CONTRIBUTION METHODOLOGIQUE A LA CONCEPTION SOUS CONTRAINTES DE DISPOSITIFS ELECTROMAGNETIQUES

Coralie Coutel

INSTITUT NATIONAL POLYTECHNIQUE DE GRENOBLE

N° attribué par la bibliothèque
|_|_|_|_|_|_|_|_|_|_|

THESE

pour obtenir le grade de

DOCTEUR DE L'INSTITUT NATIONAL POLYTECHNIQUE DE GRENOBLE

(Arrêté ministériel du 30 Mars 1992)

Spécialité Génie Electrique

Préparée au Laboratoire d'Electrotechnique de Grenoble
Dans le cadre de l'Ecole Doctorale Electronique, Electrotechnique, Automatique, Télécommunication, Signal

présentée et soutenue publiquement par

Coralie COUTEL
Ingénieur ENSIEG

le 20 Octobre 1999

CONTRIBUTION METHODOLOGIQUE A LA CONCEPTION SOUS CONTRAINTES DE DISPOSITIFS ELECTROMAGNETIQUES

JURY

Messieurs	J.M. KAUFFMANN	*Président et Rapporteur*
	A. RAZEK	*Rapporteur*
	C. VANDERSCHAEGHE	*Examinateur*
	C. CHILLET	*Examinateur*
	F. WURTZ	*Co-encadrant*
	J. BIGEON	*Directeur de Thèse*

INSTITUT NATIONAL POLYTECHNIQUE DE GRENOBLE

N° attribué par la bibliothèque
|_|_|_|_|_|_|_|_|_|_|

THESE

pour obtenir le grade de

DOCTEUR DE L'INSTITUT NATIONAL POLYTECHNIQUE DE GRENOBLE

(Arrêté ministériel du 30 Mars 1992)

Spécialité Génie Electrique

Préparée au Laboratoire d'Electrotechnique de Grenoble
Dans le cadre de l'Ecole Doctorale Electronique, Electrotechnique, Automatique, Télécommunication, Signal

présentée et soutenue publiquement par

Coralie COUTEL
Ingénieur ENSIEG

le 20 Octobre 1999

CONTRIBUTION METHODOLOGIQUE A LA CONCEPTION SOUS CONTRAINTES DE DISPOSITIFS ELECTROMAGNETIQUES

JURY

Messieurs	J.M. KAUFFMANN	*Président et Rapporteur*
	A. RAZEK	*Rapporteur*
	C. VANDERSCHAEGHE	*Examinateur*
	C. CHILLET	*Examinateur*
	F. WURTZ	*Co-encadrant*
	J. BIGEON	*Directeur de Thèse*

Remerciements

Ce travail a été réalisé au sein de l'équipe Conception et Diagnostic Intégrés du Laboratoire d'Electrotechnique de Grenoble.

Je remercie :

M. Jean-Marie Kauffmann, professeur à l'Université de Franche-Comté, pour m'avoir fait l'honneur de présider le jury et avoir accepté d'être rapporteur de ce travail.

M. Adel Razek, Directeur de Recherches au CNRS pour avoir accepté d'être rapporteur de ce travail.

MM. Christian Vanderschaeghe, Ingénieur à SOMFY Industrie, et Christian Chillet, Chargé de Recherches au CNRS, pour avoir accepté de faire partie de mon jury.

M.M. Frédéric Wurtz, Chargé de Recherches au CNRS, et Jean Bigeon, Directeur de Recherches, pour avoir encadré mon travail de thèse.

Je remercie également les directeurs du Laboratoire d'Electrotechnique de Grenoble pour m'avoir accueillie au sein de celui-ci, les anciens : M.M. Jean-Claude Sabonnadière et Robert Perret et les nouveaux : M.M. Jean-Pierre Rognon et Gérard Meunier.

Ne pouvant pas citer toutes les personnes avec lesquelles j'ai eu l'occasion de travailler, je remercie tous les chercheurs du Laboratoire et particulièrement les membres de l'Equipe Conception et Diagnostic Intégrés, que j'ai côtoyés au long de ma thèse et qui m'ont permis d'avancer plus rapidement sur différents problèmes.

Je n'oublie pas toute l'équipe informatique qui a été d'agréable secours dès que l'informatique me faisait des misères.

Je voudrais aussi remercier le reste du personnel du Laboratoire, les ITA, et particulièrement les secrétaires qui contribuent largement à maintenir le moral des thésards.

Je remercie tous les amis qui m'ont soutenue dans les moments difficiles. Avoir des amis c'est très utile... Ils se reconnaîtront.

Bon courage.

Table des Matières

Introduction

Introduction

Les concepteurs de dispositifs électromagnétiques, comme des machines électriques par exemple, se trouvent en permanence confrontés à la difficulté de devoir concevoir et dimensionner des dispositifs à la géométrie souvent complexe, dans lesquels interviennent des phénomènes non linéaires et fortement couplés (électromagnétiques, thermiques, mécaniques,...) et devant répondre à un cahier des charges induisant des contraintes sur les paramètres.

