettre un compteur à zéro à chaque changement d'état, puis à incrémenter ce compteur à chaque pas de calcul. Si la commande dicte un changement d'état mais que le compteur n'a pas atteint la valeur correspondant à la fréquence de commutation maximale prescrite, le changement n'est pas autorisé. Ce mécanisme impose une contrainte additionnelle sur le pas de calcul. Nous tenons à conserver une fréquence maximale de commutation de quatre kHz, peu importe le pas de calcul, afin de comparer entre elles des simulations faites dans des conditions similaires. Il faut donc que l'inverse de la fréquence de commutation maximale, 250 µs ici, soit un multiple exact du pas de calcul, puisque nous n'avons pas la possibilité d'incrémenter le compteur entre deux pas de calcul. En tenant compte de cette contrainte et des pas de calcul minimum et maximum établis plus haut, nous établissons le tableau 2-2 des pas de calcul qui serviront à nos comparaisons.

Tableau 2-2

Pas de calculs utilisés pour les comparaisons

Pas de calcul T_s (µs)	Valeur minimale du compteur permettant un changement d'état
125	2
$83.\overline{3}$	3
62.5	4
50.0	5
$41.\overline{6}$	6
25.0	10

Nous avons donc comparé les performances de simulation des schémas des figures 2-13 (Power System Blockset 2.0) et 2-14 (méthode proposée), en utilisant dans ce dernier cas la s-function compilée à partir de code en langage C. La simulation consiste dans chaque cas à démarrer l'entraînement, qui est initialement à l'arrêt. Les simulations ont été effectuées sur un ordinateur personnel (PC) équipé d'un processeur Pentium de première génération avec une horloge de 233 MHz et 64 Mo de mémoire vive, avec le système d'exploitation Windows 95. La première comparaison que nous faisons est de nature qualitative. Il s'agit de comparer les courbes de divers signaux des deux simulations. Nous présentons d'abord dans la figure 2-15 le courant au stator de la phase *a* du moteur asynchrone, une fois le régime permanent atteint et avec un pas de calcul de 50 µs. On peut constater que bien que légèrement décalées dans le temps, les deux courbes sont tout à fait similaires.

Nous observons ensuite la vitesse du rotor de la machine pour les deux méthodes de simulation à la figure 2-16. Cette fois-ci, nous superposons sur la même courbe la vitesse de la machine pour différents pas de calcul. On remarque d'abord que les courbes se ressemblent beaucoup pour les deux méthodes, mais qu'une différence commence à appa-

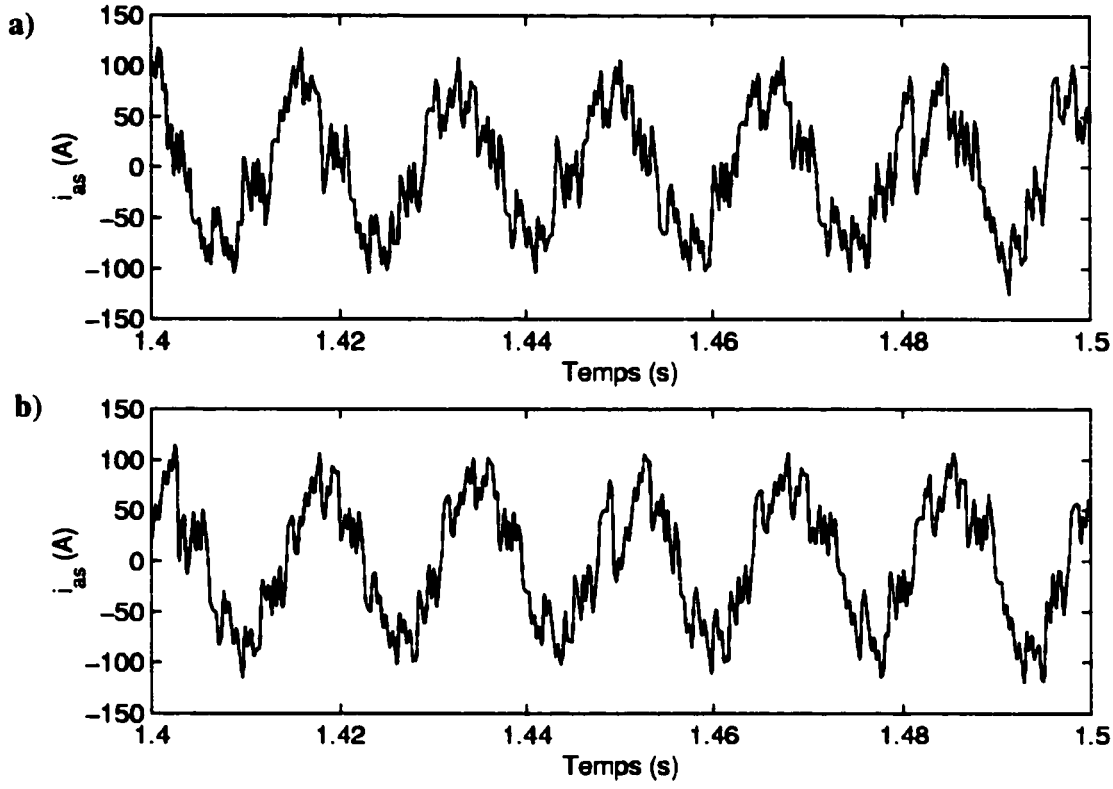


Figure 2-15 Courant statorique de la phase *a* de la machine avec les deux méthodes de simulation: **a)** PSB 2.0; **b)** méthode proposée.

raître pour les pas de plus de 62.5 µs. Cette différence est particulièrement apparente à 125 µs, mais la dynamique diffère moins de celle obtenue avec des pas de calculs moindres avec la méthode proposée. Rappelons que 125 µs est le pas de calcul maximal respectant le théorème d'échantillonnage, en considérant que la fréquence maximale de commutation des interrupteurs est de quatre kHz.

Comme notre but final est de faire de la simulation en temps réel, la temps d'exécution est un autre critère de performance important. Le tableau 2-3 résume les temps d'exécution des deux simulations pour 1.25 s de temps simulé, avec un pas de calcul de 50 µs.

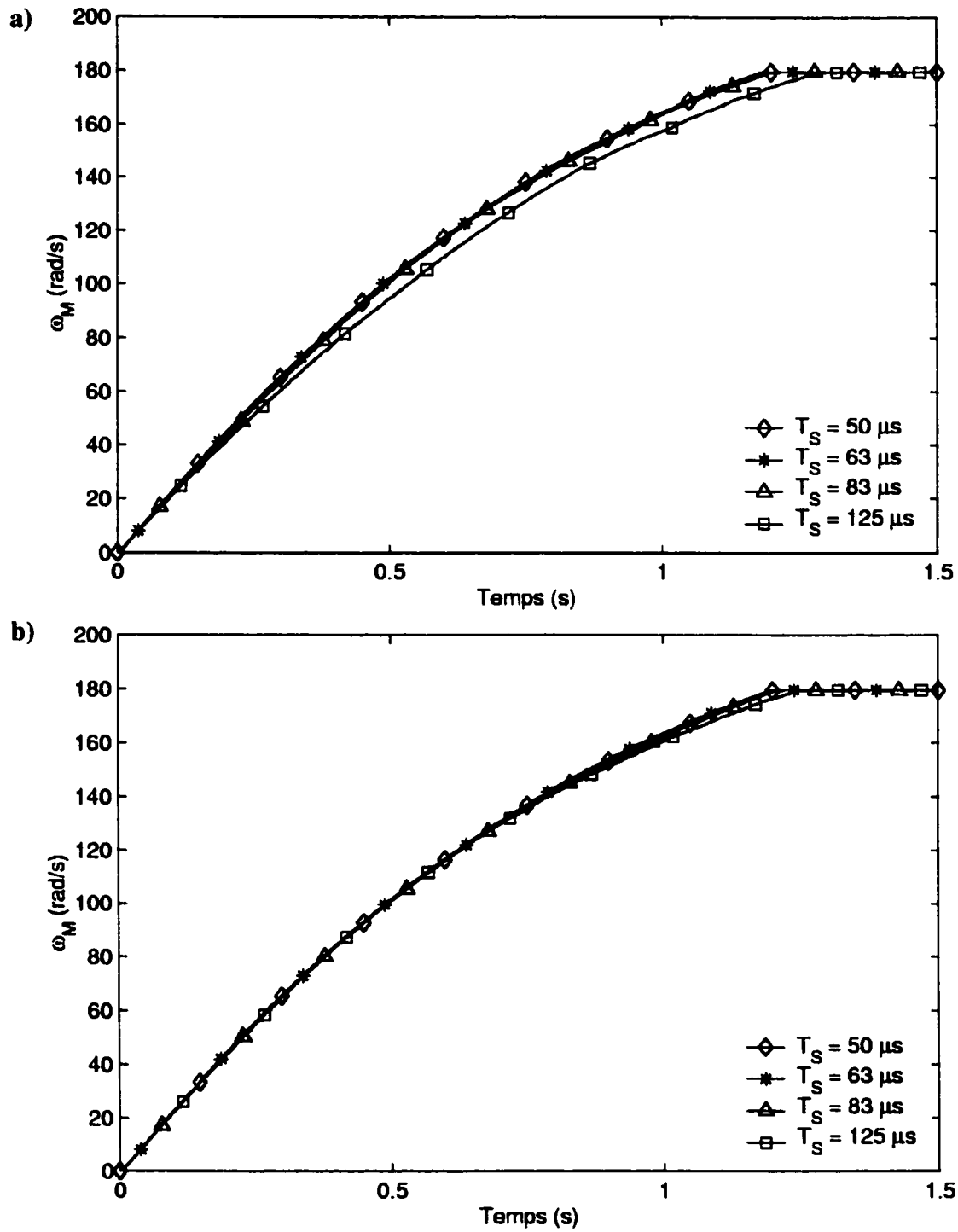


Figure 2-16 Vitesse du moteur d'induction avec les deux méthodes de simulation et pour différents pas de calcul: **a)** PSB 2.0; **b)** méthode proposée.

Tableau 2-3

Temps d'exécution des deux simulations

Méthode de simulation	Temps d'exécution (s)
Power System Blockset 2.0	36.2
Méthode proposée	12.3

Le tableau 2-3 nous permet de constater que la méthode proposée est près de trois fois plus rapide à simuler pour le montage à l'étude que la méthode utilisée dans le Power System Blockset. Ceci nous permet de conclure que les algorithmes de la méthode proposée sont non seulement précis, mais également relativement performants. La performance de ces algorithmes en temps réel fera l'objet d'une analyse détaillée au dernier chapitre.

2.5 Conclusions

Après avoir vu au premier chapitre la méthode que nous employons pour obtenir les équations d'état de la partie linéaire d'un système électrique quelconque, nous avons décrit dans le présent chapitre comment nous nous y prenons pour modéliser et simuler une classe particulière de systèmes électriques, à savoir les entraînements électriques. Nous avons d'abord discuté des principaux types de modèles d'interrupteurs et de leur inclusion dans un système électrique. Nous avons ensuite présenté une méthode innovatrice de mise à jour des matrices d'état suite à un changement d'état d'interrupteur. Rappelons que cette technique a été mise au point par Gilbert Sybille, chercheur à l'IREQ, et qu'elle a été intégrée à la version 2.0 du Power System Blockset. Nous avons par la suite discuté de modélisation de machines électriques et avons décrit un modèle de machine asynchrone dans le référentiel *dq* arbitraire. Après avoir analysé les transformations de référentiel courantes, nous avons choisi de travailler avec un modèle de machine dans le référentiel *dq* stationnaire ou fixé au stator. Les transformations associées à ce référentiel

sont également connues sous les noms de transformation de Clarke et transformation $\alpha\beta$. Notre étude des transformations de référentiel nous a également permis de développer un nouveau modèle de machine asynchrone multiréférentiel, lequel a été implanté dans la version 2.0 du PSB.

La matière présentée jusqu'à ce point est bien documentée et connue. Cependant, la technique de modélisation présentée à la section 2.3, bien que publiée dans une thèse de doctorat datant de 1979, est peu connue selon nous. Cette technique originale permet d'obtenir une représentation d'état unique pour l'ensemble convertisseur/machine d'un entraînement, et de surcroît avec comme variables d'état les tensions de condensateurs et les courants des inductances dans le référentiel *abc*, et les courants associés à la machine dans le référentiel *dq* arbitraire.

Enfin, dans la dernière section, nous avons validé l'approche proposée en comparant nos résultats avec ceux obtenus à l'aide du Power System Blockset. Après nous être assurés que les signaux d'intérêt (courants et vitesse de la machine) soient qualitativement comparables, nous avons comparé les temps de simulation pour conclure qu'avec le montage utilisé, la méthode proposée est près de trois fois plus rapide à simuler que la méthode utilisée dans le Power System Blockset version 2.0.

Le sujet n'ayant été qu'effleuré ici, bien qu'étant important, nous allons au prochain chapitre analyser plus en détail la question de l'intégration numérique qui est faite dans nos simulations.

CHAPITRE 3

INTÉGRATION NUMÉRIQUE

Le thème central de cette thèse étant la simulation de systèmes dynamiques, en l'occurrence des entraînements électriques, une partie de nos recherches a consisté à étudié les algorithmes d'intégration, lesquels sont requis pour trouver la solution de nos équations différentielles. Ce chapitre résume l'étude en question. Nous discutons d'abord des particularités des équations différentielles que nous devons résoudre et des limitations que ces particularités nous imposent au niveau du choix des algorithmes qui peuvent nous servir. Nous présentons ensuite une synthèse des algorithmes d'intégration classiques et discutons de l'implantation de l'une d'elles, la discrétisation trapézoïdale. Nous décrivons ensuite une méthode d'intégration récemment publiée et qui est destinée à la classe de problèmes que nous avons à résoudre. Enfin, les performances de cette nouvelle méthode sont comparées à celles de la discrétisation trapézoïdale à l'aide d'une étude de cas.

3.1 Équations différentielles rigides

Nous avons brièvement discuté à la section 2.1.1 du concept de rigidité. Rappelons qu'un système est dit rigide lorsque le rapport des parties réelles du pôle le plus rapide sur le plus lent est élevé. Les équations différentielles régissant le comportement dynamique des systèmes électriques sont en général rigides. Ceci est particulièrement vrai dans le cas d'un système électromécanique tel un entraînement électrique, qui est caractérisé par une dynamique électrique très rapide et une dynamique mécanique plus lente par plu-

sieurs ordres de grandeur. Une simulation du montage de la figure 2-12 a été réalisée en utilisant l'approche présentée à la section 2.3 et le coefficient de rigidité a été calculé à chaque pas de la simulation. La figure 3-1 illustre le résultat obtenu. On y remarque que le coefficient de rigidité se situe à près de 1.7 million en début de simulation et que sa valeur minimale est atteinte à la fin de la simulation et descend tout juste sous 80 000. En fait, au premier pas de calcul de la simulation, soit au temps zéro, si la vitesse initiale de la machine est zéro, la matrice A de l'équation (2-47) est singulière et ceci pose un problème lorsque nous voulons la discrétiser, car elle doit être inversée. De plus, le coefficient de rigidité est alors infini puisque la plus petite valeur propre de la matrice A est nulle. Nous devons donc démarrer le système avec une vitesse initiale non-nulle de la machine. Nous utilisons une vitesse initiale de 0.001 rad/s. Cette petite valeur suffit à rendre la matrice A non-singulière, mais le coefficient de rigidité atteint tout de même 10^{16} durant le premier pas de calcul. Au second pas, quelques interrupteurs ferment et la valeur du coefficient de rigidité s'abaisse à un niveau moindre, qui s'élève tout de même à 1.7 million.

Le principal impact de la rigidité est qu'elle restreint substantiellement le choix des algorithmes d'intégration qui peuvent être utilisés pour résoudre les équations différen-

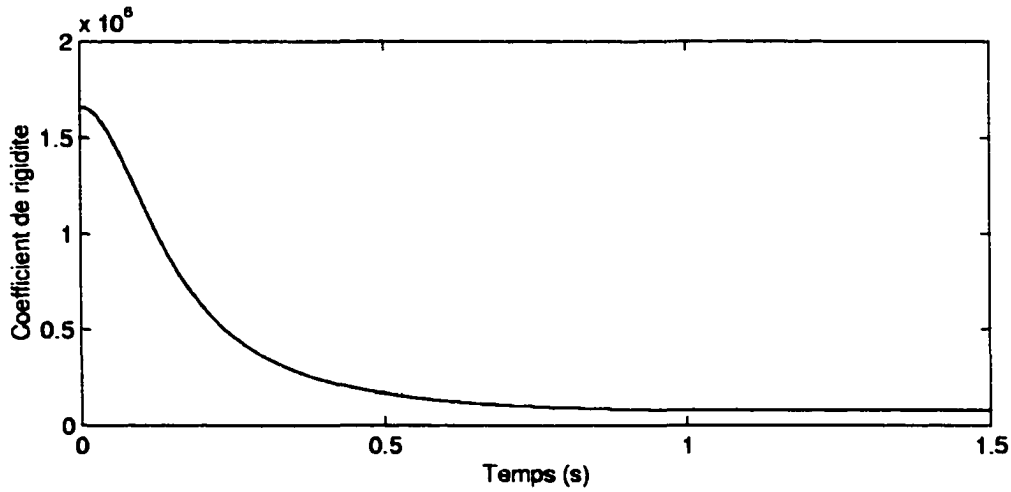


Figure 3-1 Coefficient de rigidité obtenu en simulant l'entraînement de la section 2.4.

tielles d'un système rigide. Avant d'analyser les propriétés des algorithmes d'intégration couramment employés pour la solution d'équations différentielles rigides, mentionnons qu'il existe essentiellement deux grandes familles d'algorithmes d'intégration, soit les algorithmes à pas fixe et les algorithmes à pas variable. Ces derniers étant de nature itérative, ils sont inutilisables pour des simulations en temps réel utilisant un pas de calcul de l'ordre des dizaines de microsecondes. Ainsi, nous considérons uniquement les algorithmes d'intégration à pas fixe dans nos travaux.

3.2 Caractéristiques des méthodes courantes d'intégration à pas fixe

Parmi les méthodes d'intégration à pas fixe, il existe aussi deux grandes catégories, soit les méthodes explicites et les méthodes implicites. Afin d'illustrer la différence entre ces deux types de méthodes, nous utilisons la définition (3-1) de la méthode linéaire à k pas [20].

$$\sum_{j=0}^{k} \alpha_j x_{n+j} = T \sum_{j=0}^{k} \beta_j \dot{x}_{n+j} \tag{3-1}$$

En prenant la transformée en Z de part et d'autre du signe d'égalité de (3-1) et en posant des conditions initiales nulles, on obtient la fonction de transfert discrète (3-2) pour un intégrateur.

$$H(z) = \frac{X(z)}{\dot{X}(z)} = \frac{T(\beta_k z^k + \beta_{k-1} z^{k-1} + \ldots + \beta_1 z + \beta_0)}{\alpha_k z^k + \alpha_{k-1} z^{k-1} + \ldots + \alpha_1 z + \alpha_0} \tag{3-2}$$

Une méthode d'intégration à k pas est dite implicite si β_k est non-nul (ordre du numérateur égal à l'ordre du dénominateur) et explicite si β_k est nul (ordre du numérateur inférieur à l'ordre du dénominateur). Au niveau de l'implantation des deux types de

méthodes, le vecteur d'état x_k dépend uniquement de valeurs passées (instant k-1, k-2, etc.) si la méthode est explicite mais dépend aussi de valeurs à l'instant présent (instant k) si la méthode est implicite. Dans ce dernier cas, il est nécessaire d'itérer la solution à chaque pas de calcul si le système simulé est décrit par des équations différentielles non-linéaires. Le tableau 3-1 montre les fonctions de transfert discrètes de trois méthodes d'intégration à pas fixe couramment employées dans la simulation des systèmes électriques.

Tableau 3-1

Fonctions de transfert discrètes correspondant à une approximation du processus d'intégration selon trois méthodes courantes

Nom de la méthode	Fonction de transfert discrète
Euler arrière ou Gear-1	$\frac{Tz}{z-1}$
Tustin ou trapézoïdale	$\frac{T(z+1)}{2(z-1)}$
Gear-2	$\frac{2Tz^2}{3z^2-4z+1}$

3.2.1 Évaluation de la stabilité des méthodes d'intégration

Il existe un outil d'analyse permettant d'évaluer si une méthode d'intégration à pas fixe s'applique à un système donné et avec un pas de calcul particulier et par conséquent, qui permet de comparer entre elles les méthodes d'intégration. Cet outil consiste en la région de stabilité associée à chacune des méthodes d'intégration. Chaque méthode correspond à un ensemble particulier de coefficients α et β et par conséquent, une méthode est définie par une fonction de transfert discrète qui lui est propre. Pour tracer la région de stabilité associée à la méthode d'intégration, il suffit de faire la substitution (3-3) dans la

fonction de transfert discrète et de faire varier ωT de zéro à π. Ceci nous donne la moitié de la zone de stabilité. La seconde moitié est obtenue par symétrie de la première par rapport à l'axe horizontal du plan complexe λT. Le résultat est tracé dans le plan complexe et la région de stabilité est obtenue.

$$z = e^{j\omega T} \tag{3-3}$$

Pour déterminer si un système donné peut être simulé avec une méthode d'intégration et un pas de calcul T particuliers, il suffit de connaître les valeurs propres λ de la matrice d'état A du système. Ces valeurs propres correspondent aux pôles du système en boucle fermée. Chaque valeur propre est multipliée par le pas de calcul et le résultat est reporté dans le plan complexe. Si tous les produits λT sont à l'intérieur de la région de stabilité correspondant à la méthode d'intégration, la simulation sera stable. La figure 3-2 illustre les zones de stabilité associées aux méthodes d'intégration du tableau 3-1.