En outre, la phase de conception devient un enjeu important dans les entreprises à l'heure actuelle. Les prototypes sont chers et le besoin d'évaluer en simulation le dispositif à concevoir et de l'optimiser devient primordial.

Le besoin d'outils informatiques dédiés à la conception est donc d'actualité.

Nous proposons ici une contribution méthodologique, visant à utiliser les moyens informatiques mis à disposition actuellement, afin d'exploiter au mieux les possibilités des modèles analytiques dans la phase de dimensionnement sous contraintes.

La première partie de notre étude présentera un état de l'art sur la conception sous contraintes en génie électrique, ainsi que l'étude des outils existants, qui nous permettront de définir les fonctionnalités à remplir pour le dimensionnement sous contraintes à l'aide de modèles analytiques. Notre approche vise à unifier les méthodes existantes.

Dans la seconde partie, nous nous intéresserons à l'étude d'un outil existant, Pascosma, qui répond à la problématique d'optimisation sous contraintes de machines électriques. Cette étude vise à dégager les difficultés rencontrées sur des cas concrets dans la manipulation des modèles analytiques.

Dans la partie suivante, nous nous intéresserons à un problème particulier qui se pose avec les modèles analytiques : la résolution de systèmes implicites en vue de l'optimisation sous contraintes, et nous proposerons une méthode de prise en charge de ce problème.

Enfin, à partir des constatations effectuées préalablement, nous définirons dans la quatrième partie le cahier des charges à remplir par notre nouvelle approche, et nous proposerons les bases d'une architecture y répondant.

la cinquième partie présentera l'implantation informatique de cette architecture, ainsi que les choix techniques qui y ont présidé.

Finalement, nous effectuerons le bilan en ce qui concerne la satisfaction du cahier des charges posé, et présenterons les perspectives de ce travail.

Chapitre 1. Etat de l'art

I – Le contexte de la conception en Génie Electrique

1.1 La problématique de conception

1.1.1 Définition de la conception

Selon l'analyse de [HABI 98], dont le but était de connaître l'activité de conception, « la conception est fondamentale pour le positionnement des entreprises sur le marché. Actuellement, la qualité n'est plus le souci majeur des entreprises – la bonne maîtrise des outils et des techniques de contrôle de la qualité est un fait accompli », concourant, selon le même auteur, à atteindre l'objectif "zéro défaut". De plus, il indique que la maîtrise de la conception est alors l'élément déterminant de l'amélioration de la position concurrentielle d'une entreprise et qu'il est nécessaire de garantir la transmission du savoir-faire des experts et d'être capable de réagir rapidement à la demande du marché.

Y. Harani [HARA 97] définit ainsi la tâche de conception : "Etant donné un ensemble de composants, un ensemble de relations entre ces composants, une tâche de conception consiste à spécifier comment réaliser un objet à l'aide de ces composants, de manière à satisfaire un ensemble d'exigences. Bien que dans certains types d'activités l'ensemble des composants ne soit pas toujours fini et que de nouveaux composants doivent être conçus, cela ne pose théoriquement pas de problème particulier car la conception peut être récursive."

Dans l'exemple concret de la conception d'un moteur roue, C. Espanet [ESPA 99] définit la conception comme correspondant à la définition d'un objet ou d'un système (ensemble d'objets) répondant à un besoin défini dans un cahier des charges.

Dans ces deux définitions, on voit bien l'importance soulignée des exigences à remplir par le concepteur, formulées dans le cahier des charges.

[HABI 98] précise que l'activité de conception est la phase initiale de tout produit. Une fois la spécification du "comment réaliser l'objet" clairement effectuée, les tâches restantes de la réalisation sont plus faciles à mettre en œuvre et relèvent du pratique. De plus, si la conception est bien menée, elle envisage au maximum l'optimisation du produit et de sa fabrication. C'est pourquoi la phase de conception est souvent coûteuse en temps dans le cycle de production du produit.

L'élaboration du cahier des charges doit spécifier les fonctions à réaliser par le dispositif à concevoir, et les contraintes attachées à ces fonctions et à l'environnement dans lequel va être placé ce dispositif. Par exemple, pour un électroaimant, la fonction à réaliser

sera l'ouverture d'un circuit, une des contraintes du cahier des charges pourra être la valeur de la force dans l'entrefer et une autre par exemple pourra porter sur les dimensions si on cherche à minimiser l'encombrement. Le cahier des charges peut comprendre des contraintes économiques (coût, main d'œuvre nécessaire,...), mécaniques, thermiques,... Il peut aussi comprendre une ou plusieurs fonctions objectifs, qui sont en quelque sorte les contraintes principales, comme par exemple la maximisation du couple massique, ou la minimisation du coût.

1.1.2 Les différentes phases de la conception

Dans de nombreux travaux en génie électrique [TRIC 91, ESPA 99], on décrit le processus de conception comme comportant plusieurs grandes phases. Ce processus est représenté sur la figure I-1.

Elaboration du cahier des charges

Nous avons précisé dans le paragraphe précédent l'importance de cette étape. Le cahier des charges comprend à la fois des spécifications sur la structure et des contraintes pour le dimensionnement.

Choix de structure

Cette phase de préconception est l'étape de détermination des structures possibles grâce aux connaissances et à la créativité du concepteur. En fonction du cahier des charges, on en sélectionne une ou plusieurs [ESPA 99] qui seront ensuite discriminées dans les phases suivantes.

On peut distinguer deux axes dans les approches de la démarche de conception. La conception dite "innovante" [BIGE 94, SHOOD 95] implique qu'on ait apporté une nouveauté par rapport à des structures déjà existantes (nouvel élément, combinaison de plusieurs structures existantes, forme entièrement nouvelle,...). La conception "routinière", quant à elle, s'appuie fortement sur des structures déjà connues par le concepteur et ré-exploite le savoir-faire déjà développé pour concevoir un produit similaire, la ressemblance pouvant porter sur l'identité du produit (structure, fonction, comportement,...) ou sur son processus de conception. Ce type de conception est utilisé typiquement dans le cas de gammes de produits (moteurs de différentes puissances) que l'on décline. Il existe des outils permettant, soit de créer de nouvelles structures, tels que les outils utilisant la description fonctionnelle [LECH

98], soit de choisir parmi les structures connues en utilisant la base de connaissances des concepteurs tels que les systèmes experts [ESCA 96].

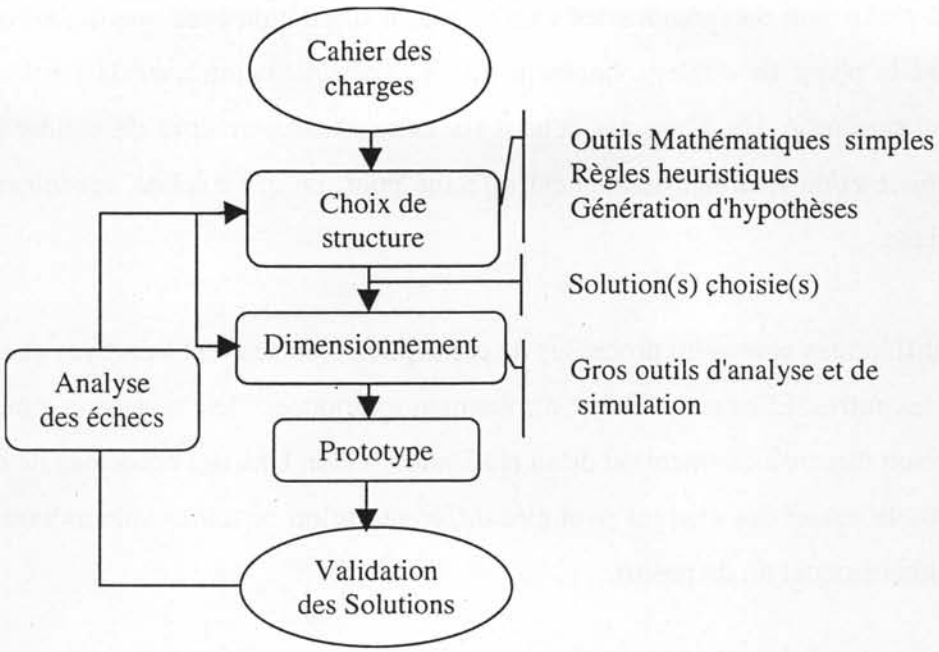


Fig I-1. Organigramme général du processus de conception

Dimensionnement

Cette phase doit permettre de trouver une solution faisable respectant le cahier des charges. Elle peut comprendre ou être suivie d'une optimisation : on cherche dans l'espace des solutions possibles une solution optimale. Pour parcourir l'espace des solutions ou même en trouver une, il faut modéliser la structure et son comportement. Si l'on a sélectionné plusieurs structures, on effectuera un dimensionnement pour chacune d'entre elles pour éventuellement en éliminer certaines au regard du respect du cahier des charges. Suivant la méthode de dimensionnement qui a été choisie, cette phase peut éventuellement amener à plusieurs solutions valides qui seront discriminés par la suite. Nous décrirons plus précisément cette phase dans la partie suivante.

Validation

Il s'agit de vérifier si la structure choisie et dimensionnée convient bien et de discriminer éventuellement plusieurs structures retenues ou plusieurs solutions de dimensionnement. On peut en effet ne pas avoir pu tout prendre en compte comme des contraintes technologiques ou de fabrication dans le cahier des charges. Dans la phase de dimensionnement, la structure peut ne pas avoir été appréhendée dans sa totalité. Des phénomènes physiques fins peuvent également ne pas avoir été pris en compte dans le modèle

utilisé. On peut aussi avoir à vérifier les choix et estimations effectués, portant sur les différentes grandeurs du système.

Cette phase peut comprendre des validations en simulation avec une modélisation plus fine que dans la phase de dimensionnement ou en expérimentation avec la fabrication d'un prototype qui sera testé. L'analyse des échecs sur cette phase permettra de valider ou non la structure choisie et/ou le dimensionnement effectué pour, en cas d'échec, revenir à la phase remise en cause.