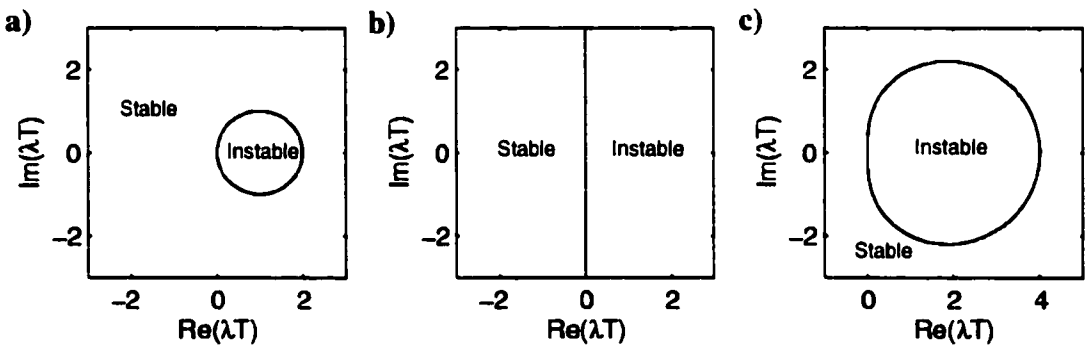


Figure 3-2 Zones de stabilité correspondant aux trois méthodes: **a)** Euler arrière; **b)** trapézoïdale; **c)** Gear-2.

Le plan complexe λT utilisé pour tracer les zones de stabilité de la figure 3-2 s'interprète de la même manière que le plan complexe utilisé pour analyser les systèmes de commande. Le demi-plan gauche est la zone de stabilité, alors que le demi-plan droite

est la zone d'instabilité. Fait à noter, les zones de stabilité des méthodes Euler arrière et Gear-2 incluent une partie du demi-plan droite. Nous discutons de cette particularité un peu plus loin. Les trois zones de stabilité de la figure 3-2 révèlent une caractéristique importante des trois méthodes d'intégration: elles sont toutes les trois *A*-stables. Une méthode d'intégration est dite *A*-stable si sa zone de stabilité comprend tout le demi-plan gauche du plan complexe λT. Ceci se traduit par le fait que si un système continu stable est discrétisé avec une méthode *A*-stable, la simulation discrète est nécessairement stable et ce, peu importe le pas de calcul employé pour la discrétisation. Ce pas de calcul peut alors être choisi en se basant uniquement sur des considérations de précision.

Bien qu'elle soit *A*-stable, la discrétisation trapézoïdale comporte une caractéristique bien visible sur la figure 3-2b, en ce sens que la frontière de sa zone de stabilité est en fait l'axe imaginaire du plan complexe. Si un pôle du système se retrouve sur cette frontière, la simulation est oscillatoire. La discrétisation trapézoïdale est en fait connue pour osciller dans certaines conditions. Dans des systèmes électriques, cette condition se produira par exemple lors d'une discontinuité dans le courant d'une inductance ou la tension d'un condensateur.

Les méthodes d'intégration sus-mentionnées n'ont pas été choisies au hasard. Nous venons de voir qu'elles sont toutes les trois *A*-stable, une propriété appréciée au niveau de la stabilité des simulations. Elles sont également toutes les trois implicites. De façon générale, une méthode d'intégration implicite possède une zone de stabilité beaucoup plus large qu'une méthode explicite du même ordre. De plus, aucune méthode explicite ne peut être *A*-stable. Cette raison justifie le fait que les trois méthodes du tableau 3-1, toutes implicites, soient les plus utilisées dans la simulation à pas fixe des systèmes rigides.

Le fait qu'une méthode d'intégration soit *A*-stable assure que cette méthode pourra simuler de façon stable un système rigide. Cependant, il existe un autre type de stabilité

moins exigeant que la *A*-stabilité et qui permet d'assurer la simulation stable de systèmes rigides. Il s'agit de la stabilité rigide, traduction libre de l'expression anglophone "stiff stability". En se référant à la figure 3-3, une méthode d'intégration est rigidement stable si sa zone de stabilité contient les régions *R* et *S* et que la méthode produit des résultats précis lorsque les produits λT sont dans la région *S* et que la partie réelle des pôles λ est négative. Une méthode d'intégration rigidement stable permet de choisir le pas de calcul *T* de sorte que la dynamique dominante (lente) se retrouve dans la zone *S* et la dynamique rapide soit automatiquement accommodée par la zone *R*. Cette zone R correspond en fait à une zone *A*-stable décalée vers la gauche.

Un commentaire s'impose sur la figure 3-3. On y remarque que la zone *S* déborde dans le demi-plan droit, qui est une zone d'instabilité. La réponse d'un système qui comprend un pôle dans le demi-plan droit sera un signal exponentiel croissant (pôle purement réel) ou une sinusoïde contenue dans une enveloppe exponentielle croissante (pôles conjugués complexes). Avec une méthode rigidement stable, on cherche à obtenir dans la portion de *S* qui se trouve dans le demi-plan droit une simulation *relativement* stable. Il s'agit en fait de simuler avec un minimum de précision l'instabilité du système. On choisit alors le pas de calcul de telle sorte que l'exponentielle ne croisse pas trop rapidement.

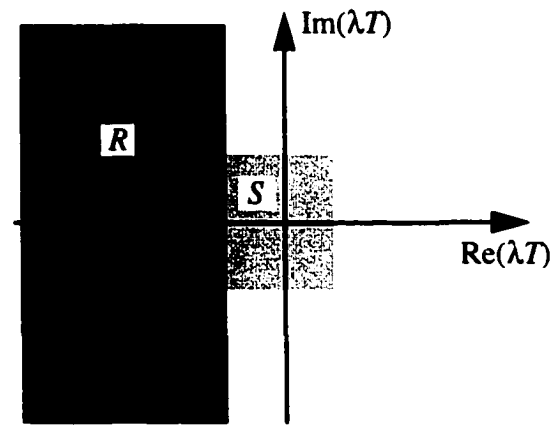


Figure 3-3 Zone de stabilité d'une méthode d'intégration rigidement stable.

3.2.2 Évaluation de la précision des méthodes d'intégration

La stabilité d'une simulation utilisant une méthode d'intégration à pas fixe est un critère de performance important. Cependant, nous avons très peu parlé jusqu'ici de la précision des résultats. Une des méthodes couramment utilisées pour développer les méthodes d'intégration linéaires à k pas consiste à utiliser l'expansion en série de Taylor. La série de Taylor permettant d'obtenir x_{n+1} à partir de x_n et de ses dérivées s'exprime tel que montré en (3-4) et la dérivée de cette dernière est montrée en (3-5).

$$x_{n+1} = x_n + T\dot{x}_n + \frac{T^2}{2!}\ddot{x}_n + \ldots + \frac{T^m}{m!}x_n^{(m)} \tag{3-4}$$

$$\dot{x}_{n+1} = \dot{x}_n + T\ddot{x}_n + \frac{T^2}{2!}\dddot{x}_n + \ldots + \frac{T^m}{m!}x_n^{(m+1)} \tag{3-5}$$

L'erreur de troncation locale (LTE) est définie comme étant l'erreur produite en un pas de calcul par une méthode d'intégration discrète donnée, en supposant une précision infinie. Cette erreur peut être définie en termes de (3-1):

$$LTE = \sum_{j=0}^{k} (\alpha_j x_{n+j} - T\beta_j \dot{x}_{n+j}) \tag{3-6}$$

En insérant (3-4) et (3-5) dans (3-6) et en ré-arrangeant les termes, on obtient aussi l'expression suivante pour le LTE:

$$LTE = C_0 x_n + C_1 T\dot{x}_n + C_2 T^2 \ddot{x}_n + \ldots \tag{3-7}$$

Les équations de Lambert définissent les relations entre les coefficients C et les α et les β. Ces équations sont données par (3-8).

$$
\begin{aligned}
C_0 &= \alpha_0 + \alpha_1 + \alpha_2 + \ldots + \alpha_k \\
C_1 &= \alpha_1 + 2\alpha_2 + \ldots + k\alpha_k - (\beta_0 + \beta_1 + \ldots + \beta_k) \\
&\ldots \\
C_q &= \frac{1}{q!}(\alpha_1 + 2^q\alpha_2 + \ldots + k^q\alpha_k) - \frac{1}{(q-1)!}(\beta_1 + 2^{q-1}\beta_2 + \ldots + k^{q-1}\beta_k)
\end{aligned}
\tag{3-8}
$$

Une méthode d'intégration est dite de précision d'ordre p si les relations suivantes sont respectées.

$$
\begin{aligned}
C_0 = C_1 = \ldots = C_p = 0 \\
C_{p+1} \neq 0
\end{aligned}
\tag{3-9}
$$

Le terme C_{p+1} est souvent défini comme étant la constante d'erreur. Cette constante ainsi que le terme de (3-7) qui l'accompagne sont identifiés comme étant l'erreur de troncation locale principale. Le tableau 3-2 indique le LTE principal pour les trois méthodes étudiées à la section précédente.

Tableau 3-2

Termes principaux du LTE pour trois méthodes courantes

Nom de la méthode	LTE principal
Euler arrière ou Gear-1	$-\frac{T^2\ddot{x}}{2}$
Tustin ou trapézoïdale	$-\frac{T^3\dddot{x}}{12}$
Gear-2	$-\frac{2T^4x^{(4)}}{9}$

Bien qu'étant toutes les deux des méthodes à un pas, on voit au tableau 3-2 que les méthodes trapézoïdale et Euler arrière sont de précision d'ordres différents. La méthode Euler arrière est de précision d'ordre deux alors que la discrétisation trapézoïdale est de précision d'ordre trois. La discrétisation trapézoïdale est d'ailleurs la méthode à un pas la plus précise. La méthode Gear-2 est quant à elle de précision d'ordre quatre, mais cette précision additionnelle a un prix, soit un algorithme sensiblement plus complexe que les deux méthodes à un pas.

3.3 Discrétisation trapézoïdale appliquée aux équations d'état

La discrétisation trapézoïdale étant *A*-stable, précise et simple, nous avons choisi de l'utiliser comme méthode d'intégration dans nos travaux. Cependant, comme nous utilisons la méthode avec un système d'équations d'état, nous devons d'abord l'exprimer sous forme matricielle plutôt que sous forme de fonction de transfert. Nous avons identifié deux implantations de la discrétisation trapézoïdale sous forme matricielle dans un contexte d'équations d'état.

La première méthode est tirée de [32]. Cette méthode est utilisée par la commande `c2d` dans Matlab, lorsque l'option `'Tustin'` est spécifiée. La discrétisation trapézoïdale étant une méthode implicite, cette implantation est basée sur un changement de variables où l'on définit un nouveau vecteur d'état. Le développement présenté dans [32] est valable pour des systèmes invariants dans le temps. Nous avons essayé de développer les mêmes équations, mais cette fois pour un système variant dans le temps, et nous ne parvenons pas à un résultat concluant. Nous en concluons que le changement de variables proposé n'est pas valable pour un système variant dans le temps. D'autre part, avec cette méthode, le changement de variables impose de discrétiser non seulement les matrices d'état A et B, mais aussi les matrices C et D, ce que nous considérons comme un inconvénient non-négligeable.

La seconde implantation de la discrétisation trapézoïdale des matrices d'état provient de [20]. Encore une fois, nous devons modifier légèrement la démarche pour accommoder notre système d'équations d'état, qui est variant dans le temps. La première étape consiste à remanier la fonction de transfert discrète correspondant à la discrétisation trapézoïdale de la manière suivante.

$$x_{n+1} = x_n + \frac{T}{2}(\dot{x}_{n+1} + \dot{x}_n) \tag{3-10}$$

On substitue ensuite aux deux dérivées de (3-10) les relations suivantes.

$$\begin{aligned} \dot{x}_{n+1} &= A_{n+1}x_{n+1} + B_{n+1}u_{n+1} \\ \dot{x}_n &= A_n x_n + B_n u_n \end{aligned} \tag{3-11}$$

En ré-arrangeant le résultat de cette substitution, on obtient (3-12).

$$\left(I - \frac{A_{n+1}T}{2}\right)x_{n+1} = \left(I + \frac{A_n T}{2}\right)x_n + \left(\frac{B_n T}{2}\right)u_n + \left(\frac{B_{n+1}T}{2}\right)u_{n+1} \tag{3-12}$$

On remarque dans (3-12) que x_{n+1} ne dépend que de valeurs passées des états. Cependant, il dépend encore de l'entrée u_{n+1}. Bien que ceci ne pose pas de problème en soi, nous devons porter une attention particulière à l'implantation de cette variante de la discrétisation trapézoïdale, particulièrement si l'on utilise les s-function de Simulink.

L'équation (3-12) est de la forme $Ax = b$, où x et b sont des vecteurs. À ce stade-ci, nous avons deux choix pour trouver la solution de (3-12). La première solution consiste à inverser la matrice à gauche de x_{n+1} dans (3-12) puis à prémultiplier tout ce qui est à droite de l'égalité par cette matrice inversée. Le résultat est (3-13).

$$x_{n+1} = \left(I - \frac{A_{n+1}T}{2}\right)^{-1}\left[\left(I + \frac{A_nT}{2}\right)x_n + \left(\frac{B_nT}{2}\right)u_n + \left(\frac{B_{n+1}T}{2}\right)u_{n+1}\right] \qquad (3\text{-}13)$$

Une autre solution consiste à résoudre le système d'équations linéaires (3-12). Nous avons choisi d'utiliser la décomposition LU pour résoudre (3-12). La factorisation LU est en fait une forme d'élimination gaussienne. Il s'agit essentiellement de décomposer une matrice carrée générale, dense et sans structure particulière qui puisse être exploitée (par exemple la symétrie) en deux matrices triangulaires L ("lower triangular", ou triangulaire inférieure) et U ("upper triangular", ou triangulaire supérieure). L'utilisation de matrices triangulaires est souhaitable parce qu'elle permet de solutionner simplement des systèmes d'équations linéaires [33]. Voyons comment solutionner le système d'équations linéaires (3-14) à l'aide de la factorisation LU.

$$Ax = b \qquad (3\text{-}14)$$

La première étape de notre algorithme consiste donc à décomposer la matrice A en un produit de deux matrices L et U, tel que montré en (3-15).

$$PA = LU \qquad (3\text{-}15)$$

Dans (3-15), la matrice P est une matrice de permutation et consiste en fait en une matrice identité dont les rangées ont été permutées. Cette permutation est requise afin d'assurer la stabilité de l'algorithme de la factorisation LU. Une fois la décomposition (3-15) obtenue, le système suivant est solutionné et un vecteur intermédiaire y est obtenu.

$$Ly = Pb \qquad (3\text{-}16)$$

Le système d'équations (3-16) pourrait, par exemple, ressembler à (3-17) pour un système d'ordre trois où la factorisation ne requiert pas de permutation (P est la matrice identité).

$$\begin{bmatrix} l_{11} & 0 & 0 \\ l_{21} & l_{22} & 0 \\ l_{31} & l_{32} & l_{33} \end{bmatrix} \begin{bmatrix} y_1 \\ y_2 \\ y_3 \end{bmatrix} = \begin{bmatrix} b_1 \\ b_2 \\ b_3 \end{bmatrix} \tag{3-17}$$

Le vecteur de solution y est trouvé en commençant par y_1 et en procédant vers le bas. Cette phase de l'algorithme est ainsi baptisée la substitution avant. Une fois obtenu le vecteur y, il suffit de solutionner le système (3-18) pour obtenir le vecteur de la solution finale x.

$$Ux = y \tag{3-18}$$

Poursuivant avec le même exemple, l'allure de (3-18) est la suivante.

$$\begin{bmatrix} u_{11} & u_{12} & u_{13} \\ 0 & u_{22} & u_{23} \\ 0 & 0 & u_{33} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} y_1 \\ y_2 \\ y_3 \end{bmatrix} \tag{3-19}$$

Cette fois, la solution de (3-19) commence par le bas, soit avec l'élément x_3, et procède vers le haut, d'où l'appellation de cette phase, substitution arrière.

Dans le cas que nous venons de décrire, x et b sont des vecteurs. Cependant, si ces deux paramètres sont des matrices (X et B), il suffit de factoriser A exactement de la même manière que ci-haut et ensuite, trouver la solution colonne par colonne. Dans le cas où B est une matrice identité, la solution X sera A^{-1}. La factorisation LU peut donc être

utilisée pour résoudre des systèmes d'équations linéaires et aussi pour inverser des matrices. La raison pour laquelle nous avons opté pour la solution d'un système d'équations linéaires est simplement la performance des calculs en termes de vitesse. En effet, la factorisation LU est de complexité mathématique $O(2N^3/3)$. Le nombre exact d'opérations mathématiques requises pour solutionner (3-12) est compté à la figure 3-4.

Nous avons travaillé longtemps sur un système qui compte onze variables d'état et une seule entrée. Dans ce cas, la solution de (3-12) requiert 1378 opérations mathématiques selon le résultat de la figure 3-4. Si l'on utilise plutôt l'inversion de matrice (3-13), les quatre premières opérations de la figure 3-4 sont inchangées. Cependant, l'inversion de matrice qui utilise la factorisation LU revient alors à solutionner un système de type $AX = B$, où tous les paramètres sont des matrices d'ordre p. En tenant compte du fait que B est la matrice identité, le nombre d'opérations mathématiques requises pour solutionner (3-13) est illustré à la figure 3-5.

$$\overbrace{\underbrace{\left(I - \frac{A_{n+1}T}{2}\right)}_{5^\circ} x_{n+1} = \underbrace{\left(I + \frac{A_nT}{2}\right)}_{1^\circ} x_n + \underbrace{\left(\frac{B_nT}{2}\right)}_{2^\circ} u_n + \underbrace{\left(\frac{B_{n+1}T}{2}\right)}_{3^\circ} u_{n+1}}^{6^\circ}$$

m: nombre d'entrées
p: nombre d'états

1 - Produit matrice (p par p) par vecteur (p par 1):	$2p^2$
2 - Produit matrice (p par m) par vecteur (m par 1):	$2pm$
3 - Produit matrice (p par m) par vecteur (p par 1):	$2pm$
4 - Somme de 3 vecteurs (p par 1):	$3p$
5 - Factorisation LU d'ordre p:	$\frac{2}{3}p^3 - \frac{1}{2}p^2 - \frac{5}{6}p - 1$
6 - Solution LU d'ordre p de type $Ax = b$:	$2p^2$
Total:	$\frac{2}{3}p^3 + \frac{7}{2}p^2 + 4mp + \frac{13}{6}p - 1$

Figure 3-4 Nombre d'opérations mathématiques requises pour solutionner (3-12) (factorisation et solution LU)

$$x_{n+1} = \underbrace{\left(I - \frac{A_{n+1}T}{2}\right)^{-1}}_{5^\circ}\Bigg[\underbrace{\underbrace{\left(I + \frac{A_nT}{2}\right)x_n}_{1^\circ} + \underbrace{\left(\frac{B_nT}{2}\right)u_n}_{2^\circ} + \underbrace{\left(\frac{B_{n+1}T}{2}\right)u_{n+1}}_{3^\circ}}_{4^\circ}\Bigg]$$

6°

m: nombre d'entrées
p: nombre d'états

1 - Produit matrice (p par p) par vecteur (p par 1):	$2p^2$
2 - Produit matrice (p par m) par vecteur (m par 1):	$2pm$
3 - Produit matrice (p par m) par vecteur (p par 1):	$2pm$
4 - Somme de 3 vecteurs (p par 1):	$3p$
5 - Inversion de matrice avec factorisation LU d'ordre p:	$2p^3 - \frac{3}{2}p^2 + \frac{3}{2}p - 1$
6 - Produit matrice (p par p) par vecteur (p par 1):	$2p^2$
Total:	$2p^3 + \frac{3}{2}p^2 + 4pm + \frac{9}{2}p - 1$

Figure 3-5 Nombre d'opérations mathématiques requises pour solutionner (3-12) (inversion de matrice à l'aide d'une factorisation LU).

Pour le même système que ci-haut (onze états et une entrée), la solution de (3-13) requiert 2936 opérations mathématiques selon le résultat de la figure 3-5, soit plus de deux fois plus que la solution de (3-12). C'est précisément pour cette raison que nous avons adopté la méthode utilisant la solution du système d'équations plutôt que celle qui inverse explicitement la matrice.

3.4 Implantation de la discrétisation trapézoïdale

Comme notre simulation est réalisée à l'aide d'une s-function dans Matlab/Simulink, nous devons décrire comment une simulation se déroule dans Simulink, lorsque cette simulation comprend une s-function. La compréhension de ce mécanisme est requise car notre problème diffère subtilement du problème type pour lequel l'interface entre Simulink et la s-function est prévu.

Une s-function est un bloc spécial de la librairie de Simulink. Comme tous les autres blocs, il est pourvu d'un port d'entrée u, d'un port de sortie y et peut contenir ou non des variables d'état x. Cependant, le comportement du bloc s-function est dicté par un programme associé au bloc. Ce programme contient plusieurs fonctions. Pour une simulation discrète à pas fixe, la fonction d'initialisation de la s-function est d'abord exécutée en début de simulation, puis à chaque pas de calcul, deux fonctions sont exécutées tour à tour. D'abord, la fonction qui calcule le vecteur des sorties de la s-function est exécutée, puis la fonction qui met à jour le vecteur des états discrets est exécutée. Ce mécanisme part du principe qu'en début de simulation, au temps t=0, on spécifie les valeurs initiales du vecteur des états comme suit.

$$x(0) = x_0 \tag{3-20}$$

Le fait que la fonction qui met à jour les sorties soit appelé avant la fonction qui met à jour les états est basé d'une part sur le fait que les conditions initiales soient celles des états et d'autre part sur l'hypothèse que les équations d'état discrètes se présentent sous la forme dite normale (3-21).

$$\begin{aligned} x(n+1) &= A\,x(n) + B\,u(n) \\ y(n) &= C\,x(n) + D\,u(n) \end{aligned} \tag{3-21}$$

Cet exemple est basé sur le fait que le système est linéaire et invariant dans le temps. Une simulation dont les équations d'état sont solutionnées par une s-function et sont exprimées telles qu'en (3-21) se déroule donc de la façon suivante. On spécifie $x(0)$ dans la fonction d'initialisation. Connaissant $x(0)$ et aussi $u(0)$, on calcule ensuite $y(0)$. En étant toujours à n=0, on calcule $x(n+1)$, soit x(1), pour le prochain pas de calcul. Au pas suivant, on calcule $y(1)$ puis $x(2)$, et ainsi de suite.