Les différentes phases du processus de conception sont souvent itératives et rebouclent les unes sur les autres. Elles peuvent être étroitement imbriquées : le choix de la structure peut dépendre de son dimensionnement ou de sa réalisation réelle. Une des équations de contrainte spécifiée dans le cahier des charges peut être différente selon certaines valeurs que prend un paramètre dimensionnel du dispositif.

1.1.3 Positionnement de notre approche

A cause de l'imbrication des différentes étapes de la conception, de nombreux travaux antérieurs ont cherché à réaliser la conception avec un seul outil permettant de déterminer à la fois la structure et le dimensionnement optimaux [TRIC 91, ESCA 96]. Ces approches, souvent de type système expert, combinent des règles heuristiques de conception et des équations analytiques modélisant le système à concevoir.

Cependant, il nous semble que les problèmes de choix de structure et de dimensionnement ne se posent pas dans les mêmes termes et que dans une optique de conception routinière, ils seront mieux résolus par des outils différents.

Dans le problème de choix de structure, il s'agit de s'appuyer sur une base de connaissances, une expertise pour effectuer un choix parmi une bibliothèque de structures possibles ou reproduire le raisonnement du concepteur pour trouver la structure heuristique habituelle, tandis que dans le problème du dimensionnement, il s'agit à partir d'une modélisation de la structure choisie, d'appliquer une méthode de dimensionnement pour trouver dans l'espace des solutions possibles une solution satisfaisante pour le concepteur – c'est un problème plus purement mathématique.

Dans les approches système expert, on a vu les limites de maintenabilité de ces outils. Les règles régissant le processus de conception sont difficiles à récolter parmi les experts et parfois à transcrire. De plus, il est peut être un peu trop ambitieux de penser qu'on pourra créer des outils qui feront le travail à la place du concepteur : c'est à dire trouver la meilleure

structure avec un dimensionnement optimal pour un problème donné. Notre ambition est plutôt ici d'offrir des outils d'aide, génériques, au concepteur, et qui puissent rapidement être opérationnels sur un nouveau problème.

Dans notre approche, nous nous sommes intéressés à cette deuxième phase du processus de conception décrit précédemment : le dimensionnement d'une structure donnée de dispositif électromagnétique à concevoir.

Cette phase représente le travail le plus courant du concepteur en génie électrique. Il s'agit souvent d'élaborer une gamme de produits de même nature (comme des moteurs électriques) mais remplissant des cahiers des charges différents en ce qui concerne le dimensionnement (puissance, volume par exemple). De plus, les structures des dispositifs électrotechniques sont connues pour la plupart (moteurs, transformateurs, convertisseurs, accouplements). Ce sera au concepteur de choisir une ou plusieurs structures possibles et ensuite d'effectuer les dimensionnement correspondants pour sélectionner la meilleure vis à vis du cahier des charges.

1.2 Le problème de dimensionnement

Nous intéressant à la phase de dimensionnement, nous allons la décrire ici plus en détails, avec les différentes étapes qu'elle comporte et les choix techniques qui lui sont associés.

1.2.1 Les étapes du dimensionnement

Les connaissances servant au dimensionnement d'un dispositif électromagnétique sont [SALD 88] :

- la représentation paramétrée du système à concevoir, ce qui correspond à la description de la structure à dimensionner.
- un modèle mathématique, qui permet d'évaluer les autres paramètres du système, notamment ses performances
- une logique de dimensionnement, qui décrit la démarche d'obtention d'un ensemble de valeurs des paramètres respectant le cahier des charges spécifié.

Dans la phase de dimensionnement, une fois le choix de la structure fait, il faut en effet pouvoir évaluer quantitativement l'expression des variables qui caractérisent le fonctionnement et les qualités du système, en particulier celles qui apparaissent dans le cahier des charges, en fonctions des grandeurs descriptives du système. Cette partie se nomme

l'analyse du système et repose sur l'évaluation du modèle du système. Une phase de modélisation, de choix du modèle de la structure choisie, est donc indispensable.

Le dimensionnement proprement dit consiste en l'opération inverse : à partir d'un fonctionnement à réaliser spécifié dans le cahier des charges, il s'agit de trouver les paramètres du système correspondant. Le problème de dimensionnement peut avoir aucune, une, plusieurs ou même une infinité de solutions. Il nécessite donc une méthode pour :

- au minimum trouver une solution s'il en existe au moins une
- et mieux, parcourir l'espace des solutions pour trouver la solution la plus optimale au regard des spécifications du cahier des charges, notamment d'une éventuelle fonction objectif, et la plus optimale vis à vis des moyens utilisés pour la trouver.

On parle généralement de problème direct pour la modélisation et de problème inverse pour le dimensionnement. On retiendra donc que, s'il est nécessaire de disposer d'un modèle de la structure à dimensionner, l'opération de dimensionnement nécessite de mettre au point une méthodologie et une stratégie pour réaliser l'inversion du modèle.

1.2.2 Analyse : choix de la modélisation

Dans la littérature, le dimensionnement s'effectue le plus souvent à l'aide de modèles numériques très fins, type éléments finis, sur lesquels on applique une méthode essai/erreur [VEIN 60] ou une méthode d'optimisation pour dimensionner [SALD 88b].

Nous avons choisi dans notre approche d'utiliser des modèles analytiques pour effectuer le dimensionnement. La principale raison de ce choix est qu'ils sont incontournables dans la phase de préconception de tout dispositif électromagnétique. On peut citer le cas des machines asynchrones où pratiquement tous les livres de dimensionnement de ces machines s'appuient sur des modèles analytiques [ALGE 70, COCH 89]. Certains concepteurs s'appuient sur des abaques [BELOT] ou des règles d'expertise pour cette étape, qui s'avèrent être des morceaux de modèle analytique simplifiés. Les modèles analytiques sont donc l'outil de base du concepteur, les modèles plus fins, numériques, n'intervenant en général que lorsque la structure est clairement définie.

En outre, les méthodes numériques sont plus lourdes à manipuler et ne permettent pas au concepteur de voir les interdépendances des paramètres à partir du moment où le problème devient complexe. La connaissance physique du système à concevoir n'y est pas formalisée. De plus ces méthodes sont souvent coûteuses en temps de calcul, et même si ce n'est pas l'élément primordial à prendre en compte, il peut devenir excessif si l'on doit évaluer plusieurs

fois le modèle, ce qui est souvent le cas dans le dimensionnement, en particulier dans les méthodes essai/erreur. Les éléments finis sont souvent utilisés pour le dimensionnement fin d'une partie seulement de l'ensemble à concevoir, ce qu'on appelle généralement « optimisation de forme » [KADD93]. On peut dans cette optique les employer en fin de conception, pour affiner une structure dimensionnée par d'autres méthodes.

On peut trouver aussi des méthodes reposant pour la partie analyse sur des réseaux de neurones [MOHA 94] ou encore des plans d'expérience [BRISS 95], qui sont des méthodes basées sur l'interpolation ou l'extrapolation du comportement observé du système. Ces méthodes utilisent des mesures effectuées sur le système. Il faut donc a priori une structure de départ observable, ce qui n'est pas toujours le cas dans la conception et on admet qu'on peut ensuite extrapoler le comportement d'une maquette au système dimensionné. Par extension, ces méthodes s'appuient sur une modélisation numérique du système. C'est à dire qu'au lieu d'interpoler le comportement observé, on interpole un nuage de points de fonctionnement issu de la simulation numérique. Ce procédé vise à alléger la partie évaluation qui comme on l'a vu ci-dessus peut être coûteuse avec des modèles numériques complexes. Cependant, l'acquisition des données (l'apprentissage, le relevé du comportement) est longue et on n'obtient qu'une interpolation d'un modèle numérique qui peut ne pas être très juste. De plus, comme pour les modèles numériques, on ne formalise pas les relations physiques entre les paramètres du système.

1.2.3 Parcours de l'espace des solutions

Trouver une solution peut se faire comme on l'a vu par une boucle itérative simple essai/erreur appliquée à la partie analyse. Il existe aussi des algorithmes plus sophistiqués pour effectuer ce genre de boucles. De plus, comme on l'a vu plus haut, le problème de dimensionnement peut avoir plusieurs solutions et il est intéressant pour le concepteur de pouvoir déterminer la meilleure solution – ou une des meilleures solutions – au regard du cahier des charges dans cet espace, c'est à dire optimiser le dimensionnement.

Dimensionnement par approche procédurale

Lorsqu'on dispose d'un modèle analytique, et d'un cahier des charges, la première idée qui vient pour trouver des valeurs pour les paramètres est d'en fixer certains et d'en déduire les autres au fur et à mesure, en revenant en arrière en cas de violation d'une contrainte et en effectuant des itérations pour améliorer. Une fois cette démarche effectuée, on peut réutiliser le même processus pour tout dimensionnement de ce modèle, avec des valeurs différentes,

mais avec un cahier des charges du même type, c'est à dire le même type de contrainte sur les mêmes paramètres du modèle. Ces procédures, très classiques sont décrites dans les articles et ouvrages usuels concernant le dimensionnement des machines électriques [BELOT, SLEM 94].

La solution trouvée par ce genre de méthode appartient à l'espace des solutions mais n'est pas forcément optimale. De plus, la procédure décrite est spécifique à un problème de conception donné car elle est liée au cahier des charges imposé, et doit être redéfinie pour un autre. Ce genre de travail nécessite une bonne connaissance de l'application de la part du concepteur.

Le modèle analytique utilisé peut être trop compliqué alors pour être manipulé de cette façon : les interdépendances entre les paramètres ne sont pas assez visibles et fixer un paramètre en connaissant l'influence qu'il aura sur les autres est parfois difficile. Ce genre de méthode s'appuie souvent sur un modèle simplifié pour cette raison et la qualité du résultat obtenu peut s'en ressentir. Le dimensionnement est alors à affiner par des méthodes plus précises.

Systèmes experts

Comme on l'a vu précédemment, les systèmes experts peuvent servir au choix de la structure, mais ils intègrent également des règles qui permettent d'optimiser le dimensionnement de cette structure [FRAN 94, GENT 91]. Dans cette optique, on peut les utiliser comme outils de dimensionnement, mais le problème de recueillir les règles utilisées par les experts et de les implanter clairement et sans conflit, demeure.