Or, notre but est d'implanter une simulation qui utilise la discrétisation trapézoïdale décrite par (3-12). Il est clair que (3-12) ne peut s'exprimer sous la forme normale (3-21) puisque le vecteur des états x au moment $(n+1)$ dépend du vecteur des entrées u au moment $(n+1)$. De plus, notre système est variant dans le temps et les matrices A, B, C et D changent. Il faut faire attention à l'ordre des opérations (changement des matrices, calcul des sorties et calcul des états) afin que le tout soit cohérent. Les fonctions standards de mise à jour des sorties et de mise à jour des états de la s-function doivent par conséquent être utilisées de façon différente de leur usage habituel.

Dans notre implantation, nous n'avons simplement pas utilisé la fonction de mise à jour des états et nous faisons tous les calculs dans la fonction de mise à jour des sorties. La figure 3-6 illustre sous forme d'organigramme le contenu de la fonction de mise à jour des sorties de notre s-function. Cette fonction est appelée à chaque pas de calcul. Les numéros des équations sont indiqués lorsqu'applicable. Au démarrage de la simulation, tous les interrupteurs sont ouverts et les valeurs initiales des matrices d'état du convertisseur sont calculées dans ces conditions. La vitesse initiale de la machine est aussi spécifiée. Nous sommes donc en mesure de calculer les matrices d'état globales de l'entraînement. Nous devons également calculer une portion de l'équation (3-12) qui sera requise lors de la mise à jour des états au prochain pas de calcul. On calcule ensuite les sorties de la s-function et le premier pas de calcul est complété. Pour tous les pas de calculs subséquents, la boucle de simulation est toute la colonne de gauche de la figure 3-6. La première tâche à accomplir dans cette boucle est de vérifier si les interrupteurs du convertisseur changent d'état. Si le convertisseur change d'état, les matrices d'état du convertisseur sont recalculées selon la méthode exposée à la section 2.1.2. On effectue ensuite la mise à jour des termes de la matrice d'état A_m de la machine qui contiennent le terme de vitesse de la machine ω_r. Il suffit ensuite de compléter les matrices d'état A et B finales englobant tout l'entraînement selon la méthode décrite à la section 2.3 et de procéder à la mise à jour des états discrets de la partie électrique du système. Une fois les variables d'état électriques modifiées, on calcule le couple électromagnétique de la machine et

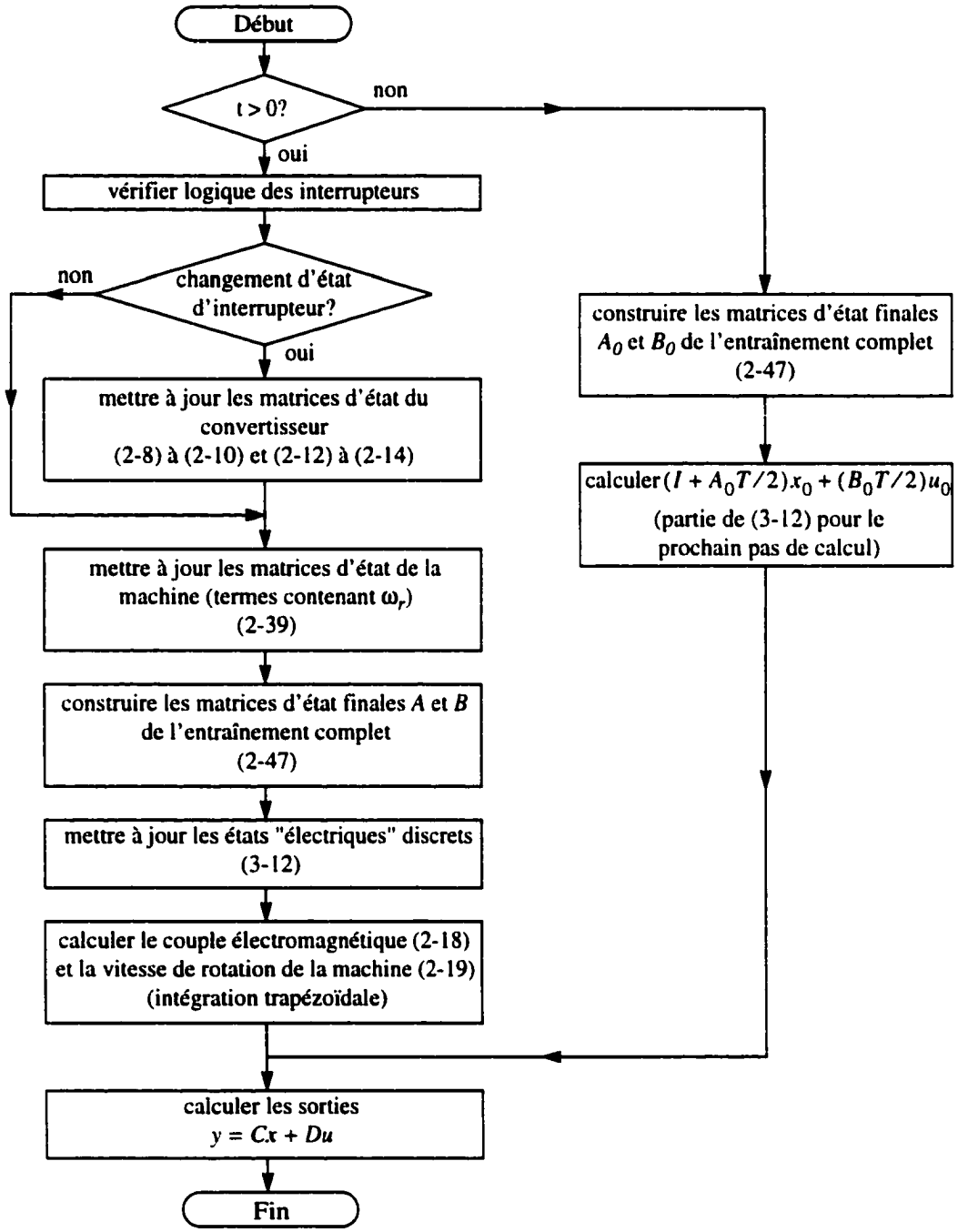


Figure 3-6 Organigramme de la fonction de mise à jour des sorties de la s-function (discrétisation trapézoïdale).

on met à jour la seule variable d'état mécanique du système, soit la vitesse de rotation du rotor. Notons que la position angulaire du rotor n'est pas requise puisque nous utilisons une transformation dans le référentiel stationnaire. Enfin, la dernière étape de chaque pas de calcul consiste à calculer le vecteur des sorties *y* de la s-function.

3.5 La méthode MSRP-2

Bien que nous ayons choisi d'utiliser la discrétisation trapézoïdale comme méthode d'intégration, nous avons également étudié dans le cadre de nos travaux une méthode d'intégration matricielle publiée récemment et baptisée "Matrix Stability Region Placement" (MSRP) [19], ou méthode de placement de région de stabilité. Cette méthode explicite est caractérisée par le fait que les coefficients d'intégration sont des matrices au lieu de scalaires. Il n'y a donc pas d'intégrateur local pour chaque état. L'intégrateur associé à chaque état tient compte de tous les autres états. De plus, la méthode est basée sur une technique de placement de pôles qui permet une transposition exacte ("mapping") des pôles principaux du plan *s* au plan *z*, avec la relation bien connue suivante.

$$z = e^{sT} \tag{3-22}$$

Les auteurs mentionnent que lorsqu'appliquée à un système linéaire invariant dans le temps, la méthode MSRP donne une réponse temporelle exacte et une erreur nulle en régime permanent. L'application de cette méthode requiert que le processus à simuler soit décrit par des équations d'état. Si le système simulé est non-linéaire, on doit le linéariser autour d'un point d'opération et obtenir le Jacobien des matrices du système *J* et de couplage des entrées *G*. Sous forme matricielle, on peut alors décrire le système à simuler par l'équation d'état suivante:

$$\dot{x}(t) = Jx(t) + Gu(t) \tag{3-23}$$

L'algorithme d'intégration explicite de la méthode matricielle de placement de région de stabilité d'ordre deux (MSRP-2) est défini comme suit:

$$x_{n+2} = -A_1 x_{n+1} - A_0 x_n + T(B_1 \dot{x}_{n+1} + B_0 \dot{x}_n) \tag{3-24}$$

En substituant l'équation (3-23) dans l'équation (3-24) et en plaçant tous les faux pôles à l'origine du plan z, on obtient les contraintes de stabilité sur les coefficients A_i et B_i. Les équations de Lambert nous permettent d'obtenir les contraintes sur la précision de la simulation en boucle fermée et on obtient les expressions suivantes pour les coefficients A_i et B_i (I_N est la matrice identité N par N, où N est le nombre de variables d'état):

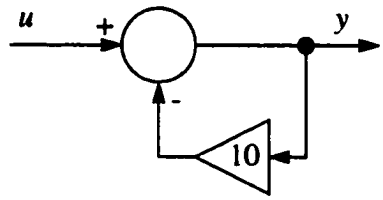
$$\begin{aligned} A_1 &= (e^{JT} - I_N)(JT)^{-1} - 2I_N & B_1 &= (A_1 + e^{JT})(JT)^{-1} \\ A_0 &= -(A_1 + I_N) & B_0 &= A_0(JT)^{-1} \end{aligned} \tag{3-25}$$

Les coefficients A_0, A_1, B_0 et B_1 sont des matrices N par N. Avant même de tester cette méthode, on peut déjà faire quelques commentaires à son sujet. D'abord, elle nécessite l'obtention de la matrice d'état du système à simuler (matrice J) ou son Jacobien si le système est non-linéaire. Ensuite, il est clair que la matrice J doit être non-singulière puisque le terme $(JT)^{-1}$ apparaît dans le calcul des coefficients. Ceci peut poser un problème dans certains cas. Cependant, il existe une généralisation de la méthode qui évite d'avoir à inverser la matrice J. Cette variante utilise plutôt une transformation de coordonnées pour solutionner le problème [19]. Enfin, il faut calculer l'exponentielle de JT. Tout ceci peut à priori sembler coûteux en termes de quantités de calcul. Cependant, les auteurs de la méthode ont fait une étude du gain en temps de calcul qui est fait si on utilise un intégrateur discret scalaire plutôt que l'opérateur matriciel proposé ici. En fait, il s'agit de quantifier la hausse de la période d'échantillonnage que la méthode *MSRP-2* permet par rapport à la période d'échantillonnage qui devait être utilisée avec un intégrateur scalaire. Par exemple, si un certain système doit être simulé avec une période

d'échantillonnage T de 10 ms avec la méthode *AB-2*, la méthode *MSRP-2*, plus complexe en terme de calculs, devient rentable si elle permet d'utiliser par exemple une période d'échantillonnage de 60 ms.

La principale raison qui a poussé les auteurs de la méthode MSRP à la développer est qu'ils simulent en temps réel des systèmes non-linéaires. L'exemple qui revient dans toutes leurs publications est la simulation d'un modèle de turbine d'avion. Mentionnons également que les simulations citées sont réalisées avec un pas de calcul de l'ordre de la milliseconde. Dans l'introduction de [19], ils affirment que les méthodes d'intégration implicites ne sont pas de bonnes candidates pour la simulation en temps réel de systèmes non-linéaires puisque leur nature implicite requiert une solution itérative à chaque pas de calcul et par conséquent, elles sont trop lentes. Les méthodes explicites sont, par contre, des candidates idéales pour la simulation en temps réel puisqu'avec une telle méthode, la mise à jour du vecteur des états discrets dépend uniquement de valeurs passées des états et des entrées et aucun processus itératif n'est requis pour trouver la solution. Le principal problème des méthodes explicites est que leurs régions de stabilité sont tellement plus petites que les méthodes implicites du même ordre qu'un pas de calcul prohibitivement petit doit être utilisé pour permettre une simulation stable. Lorsque comparée à des méthodes explicites telles Adams-Bashforth d'ordre deux, la méthode MSRP-2 permet de simuler des systèmes non-linéaires avec des pas de calcul tels que le calcul additionnel requis par la méthode est largement compensé par le grand pas de calcul qu'elle permet d'utiliser.

Nous avons cru pendant un certain temps que la méthode MSRP-2, étant explicite, nous permettrait d'éliminer certains délais que nous sommes obligés d'introduire dans nos simulations pour briser des boucles algébriques. La figure 3-7 illustre le phénomène de boucle algébrique, qui est généralement formée via une boucle de contre-réaction qui ne contient aucun délai. La sortie $y(n)$ du système est alors une fonction d'elle même.



$$y(n) = u(n) - 10y(n)$$

$$y(n) = \frac{u(n)}{11}$$

Figure 3-7 Une boucle algébrique simple et sa solution.

Dans l'exemple de la figure 3-7, la solution de la boucle algébrique est simple, mais cette situation n'est pas la norme. Par exemple, si l'on s'attarde au fonctionnement de notre s-function et que l'on analyse son interaction avec le reste du système (le contrôleur), on se rend compte qu'il peut y avoir une boucle algébrique. La figure 3-8 illustre l'interaction entre notre s-function et le contrôleur qu'on lui associe.

Notre s-function est elle même implicite, et ceci avant même que l'on considère le processus d'intégration. En effet, elle contient la logique des interrupteurs et change les matrices d'état si au moins un interrupteur change d'état. Conséquemment, les sorties $y(n)$ de la s-function sont des fonctions directes des entrées $u(n)$ de la s-function au même moment. Si les sorties du contrôleur $y_c(n)$ sont elles aussi fonctions des entrées du contrôleur $u_c(n)$ au même instant, une boucle algébrique est formée et sa solution n'est pas triviale puisque la boucle en question contient des discontinuités, en l'occurrence les

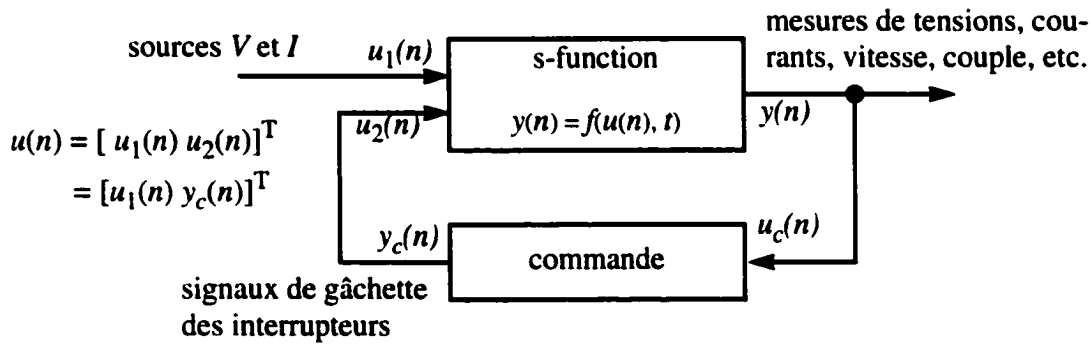


Figure 3-8 Interaction entre notre s-function et le contrôleur.

changements d'état des interrupteurs. Si cette condition se présente, nous devons ajouter un délai unitaire z^{-1} quelque part dans la boucle afin de la briser. Notre conclusion sur cet aspect est donc la suivante. Bien que l'utilisation d'une méthode d'intégration explicite puisse dans certains cas éviter des boucles algébriques qui seraient formées en utilisant une méthode d'intégration implicite, ce n'est pas notre cas, puisque notre système est implicite, et ce sans même tenir compte de l'intégration.

3.6 Étude de cas

Afin de déterminer si la méthode MSRP-2 est intéressante pour notre application, nous la comparons à la discrétisation trapézoïdale. Notre comparaison est de nature qualitative (formes d'ondes obtenues) et quantitative (temps de calcul). Nous utilisons le même système que celui utilisé au chapitre précédent, soit celui illustré à la figure 2-13. Cependant, nous utilisons cette fois-ci des s-function en langage Matlab plutôt qu'en langage pour faire la comparaison. Nous justifions ce choix par le fait que la méthode MSRP-2 requiert des fonctions plus complexes, notamment le calcul de l'exponentielle matricielle. Si la méthode MSRP-2 donne des meilleurs résultats que la discrétisation trapézoïdale avec la s-function en langage Matlab, alors nous prendrons le temps de la programmer en langage C.

La s-function avec laquelle nous implantons la méthode MSRP-2 ressemble beaucoup à celle utilisée pour l'implantation de la discrétisation trapézoïdale. Cependant, les calculs ne sont pas effectués dans le même ordre, étant donné que la méthode MSRP-2 est explicite et se prête mieux à la forme "standard" (3-21). La figure 3-9 illustre sous forme d'organigramme les tâches effectuées dans la s-function pour la méthode MSRP-2.

Nous avons donc réalisé deux simulations, une pour chaque méthode d'intégration. La figure 3-10 illustre les courants statoriques de la phase *a* obtenus avec les deux métho-

des, en régime permanent. On remarque que les courbes sont tout à fait comparables et la méthode MSRP-2 n'apporte rien de plus à la précision des courbes. La figure 3-11 illustre la vitesse de rotation de la machine obtenues avec les deux méthodes. Cette fois, les courbes sont superposées et elles sont à peu près confondues, mise à part une légère différence juste avant d'atteindre le régime permanent, ce qui nous permet de conclure encore une fois que la méthode MSRP-2 nous donne d'aussi bons résultats que la discrétisation trapézoïdale, sans être meilleurs.

Côté temps de calcul, le tableau 3-3 nous permet de constater que la méthode MSRP-2 est sensiblement plus lente que la discrétisation trapézoïdale. Or, les temps indi-

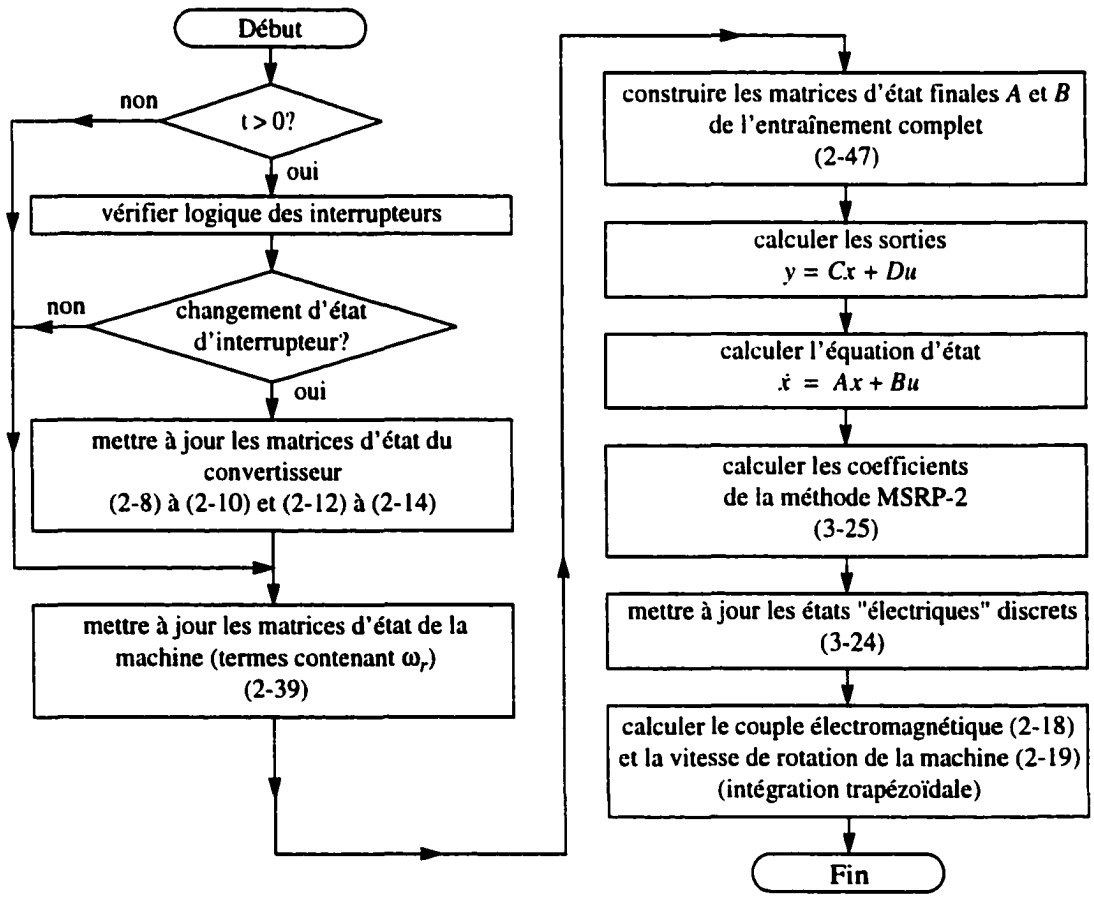


Figure 3-9 Organigramme de la fonction de mise à jour des sorties de la s-function (méthode MSRP-2).

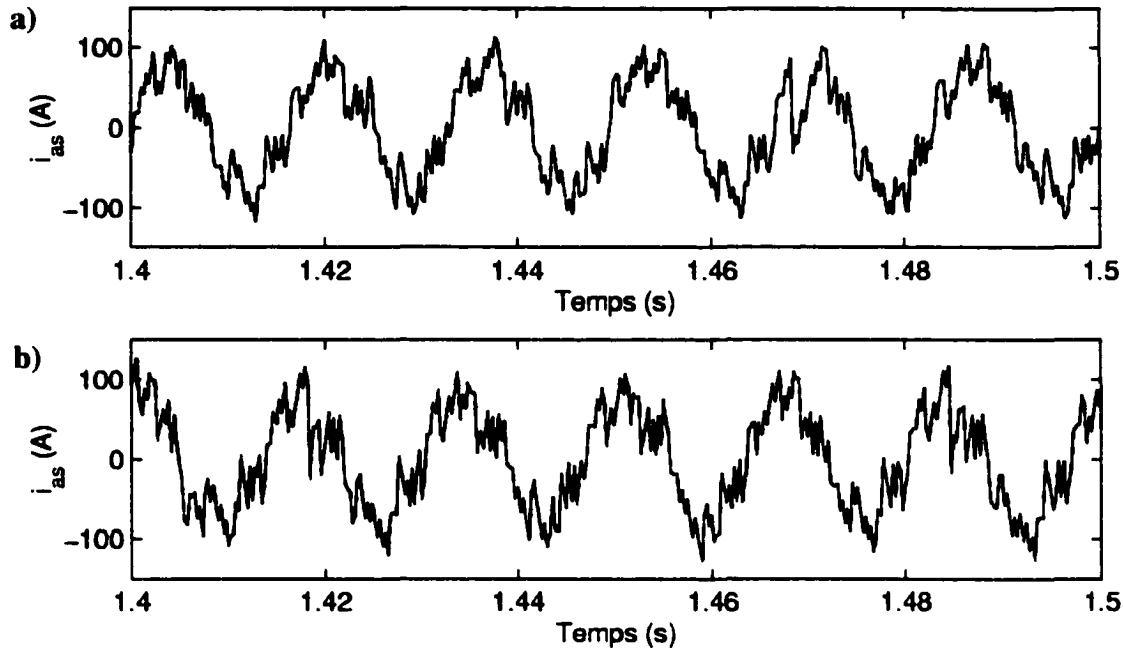


Figure 3-10 Courant statorique de la phase *a* de la machine avec les deux méthodes d'intégration: **a)** trapézoïdale; **b)** MSRP-2.

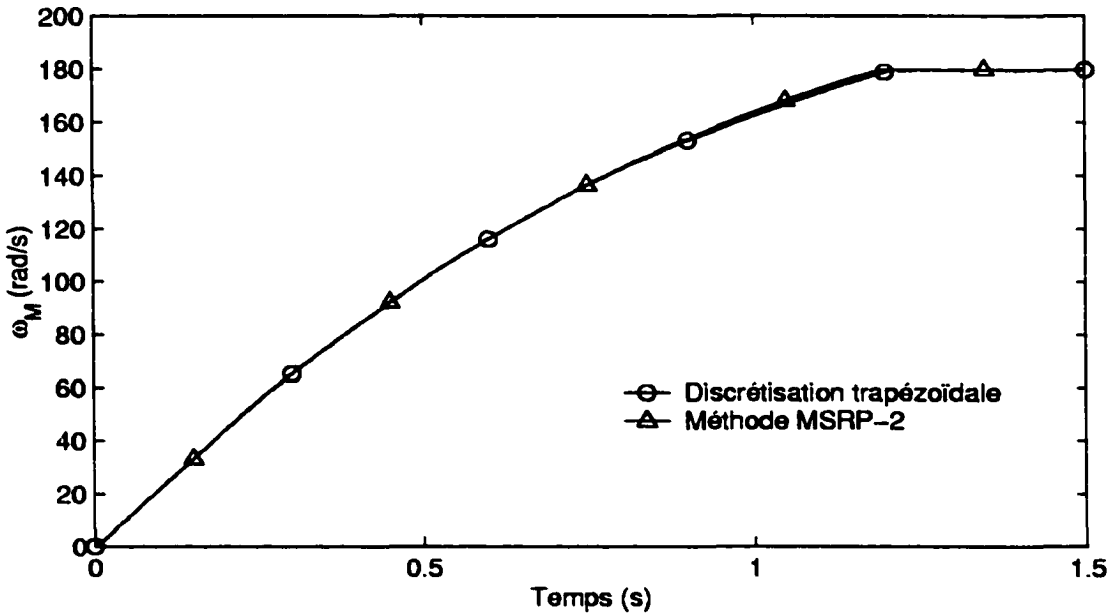


Figure 3-11 Vitesse de rotation de la machine avec les deux méthodes d'intégration.

qués dans ce tableau doivent être interprétés prudemment. En effet, cette simulation a été réalisée avec pas de calcul de 50 µs, ce qui veut dire que pour 1.5 seconde de simulation, il y a eu 30 000 pas de calcul. Or, il n'y a eu des changements d'état d'interrupteurs que dans 5 166 de ces pas de calcul, soit environ 17% du temps. La différence de durée des deux simulations a lieu exclusivement durant les pas où il y a eu des changements d'état puisque lorsqu'il n'y en a pas, les deux s-function font exactement les mêmes calculs, à savoir la mise à jour des vecteur des sorties y et des états x.

Tableau 3-3

Temps d'exécution des deux simulations

Méthode d'intégration	Temps d'exécution (s)
Trapézoïdale	336
MSRP-2	365

Avec cette façon de compter, la méthode MSRP-2 est donc plus lente que la discrétisation trapézoïdale par

$$\begin{aligned} \text{Différence de temps de simulation} &= \frac{365-336}{336} \times 100 \div 0.17 \\ &= 51\ \% \end{aligned} \tag{3-26}$$

Nous considérons ce résultat comme approximatif. Pour faire des mesures précises du temps de calcul, il nous faudrait programmer la s-function pour la méthode MSRP-2 en langage C et reprendre l'analyse. Or, suite aux résultats obtenus, nous ne voyons aucun intérêt à poursuivre cette voie.

3.7 Conclusions

Nous nous sommes concentrés dans ce chapitre à la question d'intégration à pas fixe des équations d'état liées aux entraînements électriques. Après avoir constaté que les équations différentielles de notre système sont rigides, nous avons passé en revue les caractéristiques des méthodes d'intégration les plus couramment utilisées pour résoudre cette classe de systèmes. Nous avons choisi de travailler avec la discrétisation trapézoïdale, étant donné sa grande précision et son algorithme relativement simple, puis avons décrit comment cette discrétisation a été implantée dans nos simulations. Nous avons par la suite décrit une méthode d'intégration récemment développée spécifiquement pour la simulation en temps réel d'équations différentielles rigides, soit la méthode MSRP-2. Une comparaison des résultats obtenus avec la méthode MSRP-2 et la discrétisation trapézoïdale nous permettent de conclure qua les deux méthodes donnent des résultats qui sont qualitativement très similaires. Cependant, la discrétisation trapézoïdale donnant des simulations plus rapides en temps différé, c'est avec cette méthode que nous allons continuer à travailler.

La validité de l'approche de simulation ainsi que de la méthode d'intégration choisie ayant été démontrées en temps différé, nous allons maintenant décrire la dernière phase de nos travaux, qui consiste à implanter le tout en temps réel sur un ordinateur multiprocesseurs.

CHAPITRE 4

SIMULATION EN TEMPS RÉEL

Ce dernier chapitre décrit les étapes qui nous permettent d'atteindre notre but ultime, soit de réaliser en temps réel des simulations d'entraînements électriques, lesquels sont modélisés à l'aide de l'approche par variables d'état. Il s'agit en fait d'intégrer les techniques des trois chapitres précédents en un tout cohérent.

Nous décrivons d'abord le système que nous tentons de modéliser et simuler en temps réel. Ce système consiste en un entraînement industriel commercialisé par la compagnie ABB. Nous décrivons ensuite brièvement les deux grandes architectures d'ordinateur parallèles sur lesquelles nous avons réalisé nos simulations. L'entraînement étudié étant relativement complexe, nous décrivons aussi une technique de découplage qui nous permet de simuler séparément les deux principales parties de l'entraînement. Afin de réduire le temps de calcul, nous tentons de séparer encore en deux les calculs associés à l'une des deux principales parties de notre simulation découplée. Nous identifions par la suite les principaux facteurs qui peuvent influencer la performance, en terme de vitesse de calcul, de nos simulations, après quoi nous décrivons les différentes optimisations que nous avons faites dans nos programmes. Enfin, nous présentons des résultats de simulations réalisées en temps réel sur un ordinateur parallèle.

4.1 Système étudié

Nous avons justifié notre démarche au début de cette thèse en spécifiant que l'un des aspects intéressants de la simulation en temps réel est qu'elle permet de réaliser des essais sur des commandes réelles de façon plus efficace et sécuritaire que si l'on fait ces mêmes essais sur le système réel. Il s'agit en fait de simuler en temps réel toute la partie haute puissance et de raccorder la commande au processus simulé. La simulation doit donc modéliser le plus fidèlement possible le processus à simuler. Dans cet esprit, nous avons choisi d'étudier un entraînement industriel commercialement disponible. Le contrôleur choisi est fabriqué par la compagnie ABB et est de la famille ACS 600. Notre choix est principalement motivé par le fait que plusieurs de ces contrôleurs sont disponibles dans nos laboratoires d'électrotechnique. Nous avons donc facilement accès à l'équipement ainsi qu'à toute la documentation associée. L'un des techniciens de l'É.T.S. ayant suivi une formation spéciale pour la mise en service et l'entretien de ces entraînements, nous avons accès à des plans détaillés du contrôleur. Ce dernier aspect est loin d'être négligeable puisque l'une des principales difficultés rencontrées dans la simulation des entraînements est l'obtention d'un ensemble complet de paramètres d'un système réel.

La modification du système consiste essentiellement à remplacer la batterie du schéma 2-12 par une source triphasée et un redresseur à diodes. La figure 4-1 illustre le schéma complet de l'entraînement. Les principaux paramètres associés à cet entraîne-

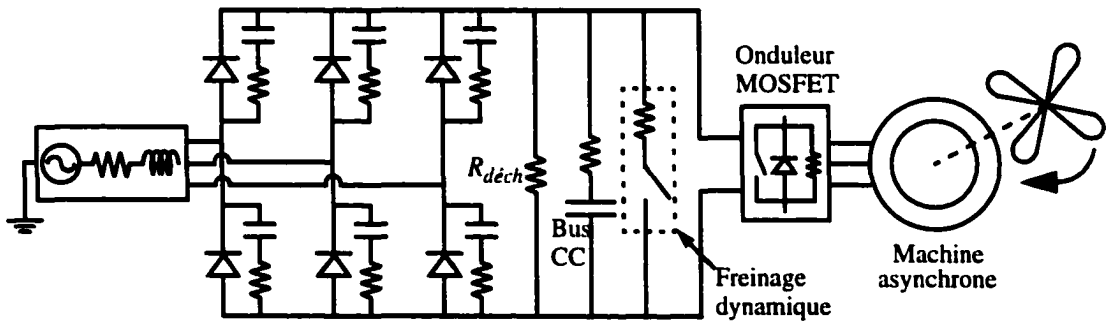


Figure 4-1 Schéma de l'entraînement étudié.

ment sont donnés à l'annexe B. La source comporte une impédance série composée d'une résistance et d'une inductance sur chaque phase. On note aussi la présence de circuits amortisseurs ("snubbers") aux bornes de chaque diode. Enfin, un interrupteur commandé en série avec une petite résistance font office de mécanisme de freinage dynamique. Ce mécanisme sert également à maintenir la tension du bus DC, qui a tendance à monter à des valeurs très élevées lorsque la machine change de mode de fonctionnement (de moteur à génératrice et vice-versa), lors de ralentissements et d'accélérations. Enfin, la commande utilisée pour la famille d'entraînements ACS 600 de ABB est une commande DFTC ("Direct Flux and Torque Control") tel que décrite au chapitre 2.

4.2 Ordinateurs parallèles

Comme notre but est de réaliser des simulations en temps réel, nous devons nous attarder à la question du matériel sur lequel ces simulations sont exécutées, c'est-à-dire l'ordinateur. Bien que ce ne soit pas encore évident à ce stade-ci, le système que nous désirons simuler est trop complexe pour être simulé en temps réel sur un seul processeur, avec un pas de calcul raisonnable. Nous devons donc utiliser un ordinateur à plusieurs processeurs, ou ordinateur parallèle. Il existe principalement deux grandes familles d'architectures d'ordinateurs parallèles, à savoir:

a. les architectures à mémoire distribuée, ou à passage de messages;
b. les architectures à mémoire partagée.

Comme nous avons eu l'occasion de travailler sur les deux types d'architectures dans le cadre de nos travaux, nous les décrivons sommairement afin d'identifier leurs principales caractéristiques.

4.2.1 Architectures à mémoire distribuée

Un ordinateur parallèle à mémoire distribuée est généralement caractérisé par un certain nombre de processeurs comportant chacun une plage de mémoire privée, mémoire qui est inaccessible aux autres processeurs. Comme il arrive que plusieurs processeurs aient besoin des mêmes données pour accomplir leurs tâches respectives, les processeurs sont reliés par des liens de communication permettant l'échange des données entre eux. La figure 4-2 illustre un ordinateur parallèle à mémoire distribuée comportant quatre processeurs. Cette figure nous permet d'identifier les principales caractéristiques de cette architecture. D'abord, chaque processeur dispose d'un nombre fini de liens de communication. La quantité de liens de communication de chaque processeur est une contrainte sur la topologie d'interconnexion des processeurs. Dans notre exemple de la figure 4-2, chaque processeur dispose d'au moins trois liens de communication et il est possible dans ces conditions d'avoir une topologie dite complètement connectée, où chaque processeur est capable de communiquer directement avec tous ses pairs. Or, avec un grand nombre de processeurs, il est difficile d'avoir une topologie complètement connectée et nous sommes contraints d'utiliser des topologies où chaque processeur n'est pas capable de communiquer directement avec tous ses pairs, mais doit passer par un ou plusieurs processeurs intermédiaires pour atteindre certains processeurs éloignés.

Mentionnons aussi que si d'autres processeurs doivent traiter les données se trouvant sur un processeur en particulier, ces données doivent être transmises d'un processeur

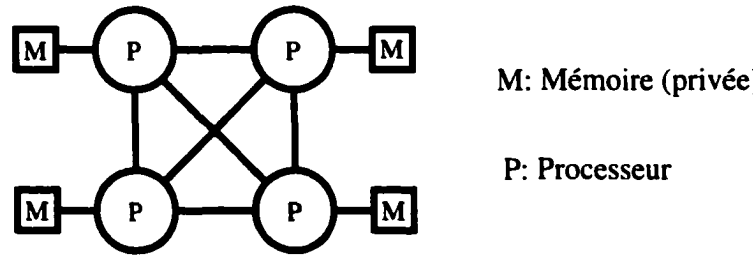


Figure 4-2 Ordinateur parallèle à mémoire distribuée.

à l'autre. Si beaucoup de données doivent transiter entre les processeurs, le temps de communication associé à la transmission de ces données peut devenir excessif et dominer le temps de calcul des processeurs. Cette architecture est donc performante si l'on parvient à minimiser la quantité de données transmises entre les processeurs à chaque pas de calcul.

L'ordinateur à mémoire distribuée avec lequel nous avons eu la chance de travailler est un ordinateur spécialement conçu pour la simulation en temps réel des réseaux électriques. Il s'agit en fait d'un simulateur Hypersim, développé par l'IREQ et commercialisé par TransÉnergie Technologies, la filiale commerciale de TransÉnergie. Cet ordinateur est composé de 15 processeurs DEC Alpha avec une horloge de 533 MHz. Chaque processeur dispose de six liens de communication. Nous décrirons plus en détail notre implantation sur cet ordinateur dans une section ultérieure.

4.2.2 Architectures à mémoire partagée

Un ordinateur parallèle à mémoire partagée consiste en un certain nombre de processeurs ayant tous accès à une zone de mémoire unique. La figure 4-3 illustre cette architecture et permet d'en dégager les principales caractéristiques. D'abord, on remarque en comparant à l'architecture à mémoire distribuée que l'accès par plusieurs proces-

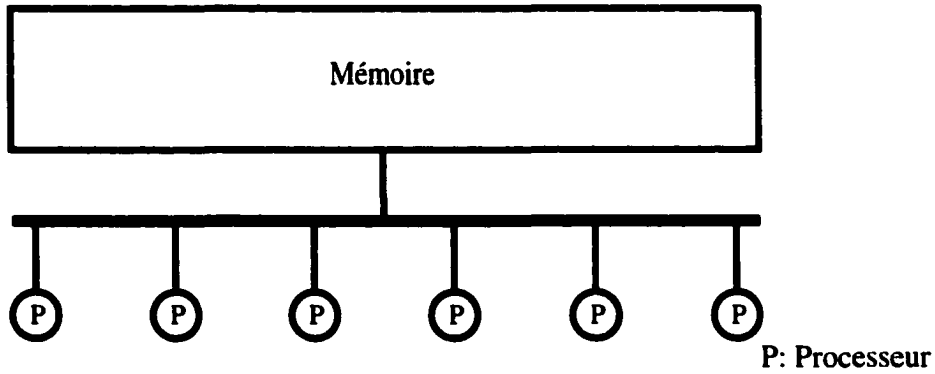


Figure 4-3 Ordinateur parallèle à mémoire partagée.

seurs à une même donnée ne nécessite pas de communication interprocesseur. Les accès à la mémoire se font via un bus commun à tous les processeurs. Dans une architecture à mémoire partagée "pure", ceci cause des problèmes lorsque plusieurs processeurs désirent accéder à la mémoire en même temps. Le bus ne pouvant être contrôlé que par un seul processeur à la fois, les autres processeurs désirant accéder à la mémoire doivent attendre que le contrôle du bus devienne disponible. S'il arrive régulièrement durant une simulation que plusieurs processeurs tentent d'accéder de façon simultanée à la mémoire, la performance sera grandement réduite. Cette situation est souvent appelée une contention de bus. Il existe toutes sortes de variantes de cette architecture permettant entre autres les accès simultanés suivants:

a. accès simultané à une zone mémoire unique par des processeurs différents en lecture seulement;
b. accès simultané à des zones mémoires différentes par des processeurs différents, en lecture ou en écriture.

Ces accès simultanés sont rendus possibles grâce à l'ajout dans l'ordinateur de matériel spécialisé d'accès à la mémoire. Nous tenons simplement à mentionner que ces possibilités existent et n'élaborerons pas la description de ce matériel.

L'ordinateur à mémoire partagée avec lequel nous avons travaillé est un SGI Origin 2000 comportant 12 processeurs MIPS R12000 avec une horloge de 400 MHz. Cet ordinateur ne possède pas une architecture à mémoire partagée "pure", mais plutôt une architecture dite "distributed shared memory", mémoire partagée distribuée, ou "virtual shared memory", mémoire partagée virtuelle. Ceci peut sembler paradoxal mais ces appellations veulent simplement dire que la mémoire est physiquement distribuée, alors que du point de vue du programmeur, elle apparaît comme un seul espace mémoire contigu. La figure 4-4 illustre l'architecture de l'ordinateur sur lequel nous avons travaillé. Nous avons con-

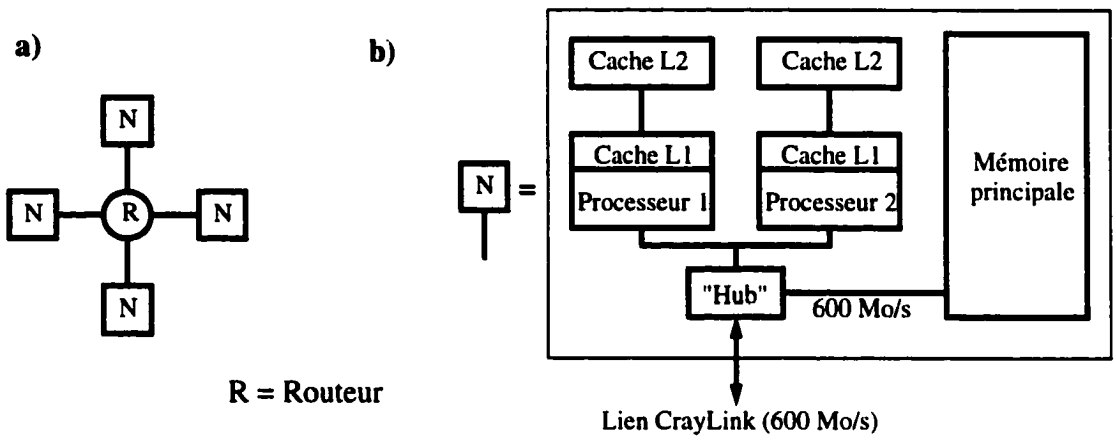


Figure 4-4 Architecture du SGI Origin 2000. **a)** système avec huit processeurs; **b)** contenu d'une carte ("Node board")

servé la notation de la littérature SGI [35] dans cette figure. Chaque symbole N représente une carte (dénommée "node board" par le fabricant) qui comprend deux processeurs. Chacun de ces processeurs dispose d'une antémémoire ("cache memory") de niveau un (L1) interne et d'une antémémoire de niveau deux (L2) externe à la puce du processeur. La mémoire cache associée à un processeur n'est pas accessible par les autres processeurs du système. Les deux processeurs disposent cependant d'une mémoire principale. Cette mémoire doit être accédée via une autre puce, dénommée "hub". Le "hub" contrôle l'accès à la mémoire principale non seulement pour les processeurs se trouvant sur cette carte, mais aussi pour les processeurs des autres cartes. Il est possible de raccorder deux "hub" ensemble directement et ainsi obtenir un système à quatre processeurs. Cependant, si l'on veut relier ensemble trois cartes ou plus, il faut utiliser une autre composante, à savoir un routeur (symbole R dans la figure 4-4). Chaque routeur dispose de six ports de communication et il est donc possible de raccorder en étoile douze processeurs (six cartes) à l'aide d'un seul routeur. Le système que nous avons utilisé correspond à la figure 4-4a.

Cette section visait simplement à présenter sommairement les deux ordinateurs parallèles avec lesquels nous avons expérimenté. Nous reviendrons sur leurs caractéristi-

ques spécifiques dans une section ultérieure, alors que nous décrirons l'implantation de nos simulations sur chacun.