Dans notre approche, visant à utiliser des modèles analytiques, les systèmes experts ne nous semblent pas l'outil le mieux adapté pour exploiter l'information contenue dans ceux-ci pour la conception.

Méthodes d'optimisation

Les méthodes d'optimisation sont les plus utilisées pour le dimensionnement des systèmes électromagnétiques. Elles s'appliquent en général à tout type de modèle : analytique ou numérique, mais certaines sont plus adaptées à l'un ou l'autre type.

Les méthodes d'optimisation parcourent l'espace des solutions en cherchant la solution optimale au regard du cahier des charges imposé, c'est à dire la solution qui minimise la fonction objectif choisie (cf. Chapitre1 § 1.1) en respectant les contraintes posées.

En général, une seule fonction objectif peut être prise en compte par les algorithmes d'optimisation. Néanmoins, on peut vouloir effectuer un dimensionnement suivant plusieurs critères [KONE 93]. Ceci pose des problèmes de conflit entre les critères, l'optimum pour tous les critères à la fois étant souvent impossible à atteindre. On cherchera alors un point tel que l'on ne pourra plus améliorer un critère sans détériorer les autres. Une telle solution est dite paréto-optimale. Il est aussi possible de transformer un problème multi-critères en un problème scalaire, soit en considérant un critère primordial et en transformant les autres en contraintes, soit en effectuant une pondération.

1.2.4 Algorithmes d'optimisation

Problème d'optimisation sous contraintes

Plusieurs auteurs ont montré l'équivalence du problème de dimensionnement sous contraintes avec un problème d'optimisation sous contraintes [KONE 93, SALD 88]. Ces contraintes imposées par le cahier des charges doivent être prise en compte par l'algorithme d'optimisation utilisé.

Formulation du problème

On veut parcourir l'espace des solutions, en cherchant à résoudre le problème suivant [DEVA 94]:

$$\begin{cases} min\ f(p) \\ g_i(p) \le 0, \quad i = 1,l \\ g_i(p) = 0, \quad i = l+1,m \\ p\,min_j \le p_j \le p\,max_j, \quad j = 1,k \end{cases} \qquad \text{(EQ I-1)}$$

où :

- p est un vecteur de dimension k, contenant les paramètres de l'espace à explorer (paramètres d'optimisation), qui sont limités entre des valeurs minimales et des valeurs maximales (intervalle de variation),
- f représente la fonction objectif dépendant de ces paramètres,
- g_i sont les fonctions dépendant de ces paramètres et soumises à des contraintes d'inégalité pour i=1,l et d'égalité pour i=l+1,m.

La plupart des méthodes d'optimisations permettant de résoudre ce problème nécessitent un point de départ.

Prise en compte des contraintes

La prise en compte des contraintes peut se faire soit par l'algorithme d'optimisation utilisé, si celui-ci est capable de les gérer, soit par une méthode de transformation.

Les méthodes de transformation permettent de transformer un problème avec contraintes en un problème ou une séquence de problèmes sans contrainte, qui peuvent alors être résolus par des techniques d'optimisation sans contrainte, que l'on trouve couramment. Ce sont les méthodes de pénalité, du Lagrangien Augmenté [POWE 69], ou la méthode des fonctions pénalité extérieure classique, appelée SUMT en anglais (Sequential Unconstrained Minimization Technique) [FIAC 68], par exemple.

Les deux familles d'algorithmes d'optimisation

On distingue deux grandes familles de méthodes d'optimisation, qui se différencient par la méthode utilisée pour se diriger dans l'espace des solutions.

Les algorithmes stochastiques

Le parcours de l'espace des solutions se fait de manière plus ou moins aléatoire, c'est à dire qu'avec le même point de départ et le même cahier des charges, le parcours de l'espace des solution peut être différent pour deux optimisations successives.

Les algorithmes stochastiques les plus courants sont la méthode du recuit simulé [ALOT 96], et les algorithmes génétiques [ÜLER 95, RIBE 94].

Un des avantages de ces algorithmes est qu'ils ne nécessitent pas d'autre connaissance sur le modèle, comme par exemple le calcul des dérivées, que son évaluation. En outre, la convergence de ces algorithmes ne dépend théoriquement pas des conditions initiales choisies, on peut tout à fait en générer aléatoirement. Ils sont normalement capables de ne pas se laisser piéger par un optimum local.

Cependant, la convergence de ce type d'algorithme est lente en général. Appliqué à un modèle numérique déjà coûteux en temps de calcul, le temps nécessaire au dimensionnement peut devenir prohibitif. De plus, les critères de convergences de ces algorithmes ne sont souvent pas rigoureux : ils ne sont pas trompé par les minima locaux mais la précision relative sur le résultat obtenu est inconnue.

Les algorithmes déterministes

Les algorithmes déterministes, comme leur nom l'indique, pour un problème donné, parcourront toujours l'espace des solutions de la même manière. Ils n'ont rien d'aléatoire, et

sont basés sur des propriétés mathématiques. On peut distinguer les méthodes directes et indirectes.