4.3 Découplage

Dès nos premières simulations avec le montage de la figure 4-1, il est clair que le système complet ne peut être simulé en temps réel sur un seul processeur, avec un pas de calcul raisonnable. Afin de réduire le temps de calcul, une première forme de parallélisme est tentée. Il s'agit en fait de séparer ou découpler le système en plusieurs morceaux. Il existe diverses façons de découpler un système électrique. L'une des méthodes couramment utilisées dans la simulation des grands réseaux électriques consiste à utiliser les lignes de transports comme élément de découplage [34]. Les lignes de transport sont caractérisées par un délai de propagation proportionnel à la longueur de la ligne. Par exemple, si l'on applique un échelon de tension à une extrémité d'une ligne, l'échelon se propage dans la ligne et arrive à l'autre extrémité après un certain temps τ. Dans une simulation à pas fixe, il suffit que le délai de propagation dans ladite ligne soit au moins égal au pas de calcul utilisé pour la simulation afin que la ligne puisse servir d'élément de découplage sans affecter le résultat. La figure 4-5 illustre l'utilisation d'une ligne de transport comme élément de découplage dans la simulation d'un lien haute tension à courant continu (HTCC).

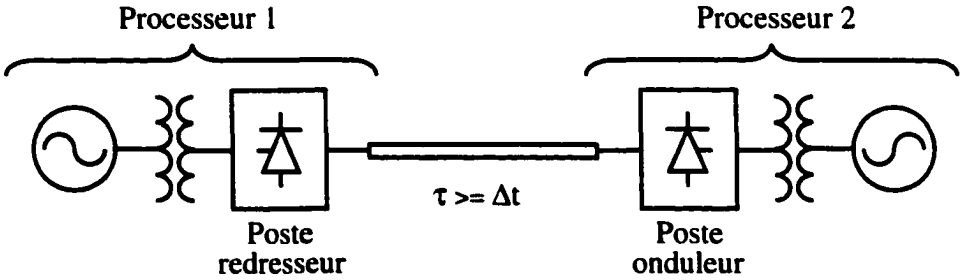


Figure 4-5 Découplage par l'utilisation d'une ligne de transport.

Cette forme de découplage géographique est très utilisée dans les simulateurs en temps réel de grands réseaux électriques. Dans le cas d'un entraînement électrique, l'ajout d'une ligne de transport assez longue pour que son délai de propagation soit de l'ordre du pas de calcul est difficile à justifier. Dans la plupart des cas, les composantes d'un entraînement sont assez proches les unes des autres. Dans le pire des cas, la source est à une extrémité d'une grande usine et la machine à l'autre extrémité, mais la distance n'est jamais assez importante pour que cette technique de découplage s'applique.

Une seconde technique de découplage consiste à couper un réseau électrique en un point judicieusement choisi et à calculer de part et d'autre de la coupure des équivalents de Thévenin et Norton, qui servent d'interface entre les deux sous-systèmes ainsi obtenus. Cette méthode est utilisée dans [16] pour interfacer un grand réseau, simulé avec l'approche nodale, avec un lien à haute tension à courant continu (HTCC), simulé avec l'approche par variables d'état. Les auteurs affirment que le succès de la méthode dépend de l'utilisation exclusive de quantités dites stables au point d'interface. Les quantités stables en question sont des courants traversant des inductances et des tensions aux bornes de condensateurs. La méthode est illustrée à la figure 4-6, où Z_1 est la résistance de Norton du sous-système #1 tandis que Z_2 est la résistance de Thévenin du sous-système #2.

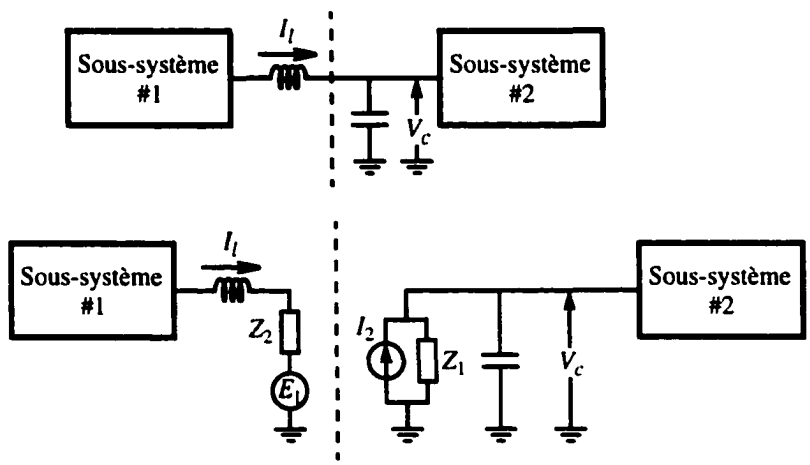


Figure 4-6 Découplage d'un réseau électrique à l'aide d'équivalents Thévenin et Norton

Ces résistances sont calculées en désactivant toutes les sources du système et en appliquant une impulsion de courant au point d'interface. On mesure la tension obtenue, que l'on divise par le courant appliqué pour obtenir la résistance. Ce processus doit être refait à chaque pas de calcul. Les amplitudes des sources sont calculées à partir de ces résistances et des valeurs de I_l et V_c au pas précédent. Les relations sont données dans [16]. Cette méthode est réputée comme étant précise, mais le processus de calcul des équivalents de Thévenin et Norton est difficilement réalisable dans une simulation en temps réel.

Nous avons donc utilisé une autre technique de découplage, qui est peu documentée, mais couramment utilisée dans le domaine de la simulation en temps réel des réseaux électriques. Il s'agit, comme dans la méthode des équivalents Thévenin et Norton, de mesurer un courant et une tension de part et d'autre de la coupure et de simplement transmettre cette mesure à une source commandée de l'autre côté de la coupure. La méthode est illustrée à la figure 4-7 pour notre entraînement. Bien que le courant servant à l'interfaçage des deux sous-systèmes ne soit pas un courant d'inductance comme tel, il est relativement stable.

4.4 Simulation parallèle de la partie onduleur/machine

À un certain stade de nos travaux, nous utilisions un montage différent de celui décrit dans cette thèse. La partie onduleur/machine comportait plus de variables d'état que le montage final et le temps de calcul était plus long. Par conséquent, nous n'étions

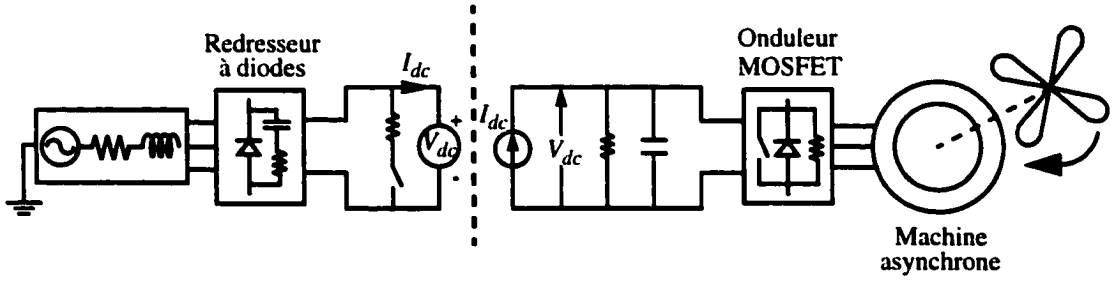


Figure 4-7 Méthode de découplage utilisée.

pas capable de faire de la simulation en temps réel. De plus, nous ne nous étions pas encore rendu compte que la mise à jour des états discrets était plus rapide en solutionnant un système d'équations plutôt qu'en inversant une matrice, comme il a été discuté à la section 3.3.

Nous avons donc essayé d'exploiter une autre forme de parallélisme. Les techniques de découplage décrites à la section précédente permettent d'isoler dans un processeur un sous-réseau quelconque. Cette forme de parallélisme se prête particulièrement bien à une implantation sur un ordinateur parallèle basé sur une architecture à mémoire distribuée, puisqu'un minimum d'information transite entre les processeurs (tensions et courants de part et d'autre du pont de découplage). Or, la seconde forme de parallélisme avec laquelle nous avons expérimenté se prête bien selon nous à une architecture à mémoire partagée et consiste à distribuer les calculs associés à un sous réseau sur plusieurs processeurs. Dans notre cas, ce sous réseau consiste en la partie onduleur/machine. Le travail décrit dans cette section a donc été réalisé sur l'ordinateur SGI Origin 2000 décrit à la section 4.2.

Afin d'étudier la performance de cette forme de parallélisme, nous avons décidé dès le départ de nous concentrer sur une petite partie de notre simulation. Nous avons donc mesuré individuellement le temps d'exécution des diverses tâches effectuées dans une itération, ou un pas de calcul, de notre simulation. Nous avons constaté que l'inversion de matrice requise pour la mise à jour des états selon la discrétisation trapézoïdale (4-1) occupait une part importante du temps de calcul à chaque pas.

$$x_{n+1} = \left(I - \frac{A_{n+1}T}{2}\right)^{-1} \left[\left(I + \frac{A_nT}{2}\right)x_n + \left(\frac{B_nT}{2}\right)u_n + \left(\frac{B_{n+1}T}{2}\right)u_{n+1} \right] \qquad (4\text{-}1)$$

Nous avons donc décidé de tenter la parallélisation de cette inversion de matrice sur deux processeurs afin de voir si un gain en temps de calcul est possible. D'abord, l'inversion de matrice est réalisée à l'aide d'une factorisation LU, suivie de deux substitutions (avant et arrière). Il existe d'autres algorithmes permettant d'inverser une matrice, mais notre choix s'est arrêté sur la factorisation LU parce qu'une part importante de l'algorithme, soit les substitutions avant et arrière, se parallélise de façon triviale tel que décrit à la section 3.3. La factorisation elle-même n'est pas facile à paralléliser et nous n'avons pas tenté de le faire. De toute façon, la factorisation elle-même est d'ordre de complexité mathématique $2N^3/3$ alors que l'inversion de matrice basée su la factorisation LU est de complexité $2N^3$. La substitution avant et arrière représentent donc ensemble deux fois plus de travail que la factorisation. Nous tenons à spécifier ici que l'ordre N de la matrice à inverser est typiquement inférieur à 15. Il est important de le mentionner puisque la plupart des algorithmes matriciels sont optimisés pour des matrices de grande dimension, par exemple d'ordre 500 et plus. La factorisation elle-même est donc réalisée sur un seul processeur, puis les substitutions avant et arrière sont chacune partagées sur deux processeurs, où chaque processeur traite la moitié des colonnes de X et B. La figure 4-8 illustre l'ordre des calculs ainsi que les mécanismes de synchronisation requis, dans ce cas-ci des barrières.

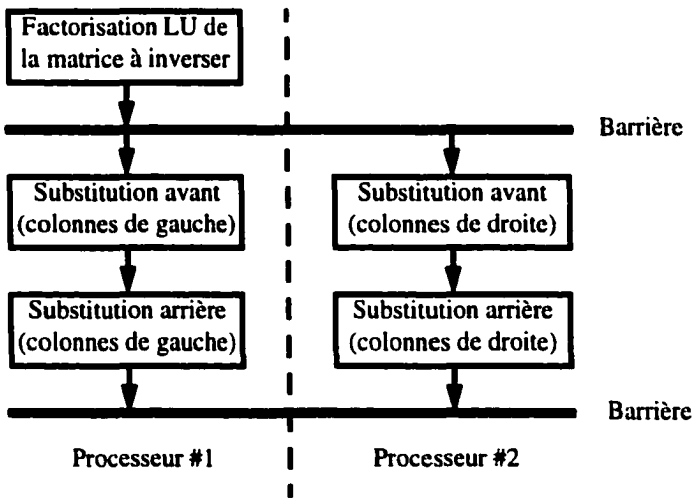


Figure 4-8 Inversion de matrice parallèle sur deux processeurs.

Notre expérimentation avec cette forme de parallélisme nous a permis de réaliser qu'aucun gain en temps de calcul ne peut être obtenu pour notre problème. Nous espérions gagner un peu de temps, mais le gain en temps dû au partage des calculs sur deux processeurs est dominé par l'incohérence de l'antémémoire ("cache misses"). En effet, la matrice à inverser se trouve dans la mémoire principale accessible à tous les processeurs. Lorsque le processeur #1 effectue la factorisation LU, il charge la matrice de la mémoire principale à son antémémoire, qui est beaucoup plus rapide. Lorsque ce même processeur commence à utiliser les facteurs L et U, ces matrices sont déjà chargées en antémémoire et ce processeur effectue sa part du calcul très rapidement. Cependant, pour l'autre processeur (le #2 dans la figure 4-8), les copies des matrices L et U qui sont dans son antémémoire ne sont plus valides, puisqu'elles datent du pas de calcul précédent et viennent d'être recalculées par le processeur #1. Des copies fraîches de ces deux matrices doivent donc être transférées à nouveau de la mémoire principale vers l'antémémoire du processeur #2. Ce transfert est très coûteux, la mémoire principale étant caractérisée par un temps d'accès de plusieurs fois supérieur à celui de l'antémémoire. Nous reviendrons sur ce point dans la section 4.5.1.

4.5 Principaux facteurs affectant le temps de calcul

Dans cette section, nous tenons à identifier les facteurs qui ont une influence directe sur le temps de calcul. Notre liste est relativement exhaustive, mais nous ne tenons qu'à identifier et décrire sommairement ces facteurs. Leur étude détaillée peut facilement faire l'objet d'un document assez volumineux. De plus, la plupart de ces facteurs étant liés au matériel, soit l'ordinateur lui-même, ils ne s'appliqueront pas de la même façon d'un type d'ordinateur à un autre. Bien que les algorithmes eux-mêmes soient un facteur important pour la vitesse des calculs, nous allons les traiter séparément dans une section ultérieure.

4.5.1 Matériel

Le matériel utilisé est d'importance capitale. Nous entendons par matériel les éléments suivants:

a. le microprocesseur;
b. l'antémémoire (mémoire "cache");
c. la mémoire principale;
d. les liens interprocesseur;
e. l'interface avec le monde extérieur (entrées et sorties du simulateur).

Le microprocesseur est le centre de tout ordinateur, incluant les simulateurs en temps réel. Nous sommes habitués de comparer les processeurs en termes de la vitesse de leur horloge, par exemple 800 MHz vs. 500 MHz, mais la vitesse de l'horloge n'est pas toujours un bon indicateur des performances du processeur. Certains processeurs 500 MHz sont plus performants que d'autres à 800 MHz. Afin de comparer les performances des processeurs entre eux dans un contexte donné, il faut analyser leur architecture interne, ce qui inclus entre autres:

a. le nombre de registres;
b. le nombre de niveaux du pipeline;
c. le nombre d'unités arithmétiques et logiques;
d. le jeu d'instructions;
e. la capacité d'effectuer certaines opérations complexes (multiplication et addition combinées, division, racine carrée) en un cycle d'horloge.

La vitesse de la mémoire principale est un autre facteur important. Nous ne mentionnons pas la quantité de cette mémoire puisqu'elle est amplement suffisante pour nos besoins dans les ordinateurs modernes, ce qui n'a pas toujours été le cas. Il faut être cons-

cient que même si un processeur est caractérisé par une horloge de 800 Mhz ou 1 GHz, il communique avec la mémoire via un bus qui possède une horloge typiquement de l'ordre de 100 MHz. Dans certaines applications dites intensives en accès à la mémoire, la vitesse de la mémoire principale est l'élément limitatif, et non pas celle du processeur.

La mémoire hiérarchique (plusieurs niveaux d'antémémoire) permet de réduire ce problème. Le principe de base de la mémoire hiérarchique est relativement simple. Il s'agit d'inclure sur la même puce que le microprocesseur, une petite quantité de mémoire très rapide, souvent de la même vitesse que l'horloge du processeur. La quantité de cette mémoire est réduite (quelques dizaines de ko) car elle est très dispendieuse. Le gestionnaire de cette mémoire y stocke les parties de programme et de données qui sont souvent accédées. Ces gestionnaires utilisent toutes sortes de règles pour décider de ce qui est souvent utilisé ou non. Ces détails dépassent le cadre de notre thèse. Cette première antémémoire est souvent dite de niveau un (L1). Il existe dans tous les processeurs modernes une quantité plus importante (quelques centaines de ko à plusieurs Mo) d'antémémoire de niveau deux (L2). Cette antémémoire est plus lente que l'antémémoire de niveau un parce qu'elle se trouve sur une puce externe à celle du microprocesseur et doit donc communiquer avec cette dernière via un bus. Les deux niveaux d'antémémoire sont illustrés à la figure 4-4.

Les accès en mémoire se font donc en général de la façon suivante. Si le processeur doit accéder à une donnée qui se trouve dans l'antémémoire L1, l'accès est très rapide. Si la donnée ne s'y trouve pas, l'antémémoire est interrogée pour voir si elle ne contient pas la donnée requise. Si c'est le cas, la donnée est transférée dans l'antémémoire L1. Si elle n'y est pas, elle doit être transférée de la mémoire principale vers l'antémémoire L2, puis vers l'antémémoire L1. On comprend donc que plus une donnée est loin du microprocesseur, plus elle est longue à accéder.

Sur une machine multiprocesseur à mémoire partagée, les accès à la mémoire peuvent être encore plus long à cause du processus d'invalidation de l'antémémoire. Ce phénomène se produit lorsque plusieurs processeurs possèdent dans leur antémémoire une copie de la même donnée et que l'un d'entre eux change la valeur de cette donnée. L'antémémoire des processeurs qui possèdent une copie de la donnée nouvellement changée est invalidée. Les processeurs en question sont ainsi avertis qu'ils doivent aller chercher une nouvelle copie de la donnée dans la mémoire principale s'ils en ont encore besoin. Ce processus peut occasionner une pénalité de plusieurs dizaines de milliers de cycles d'horloge [36]. Il faut donc utiliser l'antémémoire de façon réfléchie. Il existe plusieurs moyens de prendre avantage de l'antémémoire. L'une de ces méthodes consiste à prendre en compte la longueur des lignes d'antémémoire. Lorsqu'une donnée est transférée de la mémoire principale à l'antémémoire, ce n'est pas une seule donnée qui est transférée, mais une ligne de données. La longueur des lignes d'antémémoire varie d'un processeur à l'autre, mais elle est de 128 octets sur l'ordinateur SGI Origin 2000. Si les données sont en double précision (huit octets par donnée), une ligne contient 16 données différentes sur cet ordinateur. Il faut donc utiliser au maximum une ligne d'antémémoire lorsque celle-ci est chargée. Il est démontré dans [36] à l'aide d'un exemple qu'une multiplication matricielle multiprocesseur qui tient compte de la longueur des lignes d'antémémoire peut s'exécuter jusqu'à dix fois plus rapidement (pour un même nombre de processeurs) que la même multiplication programmée de façon "naïve".

Le dual de ces accès à la mémoire partagée se retrouve dans un ordinateur parallèle à mémoire distribuée sous la forme de liens de communication. Dans ce cas, nous avons vu plus haut que chaque processeur possède sa mémoire principale privée et inaccessible aux autre processeurs. Les données à partager avec d'autres processeurs doivent être transmises via des liens de communication. Le nombre de liens, le temps de démarrage d'une communication ainsi que le taux de transfert ont un impact majeur sur la performance dans ce type d'ordinateur parallèle.

Un dernier point important en ce qui concerne l'impact du matériel sur les performances d'un simulateur concerne les entrées et sorties. Rappelons que nous cherchons à faire de la simulation en temps réel afin de pouvoir interconnecter des équipements réels et un processus simulé. Or, l'interconnection du simulateur avec le monde externe passe par des convertisseurs analogique à numérique et numérique à analogique. De plus, ces convertisseurs étant externes au processeur, des liens de communication les relient. La bande passante de ces liens ainsi que le délai de conversion auront un impact sur la performance du simulateur.

4.5.2 Système d'exploitation

Nous tenions à aborder ce sujet parce nous entendons de plus en plus parler de systèmes d'exploitation en temps réel. Il faut bien comprendre de quoi il s'agit. Lorsque l'on parle de temps réel, il faut spécifier l'ordre de grandeur du pas de calcul utilisé si l'on veut être en mesure d'analyser la possibilité d'utiliser tel type de matériel ou tel type de logiciel. Par exemple, nous avons rencontré lors d'une conférence un chercheur qui développait un simulateur servant à former les opérateurs de réseaux ferroviaires. Son simulateur est en temps réel mais utilise un pas de calcul de deux secondes, puisqu'il est suffisant de connaître la position d'un train sur un réseau ferroviaire à toutes les deux secondes. Les systèmes d'exploitation en temps réel ont des caractéristiques intéressantes mais sont tout à fait inutilisables pour notre type d'application. Le temps que met le système d'exploitation à répondre à une requête de service est de l'ordre de la milliseconde. Si l'on compare ce temps à une simulation qui utilise un pas de calcul de 50 µs, qui est typique dans le domaine des réseaux électriques, on se rend compte que nous avons effectué plusieurs dizaines de pas de calcul alors que le soi-disant système d'exploitation en temps réel commence à peine à réagir.

Dans la version SGI du simulateur Hypersim, on charge le système d'exploitation dans un seul processeur maître. Les autres processeurs ne contiennent pas de système

d'exploitation mais sont contrôlés par le processeur maître, lequel ne peut servir à la simulation. Il n'est pas possible de faire de la simulation en temps réel à l'échelle des dizaines de microsecondes autrement.

Dans la version DEC Alpha du simulateur, aucun des processeurs ne contient de système d'exploitation. Les programmes sont chargés dans les processeurs par l'entremise d'un poste de travail lié à chacun des processeurs via une interface de communication IEEE 1394 ("Fire Wire").

4.5.3 Compilateur

Lorsque l'on écrit un programme dans un langage de haut niveau comme le langage C, ce programme doit être interprété et traduit dans un langage que le processeur comprend, le langage assemblé, avant de pouvoir être exécuté. Cette traduction est faite par le compilateur. Or, il y a souvent plusieurs façons de réaliser cette traduction et différents compilateurs vont produire des programmes avec des performances différentes à partir d'un même fichier source. De plus, un compilateur fournit à l'usager la possibilité de spécifier de nombreuses options de compilation qui affectent grandement la performance du programme. Une bonne compréhension de ces options est requise lorsque l'on recherche le niveau de performance requis dans notre application. Ceci implique que nous devons également bien comprendre l'architecture du processeur utilisé, plusieurs des options de compilation ayant un lien direct avec l'architecture du processeur. Afin de bien comprendre les options de compilation, une bonne documentation du compilateur est requise.