Méthodes directes

Ces méthodes utilisent uniquement les valeurs de la fonction objectif. On peut distinguer les algorithmes heuristiques, construits à partir d'intuitions géométriques empiriques, et les algorithmes à bases théoriques, qui ont un fondement mathématique qui permet d'établir des garanties de performances telle que la convergence.

On peut citer parmi les algorithmes heuristiques, les algorithmes de Hooke and Jeeves [BIAN 95] et de Rosenbrock [KONE 93]. Comme algorithme à base théorique, nous citerons l'algorithme des directions conjuguées de Powell [KONE 93].

Ces méthodes sont utilisées lorsque les seules informations disponibles sont les valeurs de la fonction, comme dans le cas des méthodes stochastiques. On peut élaborer rapidement un logiciel de dimensionnement basé sur un modèle mathématique donné auquel on applique une méthode de ce type [KONE 93]. Cependant, elles demandent souvent un nombre important d'évaluations de la fonction (donc un coût élevé en temps de calcul) pour localiser la solution.

Méthodes indirectes

Ces méthodes utilisent le calcul du gradient de la fonction, pour diriger la recherche dans l'espace des solutions. Ils n'ont donc rien d'aléatoire et "plongent" directement vers un optimum.

Ce sont par exemple la méthode de Cauchy ou de la plus grande pente, la méthode de Newton, la minimisation séquentielle quadratique, les méthodes du gradient conjugué ou de quasi-Newton. Kone [KONE 93] donne la description détaillée de plusieurs de ces méthodes. Leurs principaux avantages sont de converger rapidement et d'obtenir un résultat précis car les critères de convergence sont exacts.

Ils nécessitent un développement plus long que les algorithmes stochastiques et les méthodes directes : il faut en effet calculer en plus les dérivées, qui ne sont pas toujours évidentes à obtenir, spécialement dans le cas de modèles numériques où l'on a recours aux différences finies.

De plus ces algorithmes travaillent en considérant toutes les variables continues, ce qui n'est pas toujours le cas, notamment en ce qui concerne le nombre d'encoches ou de pôles dans les machines électriques.

Le principal inconvénient des méthodes directes est d'être bloquées par les optima locaux. Il faut donc choisir avec particulièrement de soin le point de départ.

Choix d'un algorithme d'optimisation

Suivant le problème à traiter et le type de modèle dont on dispose, l'un ou l'autre type d'algorithme d'optimisation peut sembler plus adapté. Les comparaisons entre eux sont difficiles [RAMA 73, WURTZ 96]. La précision et l'intérêt des algorithmes du type gradient reposent largement sur la précision des dérivées partielles calculées. Ainsi, des articles ont montré la mauvaise précision de ce type d'algorithme en calculant les dérivées partielles par différence finie [SING 92, RAMA 73]. N. Piette [PIET 98] a montré la rapidité des algorithmes de type gradient par rapport aux méthodes directes. D'autres travaux [WURT 96] ont montré tout l'intérêt d'utiliser ces algorithmes avec des modèles analytiques, en calculant automatiquement symboliquement les dérivées partielles.

Des travaux ont déjà étés menés qui effectuent une combinaison des deux types d'algorithmes d'optimisation [DEVA 94], stochastiques et déterministes, et ce, pour bénéficier des avantages des deux en éliminant les inconvénients, comme le blocage à un optimum local. Cette approche semble prometteuse.

II – Dimensionnement à l'aide de modèles analytiques

2.1 Utilisation des modèles analytiques pour le dimensionnement

2.1.1 Intérêt des modèles analytiques pour le concepteur

Nous avons déjà évoqué dans la partie précédente notre intérêt pour les modèles analytiques dans le contexte de la conception en génie électrique. Nous allons préciser ici leurs avantages et limites dans cette optique et justifier ainsi ce choix.

Les modèles analytiques présentent les avantages suivants :

- Ils sont nombreux en génie électrique et sont largement disponibles [BELOT, BOUL 90, ALGE 70, COCH 89]. En effet, c'est le premier matériau historiquement utilisé par les concepteurs, avant l'apparition plus récente des modèles numériques.
- Ils sont moins lourds à exploiter que les modèles numériques. Pour l'analyse, ils sont en général plus rapides en temps de calcul que les modèles numériques, ce qui est intéressant lorsqu'on effectue des itérations lors du dimensionnement, car il permettent d'explorer l'espace des solutions en un minimum de temps.
- Les équations qui les constituent contiennent un lien explicite entre les paramètres du système à concevoir [KONE 93b]. Ceci permet au concepteur de mieux comprendre le

fonctionnement du dispositif et de savoir les interdépendances des paramètres, ce qui peut être utile pour savoir sur quels paramètres agir lors du dimensionnement, même si ce lien peut être difficile à appréhender quand le modèle devient trop complexe.

- Ils permettent de prendre en compte les différents phénomènes qui interviennent dans le système (thermique, mécanique,...) même si ce n'est pas finement et permettent de prendre en compte les couplages entre eux. Ils permettent aussi de tenir compte de paramètres de coût par exemple.