L'optimisation du code est une option que l'on retrouve dans la plupart des compilateurs modernes. Dans un environnement UNIX, on le spécifie la plupart du temps en indiquant l'option *-On*, où *n* est généralement un petit chiffre entier indiquant le niveau d'optimisation. Le niveau zéro indique qu'aucune optimisation ne doit être effectuée. Le niveau maximum d'optimisation varie d'un compilateur à l'autre, mais est généralement

trois ou quatre. Plus le niveau est élevé et plus l'optimisation est agressive. Il faut cependant utiliser cette option avec soin puisque d'une part, le niveau d'optimisation le plus élevé ne produit pas nécessairement le programme le plus performant, et d'autre part les niveaux élevés d'optimisation peuvent produire un programme qui donne des résultats imprécis. Ces imprécisions proviennent du fait qu'à ces niveaux, le compilateur peut décider de changer l'ordre de certaines opérations en plus d'utiliser des fonctions (par exemple la racine carrée) optimisées pour la vitesse de calcul, mais non conforme à la norme IEEE 754 pour la représentation des nombres en virgule flottante. Il faut donc expérimenter avec les niveaux d'optimisation afin de déterminer quel niveau est le plus approprié pour une application donnée. Ceci peut impliquer la séparation du programme en plusieurs fichiers dans le but de permettre la compilation de chacun avec un niveau d'optimisation bien adapté aux tâches effectuées.

Les compilateurs effectuent des optimisations à divers niveaux. Le but est cependant toujours le même, soit d'utiliser le plus efficacement possible les ressources de l'ordinateur. Nous présentons brièvement trois aspects d'optimisation qui ont eu un impact appréciable sur le temps d'exécution de nos programmes sur l'ordinateur SGI, soit [35]:

a. "software pipelining";
b. déroulement de boucles;
c. restriction des alias de pointeurs.

Le "software pipelining" consiste à tirer avantage du fait que le processeur, doté d'un pipeline à plusieurs niveaux, est capable d'exécuter plusieurs instructions à chaque cycle d'horloge. Il s'agit en fait de dérouler la boucle la plus interne dans des boucles imbriquées ou de dérouler la boucle tout simplement lorsqu'il n'y pas de boucles imbriquées. Dans le cas du SGI, le pipeline à quatre niveaux permet l'exécution simultanée de

quatre instructions. Ces instructions peuvent être toute combinaison parmi la liste suivante:

a. un accès mémoire, en lecture ou en écriture;
b. une instruction de l'unité arithmétique et logique (UAL) #1;
c. une instruction de l'unité arithmétique et logique (UAL) #2;
d. une addition en virgule flottante;
e. une multiplication en virgule flottante.

Les UAL servent essentiellement au calcul des adresses (chiffres entiers) des données. La liste ci-haut nous permet de déduire que sur cet ordinateur, il est théoriquement possible d'exécuter deux opérations en virgule flottant (une addition et une multiplication) à chaque cycle. Or ce maximum théorique est difficilement réalisable, comme le démontre l'exemple suivant. Le but ici est simplement de présenter le principe, une analyse détaillée de cet exemple est disponible dans [35]. Soit une boucle sous forme de pseudo-code tel que montré à la figure 4-9.

```
for i=1:n
   y(i) = y(i) + a*x(i);
end
```

Figure 4-9 Une boucle simple.

Chaque itération de cette boucle implique:

a. trois accès en mémoire, soit la lecture de `x(i)` et `y(i)` puis l'écriture de `y(i)`;
b. une multiplication et une addition;
c. deux incrémentations d'adresse, `x(i)` et `y(i)`;

d. un test de fin de boucle;

e. un branchement.

Une fois la boucle en régime permanent, soit lorsque le pipeline est rempli, l'exécution de la ligne dans la boucle requiert trois accès mémoire pour deux opérations en virgule flottante. Ceci implique que pour cette boucle, le rapport maximum d'opérations par cycle est 2/3, ce qui est le tiers du maximum théorique de deux. Or, si la boucle est programmée telle quelle, ce rapport sera plutôt de 2/7, à cause des contraintes matérielles, ce qui est 1/7 du maximum théorique de deux opérations par cycle. Déroulons la boucle tel qu'indiqué à la figure 4-10 (la deuxième boucle `for` est requise afin de réaliser la dernière itération dans le cas où n est impair).

```
for i=1:2:n-1
   y(i+0) = y(i+0) + a*x(i+0);
   y(i+1) = y(i+1) + a*x(i+1);
end
for i=i:n
   y(i+0) = y(i+0) + a*x(i+0);
end
```

Figure 4-10 Une boucle simple déroulée une fois.

Dans ces conditions, le processeur arrive à réaliser quatre opérations en virgule flottante en huit cycles, ce qui donne 1/4 du maximum théorique et est une nette amélioration. En déroulant la boucle quatre fois plutôt que deux, on arrive à atteindre le maximum pour ce problème, qui est de deux opérations par trois cycles et représente 1/3 de la capacité maximale du processeur.

Le but est donc le suivant. Il s'agit que le compilateur détermine automatiquement le niveau optimal de déroulement des boucles interne ("software pipelining") afin de

maximiser le taux d'opérations en virgule flottante par cycle, sans avoir à faire ce travail manuellement (dans notre programme). L'efficacité de ce type d'optimisation dépend en grande partie de l'architecture interne du processeur (niveaux de pipeline, nombre d'UAL) et de la taille du problème. Plus une boucle comporte d'itérations, plus son optimisation rapportera. Dans le cas où une boucle comporte peu d'itérations, il vaut mieux ne pas utiliser cette forme d'optimisation, puisque la performance peut être réduite à cause du code additionnel requis pour remplir le pipeline et terminer les dernières itérations (deuxième boucle `for` dans l'exemple ci-haut).

Un autre concept intimement lié au "software pipelining" est le déroulement des boucles externes lorsque plusieurs boucles sont imbriquées. Cette forme d'optimisation est nommée "loop-nest optimisation" (LNO) dans la littérature SGI. Un exemple détaillé est décrit dans [35] où l'on démontre qu'avec ce type d'optimisation, il est possible, pour un algorithme de multiplication matricielle, de passer de 33% à 100% d'utilisation du processeur en déroulant judicieusement les deux boucles externes, l'algorithme comportant trois boucles imbriquées. Comme le concept est très similaire à celui du "software pipelining", nous ne présentons pas l'exemple ici. Nous tenons simplement à mentionner que nous avons utilisé cette forme d'optimisation lors de la compilation de nos programmes.

Une dernière forme d'optimisation a eu un impact favorable sur la performance de nos simulations. Cette optimisation consiste simplement à donner une information additionnelle au compilateur concernant les pointeurs utilisés dans nos programmes. Dans un programme, il est possible de déclarer des pointeurs vers différents espaces mémoire. Cependant, rien n'empêche que deux pointeurs différents pointent vers la même adresse en mémoire. Le compilateur assume le pire cas, c'est-à-dire que des pointeurs avec des noms différents vont tôt ou tard pointer vers la même adresse. Ce phénomène est baptisé "aliasing" dans la littérature SGI et empêche le compilateur d'effectuer certaines optimisations, surtout en présence de boucles. Or, nos programmes contiennent beaucoup de

boucles. Par contre, tous nos pointeurs pointent vers des zones spécifiques en mémoire, de telle sorte qu'aucun pointeur ne devient l'alias d'un autre. Nous pouvons avertir le compilateur que c'est le cas à l'aide d'une option de compilation. Lorsque cette option est utilisée, c'est cependant la responsabilité du programmeur de s'assurer que les pointeurs pointent bien vers des plages distinctes en mémoire.

4.5.4 Outils d'analyse

La mention de ce sujet dans notre thèse nous tient à coeur puisqu'ayant expérimenté avec deux types d'ordinateurs parallèles, nous avons été en mesure d'apprécier la valeur de bons outils d'analyse. Nous avons d'abord tenté de réaliser une simulation en temps réel sur le simulateur Hypersim de l'É.T.S., qui est la version à mémoire distribuée basée sur le processeur DEC Alpha. Nous ne disposons sur cet ordinateur d'aucun outil nous permettant d'analyser les performances de nos programmes. Toute forme d'optimisation consiste alors à apporter une modification au code et vérifier si le temps de simulation est amélioré ou non. Après un certain temps, nous avons eu l'occasion d'essayer nos programmes sur le simulateur Hypersim de l'IREQ, basé sur la technologie SGI. La compagnie SGI fournit avec ses ordinateurs une quantité impressionnante d'outils de développement et de documentation. De plus, ces ordinateurs sont dotés de matériel interne sophistiqué permettant de faire une analyse détaillée du comportement d'un programme. La combinaison de ces facteurs nous a permis d'apporter des modifications importantes qui ont contribué à améliorer la performance de nos programmes sur cet ordinateur. Nous nous sommes aperçus à plusieurs reprises que les parties plus lentes de notre programme ne se trouvaient pas aux endroits où l'on s'attendait. Enfin, c'est grâce à ces outils que nous avons pu constater hors de tout doute que notre tentative de parallélisation décrite à la section 4.4 n'apportait aucun gain en termes de temps de calcul.

4.5.5 Style de programmation

Notre but étant de réaliser de la simulation en temps réel, nous nous sommes aperçus très tôt dans nos travaux qu'il fallait laisser de côté toutes les bonnes pratiques de programmation qu'on nous enseigne dans les cours de programmation de base. Par exemple, lorsqu'une fonction dépasse tant de lignes de code, il faut en créer une autre car la fonction devient trop complexe, ou encore il faut minimiser la quantité de variables globales et utiliser le plus possible les variables locales. Lorsque l'on veut résoudre des équations différentielles, interfacer notre simulation avec de l'équipement externe, communiquer entre plusieurs processeurs, le tout en quelque dizaines de microsecondes, ces bonnes pratiques ne sont tout simplement pas applicables. D'abord, à chaque fois qu'une fonction est appelée dans un programme, un changement de contexte survient. Le processeur doit d'abord sauvegarder l'état de tous les registres sur la pile. Ensuite, les arguments de la fonction sont interprétés et passés à la fonction, puis la fonction elle-même s'exécute et son argument de sortie retourné au programme appelant. Enfin, le processeur doit récupérer les registres sur la pile et les remettre dans le même état que celui dans lequel ils étaient avant l'appel de la fonction. Ces changements de contexte peuvent facilement prendre quelques microsecondes chacun. Si dans notre boucle de simulation plusieurs fonctions sont appelées de la sorte, il ne reste pas beaucoup de temps pour faire les calculs.

Pour ce qui est de l'utilisation restreinte de variables globales, il faut en fait faire exactement l'inverse lorsque la vitesse de calcul est importante. Il faut minimiser le nombre de variables, et le moyen le plus simple de faire ceci est d'utiliser les variables globales. Nous avons étudié pendant un certain temps le générateur de code "Real-Time Workshop", un produit de la compagnie Mathworks qui permet de générer du code C à partir d'un schéma Simulink. Il s'avère que toutes les données liées à une simulation sont contenues dans trois gros vecteurs:

a. un vecteur pour les données entières;
b. un vecteur pour les données en point flottant (dans ce cas-ci en double précision);
c. un vecteur pour tous les pointeurs.

Il n'y a donc dans le code généré par le Real-Time Workshop que trois variables pour toutes les données liées à la simulation, et elles sont globales. Une approche semblable est utilisée dans Hypersim. Une analyse rapide du code généré par Hypersim nous a permis de constater que tous les paramètres liés à une simulation sont contenus dans quelques gros vecteurs.

Nous n'avons pas été jusqu'à réduire le nombre de variables dans nos programmes car cette approche rend le code assez difficile à lire et par conséquent, à modifier ou déverminer. Notre programme principal, la s-function qui implante la simulation de la partie onduleur et machine, utilise une trentaine de vecteurs différents.

Un autre point qui nous semblait anodin au départ, s'est avéré très important. Il s'agit de la façon dont sont stockées en mémoire les différentes données, particulièrement les matrices. Nous avions adopté au départ la méthode décrite dans [37] qui consiste à utiliser des double pointeurs. Il s'avère qu'à moins de prendre des précautions particulières, cette façon de procéder a pour effet qu'une même matrice n'est pas nécessairement dans un espace contigu en mémoire. Les accès en mémoire ayant un impact majeur sur la performance d'un programme, cette alternative à été mise de côté. Nous stockons plutôt nos matrices dans des vecteurs en plaçant bout à bout les rangées ou les colonnes de la matrice originale. Le choix optimal entre stocker les rangées ou stocker les colonnes dépend des algorithmes utilisés [33]. Dans notre cas, la factorisation LU et la solution à partir des facteurs L et U nous ont encouragé à adopter le stockage par colonnes. Nous avons également expérimenté avec le stockage par rangées et n'avons pas noté de différence appréciable dans les temps de calcul.

4.6 Optimisation des calculs

Nous avons présenté à la section précédente plusieurs facteurs qui affectent le temps de calcul associé à un programme, mais n'avons pas abordé la question de l'optimisation des algorithmes, ce que nous allons faire maintenant. Nous identifions d'abord les points auxquels nous devons porter une attention particulière lorsque nous réalisons des algorithmes matriciels, après quoi nous décrivons quelques optimisations spécifiques apportées à nos programmes.

4.6.1 Algorithmes matriciels

Un facteur sur lequel un algorithme a un impact direct est le patron d'accès aux données en mémoire. Nous avons vu à la section 4.5 que l'antémémoire est accédée non pas une donnée à la fois, mais une ligne à la fois. Nous n'avons pas la possibilité de spécifier explicitement comment l'antémémoire stocke et récupère les données, mais nous pouvons le faire indirectement via nos algorithmes. Considérons par exemple la multiplication d'une matrice A, de dimension m par p, par une matrice B, de dimension p par n. Le résultat est stocké dans une matrice C, de dimension m par n. L'algorithme de la figure 4-11 effectue ce calcul.

```
for i = 1:m
    for j=1:n
        for k=1:p
            C(i,j) = A(i,k) * B(k,j)
        end
    end
end
```

Figure 4-11 Algorithme de multiplication matricielle, version *ijk*.

Cet algorithme porte souvent l'appellation de variante *ijk*. Cette appellation provient de l'ordre des trois boucles `for`. Les indices *i* et *j* sont couramment utilisés pour parcourir les rangées et les colonnes du résultat, respectivement, tandis que la variable *k* sert pour la troisième boucle. L'algorithme de la figure 4-11 comporte six variantes (*ijk*, *ikj*, *jik*, *jki*, *kij*, *kji*) qui s'obtiennent simplement en changeant l'ordre des trois boucles `for`. Bien que les six variantes représentent le même travail en termes du nombre d'opérations mathématiques, les accès aux différentes données se font selon des patrons variés qui ne donneront pas tous les mêmes performances. Le tableau 4-1 nous permet de constater comment les matrices sont accédées pour chaque variante.

Tableau 4-1

Patrons d'accès aux données pour les variantes de la multiplication matricielle

Variante	Accès aux données dans la boucle interne
ijk	*A* par rangées, *B* par colonnes
jik	*A* par rangées, *B* par colonnes
ikj	*B* par rangées, *C* par rangées
jki	*A* par colonnes, *C* par colonnes
kij	*B* par rangées, *C* par rangées
kji	*A* par colonnes, *C* par colonnes

Ceci dit, une fois choisi la méthode de stockage des matrices (par rangées ou par colonnes), il faut porter une attention particulière à l'ordonnancement des boucles dans nos algorithmes matriciels.

4.6.2 Mise à jour des matrices d'état suite à une commutation

La méthode de mise à jour des matrices d'état de la section 2.1.2 a été implantée dans le Power System Blockset version 2, mais d'une façon légèrement différente de celle présentée à la section 2.1.2. Au lieu de faire l'inversion de matrice (2-8), il s'agit de modifier les matrices en prenant en compte un changement d'interrupteur à la fois. Par exemple, si deux interrupteurs changent d'état durant le même pas de calcul, on change d'abord l'état du premier, on obtient les matrices d'état correspondant au nouvel état de cet interrupteur, puis on change l'état du second interrupteur et on remet à jour les matrices d'état. Cette façon de faire permet d'éviter d'inverser une matrice et elle est plus efficace en termes de temps de calcul, lorsqu'un nombre restreint d'interrupteurs changent d'état durant un même pas de calcul. Cette situation se produira le plus souvent lorsque les interrupteurs sont de type à commutation naturelle (diode, thyristor). Par exemple, dans un redresseur à diodes en pont triphasé, il est rare que plus de deux ou trois diodes changent d'état au même moment.

Cependant, dans le cas d'interrupteurs à commutation forcée (GTO, IGBT, MOSFET), il y a des moments où tous les interrupteurs d'un convertisseur changent d'état au même moment. Dans ces conditions, la méthode de mise à jour "un interrupteur à la fois" n'est pas la plus efficace. De plus, la matrice qu'il faut inverser dans (2-8) comporte des propriétés intéressantes qui peuvent être exploitées afin de réduire la quantité de calculs requise.

On remarque d'abord que le terme de droite de (2-8) consiste à post-multiplier la matrice D_o par la matrice D_{sw}. Or, cette dernière est une matrice diagonale contenant des valeurs non-nulles uniquement dans les colonnes correspondant à des interrupteurs fermés, si l'on considère des interrupteurs avec une résistance infinie à l'état bloqué. En post-multipliant la matrice D_o par cette matrice diagonale, le résultat est une matrice comportant des éléments non-nuls seulement dans les colonnes correspondant à des inter-

rupteurs fermés. On soustrait ensuite ce résultat de la matrice identité. Par exemple, pour un système comportant quatre interrupteurs et huit sorties, avec le premier et le quatrième interrupteur fermé, la matrice $(I - D_o D_{sw})$ a la forme montrée à la figure 4-12.

$$\begin{bmatrix} X & & X & & \\ X & 1 & X & & \\ X & & 1 & X & \\ X & & & X & \\ X & & & X & 1 \\ X & & & X & & 1 \\ X & & & X & & & 1 \\ X & & & X & & & & 1 \end{bmatrix}$$

Figure 4-12 Forme de la matrice $(I - D_o D_{sw})$.

La forme de la matrice de la figure 4-12 nous intéresse parce que nous nous sommes aperçu que cette matrice est diagonale dominante pour notre système, c'est-à-dire que pour chaque colonne, aucun élément n'est supérieur en valeur absolue à l'élément de la diagonale. Cette condition fait que si l'on effectue une factorisation LU sur cette matrice, il n'est pas nécessaire d'effectuer de permutation des rangées en vue d'obtenir un pivot adéquat. Sachant que nous n'avons pas a permuter de rangées, la factorisation elle-même peut se faire de façon très efficace parce que nous savons dans quelles colonnes se trouvent tous les éléments non-nuls. De plus, le fait de ne pas avoir à permuter de rangées fait que les facteurs L et U obtenus par la factorisation auront l'allure montrée à la figure 4-13.

Encore une fois, nous savons dans quelles colonnes se trouvent les éléments non-nuls et la solution des équations (2-9) et (2-10) à partir des facteurs L et U de la figure 4-13 peut se faire de façon très efficace.

$$L = \begin{bmatrix} 1 & & & & & & & \\ X & 1 & & & & & & \\ X & & 1 & & & & & \\ X & & & 1 & & & & \\ X & & & X & 1 & & & \\ X & & & X & & 1 & & \\ X & & & X & & & 1 & \\ X & & & X & & & & 1 \end{bmatrix} \qquad U = \begin{bmatrix} X & & & X & & & & \\ & 1 & & X & & & & \\ & & 1 & X & & & & \\ & & & X & & & & \\ & & & & 1 & & & \\ & & & & & 1 & & \\ & & & & & & 1 & \\ & & & & & & & 1 \end{bmatrix}$$

Figure 4-13 Forme des facteurs L et U de l'exemple.

4.7 Étude de cas

Cette section vise à présenter les résultats que nous avons obtenus lors de l'implantation en temps réel des techniques de modélisation et d'intégration des chapitres précédents. Ces résultats sont présentés dans l'ordre chronologique dans lequel nous les avons obtenus. Des références sont faites aux sections 4.5 et 4.6 afin de quantifier les gains en temps de calcul réalisés à l'aide des différentes formes d'optimisation.

Rappelons que le système simulé est celui décrit à la section 4.1. Le système comporte une source triphasée avec une impédance interne calculée de telle sorte que la puissance de court-circuit de la source est 10 fois supérieure à la puissance de la machine et que le facteur de qualité des réactances est de 10. Cette source alimente un redresseur à diodes en pont complet. Chaque diode est munie d'un circuit d'amortissement consistant en une résistance en série avec un condensateur. Le redresseur alimente à son tour un bus à courant continu (CC) comprenant un condensateur en parallèle avec une résistance élevée, qui sert à décharger le condensateur lorsque le système est désalimenté. Un système de freinage dynamique, servant d'une part à freiner la machine et d'autre part à maintenir la tension du bus CC, est également branché sur ce bus. Le bus CC alimente enfin un onduleur triphasé qui est commandé par une commande DFTC. La partie électrique du

système est scindée en deux en utilisant la dernière technique de découplage décrite à la section 4.3. La commande DFTC est simulée sur un troisième processeur. La figure 4-14 illustre les trois parties du système sous forme de schémas Simulink/PSB. Le tableau 4-2 résume les principaux paramètres de chaque partie du système. Enfin, la figure 4-15 illustre le schéma Hypersim utilisé pour la simulation en temps réel.

Tableau 4-2

Principaux paramètres de chaque partie de la simulation

Partie	Nombre d'interrupteurs	Nombre d'états	Nombre d'entrées	Liste des entrées	Nombre de sorties	Liste des sorties
Redresseur	7	8	4	3 tensions de la source, tension du bus CC	15	7 tensions et 7 courants d'interrupteurs, courant du bus CC
Onduleur et machine	6	11	7	courant du bus CC, 6 impulsions pour l'onduleur	18	6 tensions et 6 courants d'interrupteurs, 2 tensions et 3 courants de machine, tension du bus CC
Commande DFTC	----	3	6	2 tensions, 3 courants et vitesse de la machine	6	6 impulsions pour l'onduleur

Nos premières simulations sur un ordinateur parallèle ont été réalisées sur le simulateur Hypersim de l'É.T.S., basé sur la technologie DEC Alpha 533 MHz. Le tableau 4-3 indique nos meilleurs temps de calculs pour la partie onduleur et machine. Notons que ces temps incluent seulement le calcul de nos s-function et n'incluent pas les temps de communication interprocesseur ni les temps d'acquisitions des entrées et sorties. Les temps indiqués au tableau 4-3 sont pour un système comportant 11 variables d'état, dont six sont les tensions des condensateurs des circuits d'amortissement. Nous avons expérimenté avec ce montage en incluant ou non des condensateurs aux bornes des interrupteurs en guise de circuit d'amortissement. Selon nos constatations, il n'y a pas de circuit

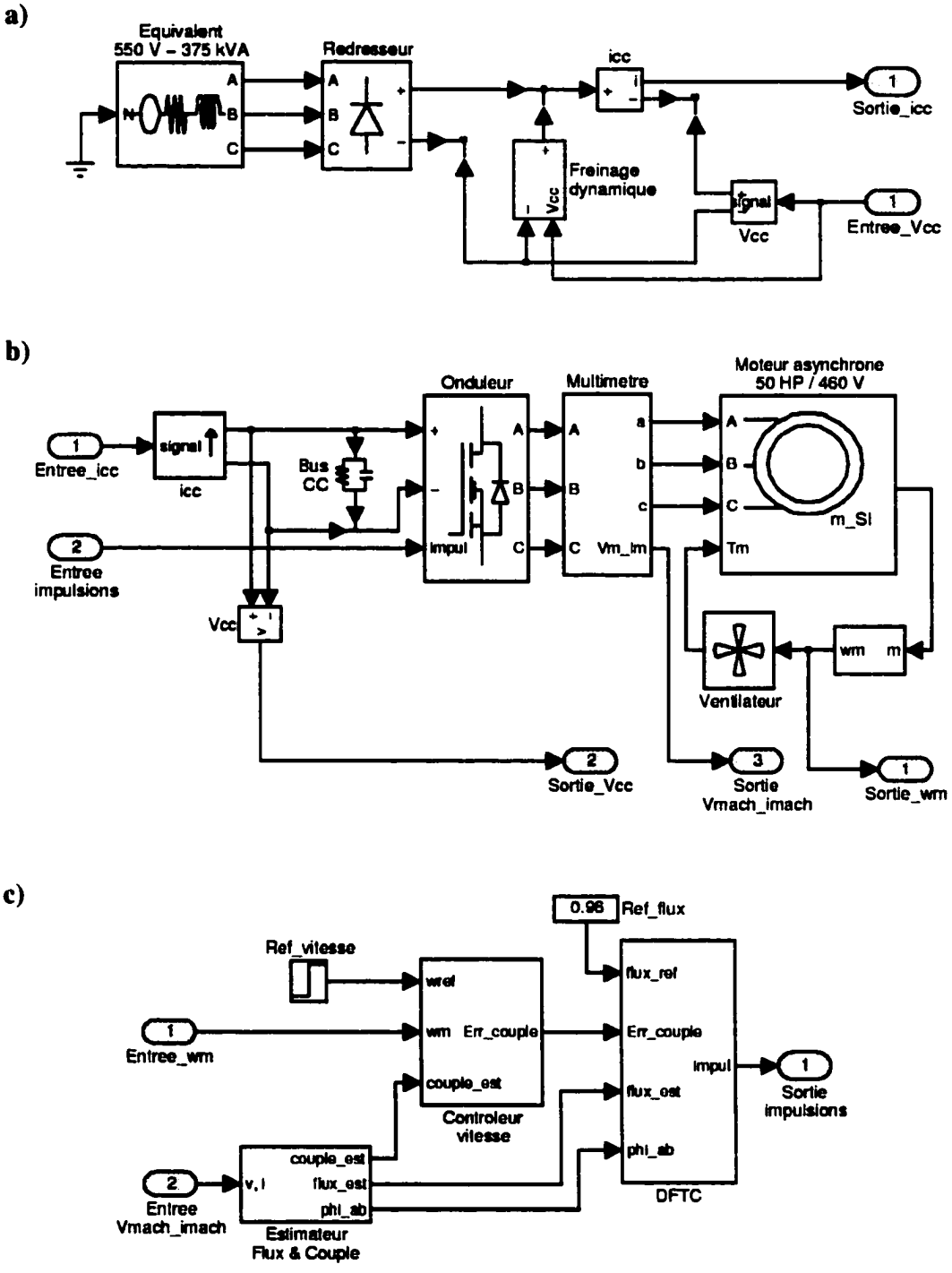


Figure 4-14 Schéma Simulink / PSB des trois parties du système: **a)** source et redresseur; **b)** onduleur, machine et charge; **c)** commande DFTC.

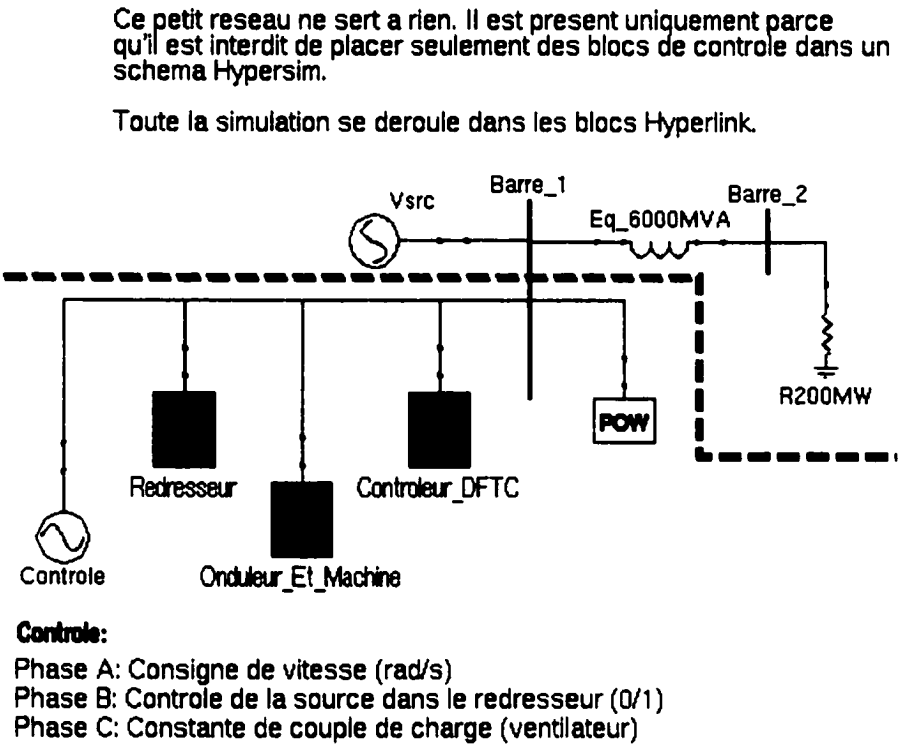


Figure 4-15 Schéma Hypersim utilisé pour la simulation en temps réel.

d'amortissement aux bornes des interrupteurs de l'onduleur dans les entraînements industriels de moyenne et grande puissance. Nous avons tout de même continué à tester nos algorithmes avec ces circuits d'amortissement puisqu'ils augmentent le nombre de variables d'état et permettent de mettre en évidence les endroits dans nos programmes qui consomment le plus de temps de calcul.

Le temps de simulation maximum total de la partie onduleur et machine sans circuit d'amortissement aux bornes des interrupteurs de l'onduleur est de 17 µs. Cette version du système est d'ailleurs celle qui représente fidèlement l'entraînement ABB décrit au début de ce chapitre.

Tableau 4-3

Temps d'exécution de la partie onduleur/machine avec 11 variables d'état

(DEC Alpha 533 MHz)

Numéro	Description de la tâche	Temps d'exécution (µs)	
		minimum	maximum
1	Vérification logique des interrupteurs	1.1	1.4
2	Mise à jour des matrices d'état suite à un changement d'état	0.1	34.3
3	Mise à jour des matrices liées à la machine et construction de la matrice A finale	0.5	6.6
4	Décomposition LU de $\left(I - \frac{AT}{2}\right)$	13.5	14.0
5	Mise à jour des états discrets, incluant solution LU d'un système d'équations	3.1	3.7
6	Calcul des sorties	2.8	4.0
Total maximal obtenu:			64

Jusqu'à présent, nous n'avons fait aucune mention du temps de simulation des deux autres parties, soit celle correspondant à la commande et celle correspondant à la partie redresseur du montage. D'abord, le temps de calcul associé à la commande DFTC ne nous intéresse pas beaucoup puisque notre but final est de permettre à un usager d'interconnecter la véritable commande à une simulation en temps réel de l'étage de puissance complet du système. De plus, les calculs associés à la commande sont insignifiants comparativement aux deux autres morceaux du système et se font en moins de cinq µs. Pour ce qui est de la partie source et redresseur, nous ne nous y intéressions pas au départ puisque la simulation de la partie onduleur et machine comportait un temps de calcul beaucoup plus long. Cependant, suite à nos optimisations et à l'élimination des condensateurs

d'amortissement du côté onduleur, c'est dorénavant la partie redresseur qui prend le plus de temps à calculer, tel qu'indiqué au tableau 4-4.

Tableau 4-4

Temps de calcul sur DEC Alpha 533 MHz de chacun des trois sous-systèmes de l'entraînement avec cinq variables d'état du côté onduleur/machine

Processeur	Description	Temps de calcul maximum (µs)
1	Source triphasée, redresseur à diodes et freinage dynamique	30
2	Bus CC, onduleur, machine asynchrone et charge de type ventilateur	17
3	Commande DFTC	< 5.0

Notons que nos travaux de recherche ont porté essentiellement sur la partie du système qui comprend l'onduleur et la machine et nous n'avons malheureusement pas eu le temps de tenter d'optimiser la partie redresseur, qui est simulée de façon tout à fait identique à celle utilisée dans le PSB version 2.1. Nous sommes cependant convaincus qu'il y a moyen de réduire le temps de calcul de la partie redresseur.

Afin de déterminer le pas de calcul minimum avec lequel le système peut être simulé en temps réel, il faut additionner au temps de calcul maximal du tableau 4-4 le temps requis pour les communications interprocesseurs ainsi que le temps de conversion d'éventuelles entrées et sorties et leur envoi aux processeurs via un autre lien de communication. Ces temps totalisent environ 25 µs sur le simulateur Hypersim basé sur la technologie DEC Alpha. En additionnant ces 25 µs aux 30 µs de temps de calcul du redresseur (la tâche la plus longue du tableau 4-4), on obtient un pas de calcul minimum utilisable en temps réel de 55 µs. En se référant au tableau 2-2 où nous avons établi les pas de calculs en fonction de la fréquence de commutation maximale de l'onduleur, on

constate qu'un pas de calcul de 62.5 µs pourrait être utilisé. Ce pas correspond à une fréquence de 16 kHz, ce qui est quatre fois plus que la fréquence de commutation maximale imposée à l'onduleur, soit 4 kHz.

Nous avons également expérimenté avec ces simulations sur l'ordinateur SGI Origin 2000. Cependant, la comparaison a été réalisée avec une version intermédiaires de nos programmes. Les tâches ne sont pas tout à fait les mêmes que celles montrées au tableau 4-4. Les résultats obtenus avec l'ordinateur SGI Origin 2000 sont montrés au tableau 4-5.

Tableau 4-5

Temps d'exécution des différentes tâches avec 11 variables d'état

(DEC Alpha 533 MHz et SGI Origin 2000 400 MHz)

Numéro	Description de la tâche	Temps d'exécution (µs)	
		SGI Origin 2000 à 400 MHz	DEC Alpha à 533 MHz
1	Mise à jour des états discrets	2.4	3.7
2	Vérification logique des interrupteurs	1.6	1.4
3	Mise à jour des matrices d'état suite à un changement d'état	20.0	34.3
4	Calcul des sorties	2.4	4.0
5	Mise à jour des matrices liées à la machine et construction de la matrice A finale	7.2	10.7
6	Inversion de $\left(I - \frac{AT}{2}\right)$	25.6	41.9
Total maximal:		53	93

Le temps de simulation maximum total obtenu sur le SGI est de 53 µs contre 93 µs pour le DEC Alpha. Ceci nous a d'abord surpris puisque les processeurs du SGI ont une horloge de 400 MHz alors que les processeurs DEC Alpha ont une horloge de 533 MHz. De plus, les processeurs Alpha sont réputés comme étant parmi les plus performants pour faire du calcul en virgule flottante. Ce résultat sert cependant à démontrer notre affirmation de la section 4.5.1, comme quoi il ne faut pas simplement comparer les horloges de deux processeurs afin d'évaluer leurs performances pour une application donnée. Nos collaborateurs à l'IREQ nous ont mentionné qu'ils ont noté des différences appréciables entre les temps de calculs obtenus sur les deux plate-formes, mais le SGI n'est pas toujours la solution la plus avantageuse. Cet ordinateur est cependant clairement supérieur pour notre application.

Pour ce qui est des résultats qualitatifs (formes d'ondes), ils sont tout à fait comparables à ce que nous avons obtenu en temps différé pour ce montage. Nous avons soumis le système à un changement de référence de vitesse de la vitesse maximale positive à la vitesse maximale négative. La figure 4-16 illustre le courant statorique de la phase *A*, la partie du bas étant un agrandissement de la fin de la partie du haut. La figure 4-17 illustre la réponse à un échelon de vitesse ainsi que la tension du bus CC. Cette dernière courbe nous permet de constater que la tension du bus CC fluctue beaucoup. Enfin, nous avons tracé à la figure 4-18 les enveloppes des temps d'exécution des deux principales parties de notre simulation. On peut constater que la partie onduleur et machine s'exécute plus rapidement que la partie source et redresseur, conformément au tableau 4-4, et que le temps d'exécution de cette dernière fluctue selon le nombre d'interrupteurs qui changent d'état, ce qui n'est pas le cas dans la partie onduleur machine. La raison pour laquelle le temps de simulation de la partie redresseur est sensible au nombre d'interrupteurs qui changent d'état est que cette partie utilise la méthode de mise à jour des matrices d'état "un interrupteur à la fois" décrite à la section 4.6.2. La partie onduleur et machine utilise

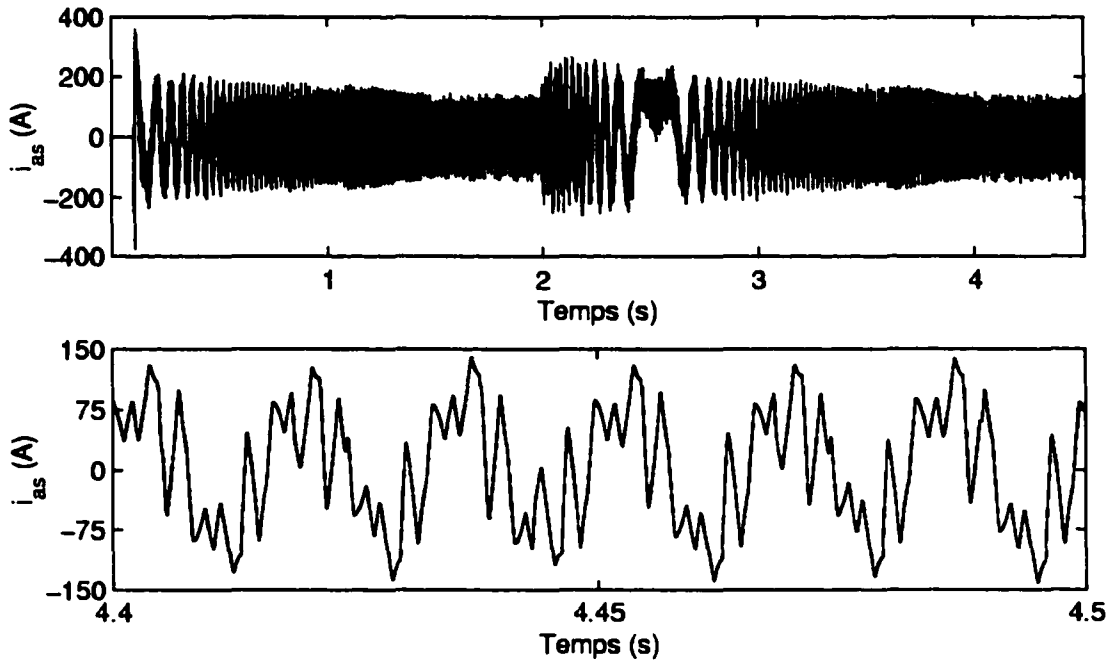


Figure 4-16 Courant statorique de la machine, phase *a*.

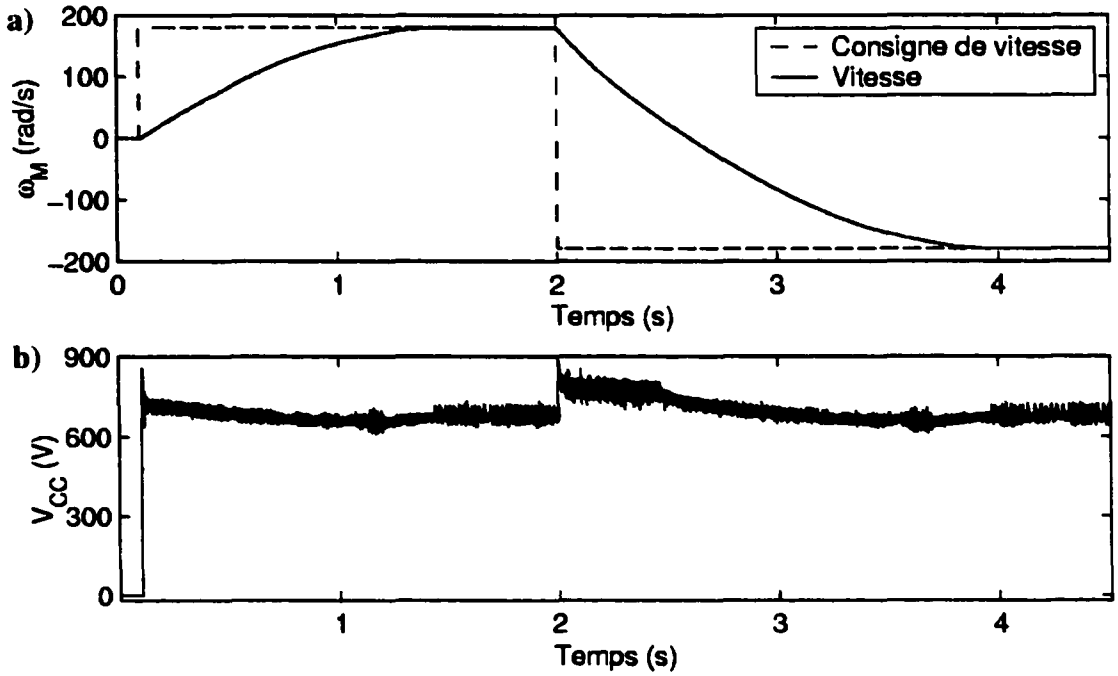


Figure 4-17 **a)** Consigne de vitesse et vitesse de la machine; **b)** tension du bus CC.

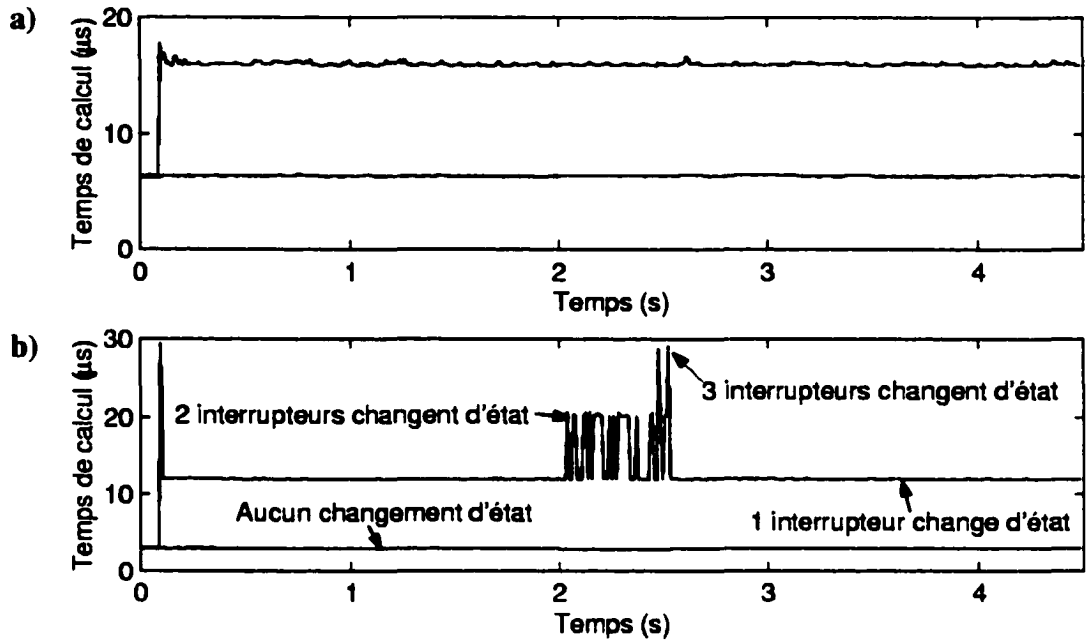


Figure 4-18 Enveloppes des temps d'exécution des deux principales parties: **a)** partie onduleur et machine; **b)** partie source et redresseur.

quant à elle la décomposition LU développée "sur mesure", décrite dans la même section, et qui est insensible au nombre d'interrupteurs qui changent d'état.

En plus d'observer la réponse transitoire de notre simulation, nous avons également vérifié sa stabilité sur une longue période de temps. Nous avons réalisé une simulation en temps réel de plusieurs heures pour voir si les accumulations d'erreurs sur les nombreux pas de calculs ne faisaient pas diverger la simulation. Après huit heures de simulation, l'exécution se poursuivait normalement. Nous n'avons pas effectué de tests sur des périodes plus longues.

4.8 Conclusions

Ce chapitre visait à exposer le travail accompli afin de parvenir à simuler en temps réel un entraînement électrique industriel en le modélisant à l'aide de l'approche par

variables d'état. Après avoir décrit le système simulé, nous avons brièvement énoncé les principales caractéristiques de deux ordinateurs parallèles sur lesquels nous avons eu l'occasion d'expérimenter. Étant donné que le système considéré ne peut être simulé en temps réel sur un seul processeur à un pas de calcul raisonnable, nous avons décrit quelques méthodes de découplage, dont la plupart sont difficilement utilisables en temps réel. Nous avons aussi décrit une tentative de paralléliser une des parties déjà découplée en essayant de tirer avantage de l'architecture à mémoire partagée. Notre conclusion pour cette tentative est que sur ce type d'ordinateur, si les données sont accédées par plusieurs processeurs, il existe un coût associé à ces accès et ce coût s'apparente au coût de la transmission des données via des liens de communication dans une architecture à mémoire distribuée. Par conséquent, le gain réalisé en temps de calcul suite au partage de ces calculs sur deux processeurs est annulé par des invalidations de lignes d'antémémoire ("cache misses"), elles-mêmes causées par une mauvaise localisation, inhérente au problème, des données. Nous avons par la suite identifié de nombreux facteurs qui affectent le temps de calcul. La plupart de ces facteurs étant intimement liés à l'ordinateur utilisé, une bonne connaissance de l'architecture interne de ce dernier est nécessaire. Nous avons également décrit quelques formes d'optimisation qui ont été appliquées aux algorithmes utilisés et qui ont permis des gains intéressants en temps de calcul.

Enfin, le fruit de tous nos travaux a été exposé sous forme d'une étude de cas consistant en la simulation en temps réel de la dynamique de l'entraînement décrit au début du chapitre. Un pas de calcul correspondant à une fréquence quatre fois plus élevée que la fréquence maximale de commutation de l'onduleur a été utilisé pour cette étude de cas. De plus, bien qu'aucun équipement externe n'était raccordé au simulateur lors de nos essais, le temps des conversions des signaux venant éventuellement de l'extérieur et y retournant a été pris en compte et le pas de calcul utilisé permet la simulation du système en temps réel dans ces conditions. Nous sommes convaincus qu'il y a encore moyen de réduire le temps de calcul de la partie source et redresseur. Au moment de publier cette thèse, un de nos collègues venait de développer une méthode alternative de mise à jour

des équations qui semble beaucoup moins complexe en termes du nombre de calculs impliqués [38]. L'implantation de cette méthode n'a cependant pu être complétée à temps pour que des résultats soient présentés dans notre thèse.

Mentionnons enfin que les techniques décrites ici ont une limite, comme il a clairement été démontré dans le cas où la partie onduleur et machine comportait 11 variables d'état plutôt que cinq. Nous sommes cependant confiants que la méthode alternative sus-mentionnée aidera aussi à réduire le temps de calcul de cette partie.

CONCLUSION

Les travaux de recherche liés à cette thèse ont consisté à étudier la possibilité de simuler en temps réel des entraînements électriques. Nous nous sommes intéressés au sujet après avoir constaté que les simulateurs numériques en temps réel sont principalement utilisés pour tester des composants de réseaux électriques, tels les relais de protection, contrôleur de compensateurs statiques, et autres. Nous étions d'avis que les ingénieurs responsables de la conception d'entraînements électriques de grande puissance pouvaient eux aussi bénéficier de tels simulateurs dans leur travail. Cependant, les simulateurs actuels comportent des limitations qui réduisent leur utilité pour la simulation des entraînements électriques. Il est en effet requis, dans certaines conditions, d'ajouter des composants parasites comme charge additionnelle aux bornes des machines, afin d'assurer la stabilité de la simulation. L'ajout de ces composants parasites n'est pas acceptable dans un contexte d'entraînement.

Nous avons choisi d'utiliser dans nos travaux l'approche par variables d'état principalement parce que nous trouvons cette approche bien adaptée au problème. Les équations des machines électriques et des convertisseurs de puissance s'expriment en effet de façon naturelle sous forme d'équations d'état. Tous les simulateurs en temps réel actuels utilisent l'approche nodale. De plus, il y a beaucoup de littérature qui traite de la simulation en temps différé des systèmes électriques et circuits électroniques en utilisant l'approche par variables d'état, mais très peu en temps réel. Ceci nous a motivé à étudier la question à fond.

Au premier chapitre, nous avons décrit une méthode d'obtention automatique des équations d'état de la partie linéaire d'un système électrique général. Nous nous sommes

intéressés à cette méthode suite à de nombreux calculs manuels d'équations d'état causés par le changement fréquent des topologies étudiées. Nous avons écrit en langage Matlab un petit programme très limité, autant pour comprendre les concepts liés à la théorie des graphes et son application aux systèmes électriques que pour nous éviter de refaire ces calculs à la main pour chaque nouvelle topologie. Au fur et à mesure que les systèmes étudiés devenaient plus gros, nous avons constaté que notre programme était relativement performant. Après avoir ajouté les modèles de composantes qui n'étaient pas requis pour simuler des entraînements comme tel, nous avons analysé avec nos routines des réseaux de grande envergure pour réaliser que cette approche était beaucoup plus performante que l'approche originale du Power System Blockset, autant en termes de vitesse de calcul qu'en termes de salubrité des matrices d'état obtenues. Cette constatation a valu à notre groupe de recherche, le G.R.É.P.C.I., une commandite additionnelle permettant l'implantation de cette technique d'obtention des équations d'état dans la version 2.0 du PSB. Le petit programme, requis pour pouvoir commencer nos travaux, est devenu une contribution majeure de notre recherche.

Au chapitre 2, nous avons adapté à notre application une technique originale de mise à jour des matrices d'état suite à un changement d'état d'interrupteur dans un circuit électrique. Nous avons contribué à la conversion de cette technique du langage Matlab au langage C et avons également optimisé la fonction qui implante la méthode. Nous avons également implanté une technique de modélisation des convertisseurs et machines électriques permettant d'obtenir une représentation d'état unique pour le système dans son ensemble. Les variables d'états sont les tensions de condensateurs et les courants d'inductances de la partie convertisseur dans le référentiel *abc* habituel, ainsi que les courants rotoriques et statoriques dans le référentiel *dq0* arbitraire. La méthode permet de solutionner simultanément et sans délai toutes les équations dynamiques de la partie électrique de l'entraînement. Nous croyons que lorsque le même système est simulé de façon modulaire, les délais introduits entre les diverses parties apportent des imprécisions dans les résultats de simulation, qui sont particulièrement visibles lorsque des grands pas de

calcul sont utilisés. Enfin, nous avons également développé un nouveau modèle de machine d'induction multiréférentiel qui a également été incorporé à la version 2.0 du PSB.

Au chapitre 3, nous avons implanté une version de la discrétisation trapézoïdale adaptée aux systèmes variant dans le temps. Cette implantation semblait ne pas demander trop d'effort au départ, mais a finalement exigé du doigté. Nous avons également implanté la même simulation avec la méthode d'intégration MSRP-2, sur laquelle nous comptions beaucoup au début du projet afin de potentiellement éliminer les problèmes que nous croyions liés au fait que la discrétisation trapézoïdale est implicite. Finalement, notre problème lui-même est implicite et une méthode de discrétisation explicite n'apporte rien de plus. De plus, la méthode MSRP-2 exige le calcul de l'exponentielle de la matrice d'état A. Ce calcul est très coûteux et la méthode MSRP-2 est difficilement utilisable à l'échelle de temps réel requise dans notre application.

Enfin, le dernier chapitre est l'aboutissement de tous nos travaux. Après avoir étudié les architectures des ordinateurs parallèles sur lesquels nous avons expérimenté, nous avons choisi une technique de découplage adaptée à notre situation et avons réussi à obtenir de bons résultats qualitatifs en séparant notre système en plusieurs parties. Nous avons pris connaissance des principaux facteurs affectant le temps de calcul et avons ajusté nos algorithmes en conséquence. Nous avons également réussi à tirer avantage de la topologie particulière de notre problème et à réduire ainsi substantiellement le temps de calcul en développant un algorithme creux de factorisation et solution LU adapté au problème. Nous avons également eu la chance d'essayer nos algorithmes sur un ordinateur parallèle à mémoire partagée, ce qui nous a permis de comprendre les mécanismes d'accès à la mémoire hiérarchique dans ce type d'architecture. Nous avons constaté que notre méthode est applicable en temps avec un nombre restreint de variables d'état dans la partie onduleur et machine. Cependant, il semble que les entraînements de moyenne et grande puissance soient effectivement caractérisés par un nombre réduit de variables

d'état et la simulation que nous réalisons en temps réel correspond à un entraînement industriel réel. Enfin, c'est dorénavant l'autre partie du système, la source et le redresseur, qui domine le temps de calcul. Nous sommes confiants que les temps de calcul des deux parties seront réduits de façon appréciable lors de l'implantation future d'une technique de mise à jour des équations du système récemment développée par un de nos collègues.

Nous devons avouer qu'au départ, nous étions fort sceptiques de parvenir à simuler un entraînement électrique en temps réel en utilisant l'approche par variables d'état. Nous y sommes cependant parvenus et donc nous concluons que nous avons atteint nos objectifs. La principale contribution de notre travail de recherche consiste en l'intégration de diverses techniques de modélisation, de simulation et d'intégration numérique, en un tout cohérent solutionnant de façon satisfaisante un problème bien particulier, celui de simuler en temps réel de façon stable et précise un entraînement électrique industriel.

Nous identifions plus spécifiquement deux contributions originales de nos travaux. D'abord, il s'agit du fait que l'approche par variables d'état a été utilisée pour simuler en temps réel un système électrique. Ensuite, nos simulations sont basées sur l'utilisation d'un modèle de plus haut niveau que les composants individuels typiquement utilisés dans ce genre de simulation. Il s'agit de la méthode décrite à la section 2.3, où l'ensemble convertisseur et machine est solutionné sans délai dans un seul système d'équations. Cette approche permet d'éliminer certains problèmes de stabilité et de précision dus a la solution séparée des mêmes composants.

RECOMMANDATIONS

Nos conclusions quant à la précision de nos algorithmes sont essentiellement basées sur une comparaison avec un autre logiciel de simulation, le PSB. Cependant, un étudiant de maîtrise a récemment commencé son projet au G.R.É.P.C.I. et il va interconnecter la véritable commande pour laquelle nous avons modélisé l'entraînement avec notre simulation en temps réel. Le comportement de l'entraînement sera alors analysé et probablement comparé à des résultats expérimentaux, qui nous permettront de valider de façon plus déterminante nos algorithmes.

Notre groupe est également en contact avers divers fabricants d'entraînements afin de déterminer leur intérêt pour un éventuel simulateur numérique adapté à leurs besoins. Les travaux décrits dans cette thèse pourraient donc éventuellement faire l'objet d'une commercialisation si l'industrie des entraînements démontre un intérêt.

Les résultats présentés au chapitre 2 nous ont également permis de conclure que les algorithmes proposés sont performants et précis en temps différé aussi. Ceci tombe sous le sens, puisque toute optimisation visant à améliorer les performances d'une simulation en temps réel va également être bénéfique pour la simulation en temps différé du même système. Nous sommes actuellement impliqués dans le développement d'une autre librairie spécialisée pour la simulation des entraînements et songeons à y intégrer nos travaux de doctorat. Nous n'avons cependant pas encore évalué comment les modèles de machines autres que la machine asynchrone pourraient être adaptés aux méthodes préconisées ici.

Un dernier point concerne la linéarité du circuit magnétique de la machine. Dans nos travaux, nous avons supposé un circuit magnétique linéaire, c'est à dire que nous ne simulons pas la saturation magnétique de la machine. Nous n'avons pas évalué comment la saturation pourrait être incorporée dans nos simulations. La saturation magnétique est plus facile à implanter lorsque les variables d'état associées à la machine sont les flux, comme nous avons été en mesure de le constater lors du développement du modèle de machine synchrone pour le PSB 1.0 [2]. Ceci demanderait probablement un remaniement majeur de nos systèmes d'équations et pourrait faire l'objet d'une étude future s'il devient important de simuler la saturation magnétique.

Enfin, une technique alternative de mise à jour des matrices d'état suite à un changement d'état d'interrupteur, récemment développée par un de nos collègues, semble très prometteuse et devrait être étudiée plus à fond.

RÉFÉRENCES BIBLIOGRAPHIQUES

1. The Mathworks, inc. (1998). Power System Blockset User's Guide.

2. Champagne, R. (1997). Conception et validation d'un modèle de machine synchrone avec saturation magnétique (mémoire de maîtrise). École de technologie supérieure, Montréal.

3. Balabanian, N., Bickart, T.A. (1969). Electrical Network Theory. New York : John Wiley & Sons, inc.

4. Rohrer, R.A. (1970). Circuit theory: an introduction to the state variable approach. New York : McGraw-Hill.

5. Gille-Maisani,. J.-C., Clique, M. (1975). La représentation d'état pour l'étude des systèmes dynamiques. Paris : Eyrolles.

6. Chua, L.O., Lin, P.M. (1975). Computer-Aided Analysis of Electronic Circuits: Algorithms and Computational Techniques. New Jersey : Prentice Hall.

7. Rajagopalan, V. (1987). Computer-Aided Analysis of Power Electronic Systems. New York : Marcel Dekker, inc.

8. Lauw, H.K., Meyer, W.S. (1982). Universal machine modeling for the representation of rotating electric machinery in an electromagnetic transients program. IEEE Transactions on Power Apparatus and Systems, 101(6), pp. 1342-1351.

9. Vainio, O., Seppo, J.O., Pasanen, J. (1992). A Digital Signal Processing Approach to Real-Time AC Motor Modeling. IEEE transactions on Industrial Electronics, 39(1), pp. 36-45.

10. Gehlot, N.S., Alsina, P.J. (1993). A Discrete Model of induction Motors for Real-Time Control Applications. IEEE transactions on Industrial Electronics, 40(3), pp. 317-325.

11. Haskew, T.A., Biakod, R. (1995). Real-Time simulation of a twelve-pulse full converter. Electric Power Systems Research, 34, pp. 57-63

12. Kleinhans, C., Harley, R.G., Diana, G., McCulloch, M. (1994). The Application of CASED as a Simulation Tool for Design and Analysis of Variable Speed Drives. IEEE Industry Applications Society (IAS) Annual Meeting, pp. 750-757.

13. Chhaya, S.M., Bose, B.K. (1995). Expert system aided automated design, simulation and controller tuning of AC drive system. IECON (Industrial Electronics Conference) Proceedings, pp. 712-718

14. Matuonto, M., Monti, A., Torri, G. (1994). Testing Software Through Real-Time Model Of The Plant: An Experimental Example. IECON (Industrial Electronics Conference) Proceedings, pp. 1807-1812.

15. Dezza, F.C., Cristaldi, L., Ferrero, A., Monti, A. (1996). Real-time virtual system for electric drive testing: basic concepts and implementation. Mediterranean Electrotechnical Conference, MELECON, pp. 513-516.

16. Zavahir, J.M., Arrillaga, J., Watson, N.R. (1992). Hybrid electromagnetic transient simulation with the state variable representation of HVDC converter plant. IEEE Transactions on Power Delivery, 8(3), pp. 1591-1598.

17. Kang, Y., Lavers, J.D. (1996). Transient analysis of electric power systems: reformulation and theoretical basis. IEEE Transactions on Power Systems, 11(2), pp. 754-760.

18. Wasynczuk, O., Sudhoff, S.D. (1996). Automated State Model Generation Algorithm for Power Circuits and Systems. IEEE Transactions on Power Systems, 11(4), pp. 1951-1956.

19. De Abreu-Garcia, J.A., Hartley, T.T. (1990). Multistep matrix integrators for real-time simulation. Control and dynamic systems, 38, pp. 211-272.

20. Hartley, T.T., Beale, G.O., Chicatelli, S.P. (1994). Digital Simulation of Dynamic Systems: A Control Theory Approach. New Jersey : Prentice Hall.

21. Sudha, S.A., Chandrasekaran, A., Rajagopalan, V. (1993). New approach to switch modelling in the analysis of power electronic systems. IEE proceedings-B, 140(2), pp. 115-123.

22. Electromagnetic Transients Program (EMTP), Development Coordination Group of the EMTP.

23. Dessaint, L.-A., Al-Haddad, K., Le-Huy, H., Sybille, G., Brunelle, P. (1999). A Power System Simulation Tool Based on Simulink. IEEE Transactions in Industrial Electronics, 46(6), pp. 1252-1254.

24. Krause, P.C., Thomas, C.H. (1965). Simulation of symmetrical induction machinery. IEEE Transactions on Power Apparatus and Systems, PAS-84(11), pp. 1038-1053.

25. Krause, P.C., Wasynczuk, O., Sudhoff, S.D. (1995). Analysis of electric machinery. New York : IEEE Press.

26. Davat, B. (1979). Étude - Mise au point et applications d'une méthode de simulation globale de convertisseurs statiques connectés à des charges électriques complexes (thèse de doctorat). Institut national polytechnique de Toulouse, Toulouse.

27. Takahashi, I., Noguchi, T. (1986). A new quick-response and high-efficiency control strategy of an induction motor. IEEE Transactions on Industry Applications, IA-22(5), pp. 820-827.

28. Champagne, R., Dessaint, L.-A., Sybille, G., Khodabakhchian, B. (2000). An approach for real-time simulation of electric drives. Canadian Conference on Electrical and Computer Engineering (CCECE 2000), Halifax, May 7-10 2000.

29. Champagne, R., Dessaint, L.-A., Sybille, G., Casoria, S. (1999). Real-Time Simulation of Electrical drives Using the State Variable Approach. Third International Conference on Digital Power System Simulators (ICDS '99), Västerås, Sweden, May 25-28 1999.

30. The Mathworks, inc. (1998). Writing S-Functions.

31. The Mathworks, inc. (1998). Application Program Interface Reference Guide (disponible en version électronique seulement).

32. Franklin, G.F., Powell, J.D., Workman, M.L. (1990). Digital Control of Dynamic Systems. New York : Addison Wesley, inc.

33. Golub, G.H., Van Loan, C.F. (1996). Matrix Computations. Baltimore : Johns Hopkins University Press

34. Thomas, R.J., Thorp, J.S., Linke, S., Pottle, C., Strohman, R. (1985). The Cornell University Kettering Energy Systems Laboratory. IEEE Transactions on Power Apparatus and Systems, PAS-109, pp. 2302-2305.

35. Silicon Graphics, inc. Origin2000 and Onyx2 Performance Tuning and Optimization Guide, document no. 007-3430-002.

36. Lewis, B., Berg, D.J. (1998). Multithreaded Programming with PThreads. Mountain View : Sun Microsystems, inc.

37. Press, W.H., Teukolsky, A.A., Vetterling, W.T., Flannery, B.P. (1992). Numerical Recipes in C. New York : Cambridge University Press.

38. De Kelper, B. (2001) Thèse de doctorat en préparation, publication prévue fin 2001.

ANNEXE A

Paramètres des simulations des chapitres 2 et 3

Source et bus à courant continu

- $V_{dc} = 700$ V
- $R_{déch} = 30$ kΩ
- C = 1570 μF, R = 1 mΩ

Onduleur

- Résistance des interrupteurs à l'état passant: 1 mΩ
- Résistance des interrupteurs à l'état bloqué: infinie
- Résistance en parallèle avec chaque interrupteur: 1 MΩ

Machine asynchrone

- Puissance nominale: 50 HP
- Tension nominale ligne-ligne: 460 V
- Fréquence nominale: 60 Hz
- Stator: $R_s = 0.087$ Ω, $L_{ls} = 0.8$ mH
- Rotor: $R'_r = 0.228$ Ω, $L'_{lr} = 0.8$ mH
- Inductance magnétisante: 34.7 mH
- Inertie combinée du rotor et de la charge: 1.662 kg.m^2
- Nombre de paires de pôles: 2

Commande

- Type: Commande directe du flux et du couple (DFTC)
- Fréquence maximale de commutation de l'onduleur: 3.9 kHz

ANNEXE B

Paramètres des simulations du chapitre 4

Réseau équivalent 550 V / 375 kVA

- Sources de tension: 550 V rms ligne-ligne
- Résistance: 0.08 Ω
- Inductance: 2 mH

Redresseur

- Résistance des interrupteurs à l'état passant: 1 mΩ
- Résistance des interrupteurs à l'état bloqué: infinie
- Circuit d'amortissement RC série: R = 66 Ω, C = 2.2 μF

Freinage dynamique

- R = 10 Ω

Bus à courant continu

- Résistance de décharge: 30 kΩ
- Condensateur: 1570 μF

Onduleur

- Résistance des interrupteurs à l'état passant: 1 mΩ
- Résistance des interrupteurs à l'état bloqué: infinie
- Résistance en parallèle avec chaque interrupteur: 1 MΩ

Machine asynchrone

- Puissance nominale: 50 HP
- Tension nominale ligne-ligne: 460 V
- Fréquence nominale: 60 Hz
- Stator: $R_s = 0.087\ \Omega$, $L_{ls} = 0.8$ mH
- Rotor: $R'_r = 0.228\ \Omega$, $L'_{lr} = 0.8$ mH
- Inductance magnétisante: 34.7 mH
- Inertie combinée du rotor et de la charge: 1.662 $kg.m^2$
- Nombre de paires de pôles: 2

Commande

- Type: Commande directe du flux et du couple (DFTC)
- Fréquence maximale de commutation de l'onduleur: 3.9 kHz