Cependant ils ont certaines limites:

- Ils sont adaptés aux calculs de performances moyennes mais peu aux calculs de phénomènes locaux – ils ne permettent pas, ou difficilement, de modéliser des phénomènes microscopiques comme une saturation dans une dent du stator d'une machine. Ainsi, ils ne sont pas forcément très fins ni très précis mais ils permettent une phase de préconception du dispositif, qui peut être affinée par la suite à l'aide de logiciels plus précis en modélisation (éléments finis).
- Ils ne sont pas génériques. Pour chaque nouvelle structure de dispositif à concevoir, un modèle analytique correspondant doit être développé, et il n'existe pas de démarche générique pour l'obtenir. En outre, il existe souvent de nombreuses modélisations possibles.

Les modèles analytiques sont, de toutes façons, un passage obligé dans la démarche de conception, et notamment dans la phase de préconception, où la structure n'est encore pas totalement figée. Nous avons donc décidé au vu du contexte de conception dans lequel nous nous plaçons d'utiliser des modèles analytiques pour le dimensionnement des dispositifs électrotechniques, qui nous semblent constituer l'information de base pour le concepteur.

Nous allons maintenant examiner plus précisément ce que peut faire et ce que fait le concepteur traditionnellement à l'aide de modèles analytiques et par là même montrer la nécessité d'une méthodologie de conception.

2.1.2 Manipulation des modèles analytiques

Nous allons examiner ici ce que peut faire un concepteur à partir d'un modèle analytique pour dimensionner un dispositif électromagnétique, c'est à dire les manipulations qu'il peut effectuer dessus et les informations qu'il peut en tirer.

Le modèle analytique

Le modèle analytique est constitué d'équations symboliques, et il relie les points de fonctionnement et les performances du systèmes aux paramètres géométriques décrivant la structure du système à concevoir ainsi qu'aux autres paramètres caractérisant le système comme les caractéristiques des matériaux, les grandeurs électriques, etc....

Il est en général la solution d'un problème mathématique plus général à résoudre, comme des équations aux dérivées partielles ou des équations différentielles, que l'on rencontre souvent dans les problèmes électriques.

Par exemple, pour une résistance, on pourrait donner ce modèle :

$$\begin{cases} U = R \times I \\ R = \dfrac{\rho \times L}{S} \\ P = R \times I^2 \\ S = \dfrac{\pi \times D^2}{4} \\ V = L \times S \end{cases} \qquad \text{(EQ I-2)}$$

où R est la résistance, U, la tension électrique appliquée, I, le courant qui la traverse, ρ, la résistivité du matériaux, P, la puissance dissipée, L, la longueur, S, la surface, D, le diamètre et V, le volume total. C'est un modèle très simple, les modèles analytiques peuvent comprendre quelques centaines d'équations et de paramètres.

Calcul d'analyse

Tout d'abord le concepteur doit pouvoir évaluer le modèle, pour un jeu de paramètres donné. C'est une des étapes indispensable dans le dimensionnement. Ceci revient à résoudre le système d'équations constituant le modèle analytique, c'est à dire fixer des paramètres, qui deviennent des entrées, et calculer les paramètres qui en dépendent, qui sont les paramètres de sortie. On travaille donc sur un modèle orienté.

Les modèles analytiques comme on peut les trouver dans la littérature, et dans l'exemple ci-dessus (EQ I-2) possèdent en général une orientation : certains paramètres sont clairement exprimés en fonction des autres. Ainsi dans l'exemple (EQ I-2), P, U, R, S et V s'expriment en fonction de I, ρ, L, et D. Evaluer le modèle revient ici à fixer des valeurs pour ces paramètres pour calculer les premiers.

Cependant, cette orientation n'est pas toujours claire, surtout en phase de développement du modèle, et l'utilisateur peut vouloir la modifier et fixer d'autres paramètres d'entrée, par exemple s'il ne connaît pas la valeur d'un des paramètres d'entrée de départ.

En outre, il peut apparaître des cycles dans le système d'équations constituant le modèle, qui font que l'on ne peut exprimer certains paramètres en fonction des paramètres d'entrée choisis seulement. Ce problème de cycles, que nous appelons systèmes implicites, sera étudié en détail dans le Chapitre 3.

Propagation de contraintes

L'évaluation du modèle est une part du dimensionnement, notamment lorsqu'on utilise une méthode d'optimisation, mais le calcul d'un modèle n'implique pas que les paramètres appartiennent au domaine de solutions pour le cahier des charges posé. En outre, les méthodes d'optimisation nécessitent en général un point de départ cohérent, c'est à dire appartenant à cet espace des solutions.

La propagation de contrainte correspond à un parcours de l'espace des solutions par tâtonnement. Elle consiste à fixer certains paramètres petit à petit et à en déduire les autres. Au fur et à mesure, on vérifie que les paramètres obtenus respectent le cahier de charges, et si ce n'est pas le cas, on revient en arrière en modifiant les paramètres d'entrée dans le sens nécessaire. Il est clair que même en effectuant des itérations et des retours en arrière, on ne peut explorer tout l'espace et qu'on obtient une solution qui n'est pas optimale.

Souvent, on utilise un modèle simplifié du système et on fixe a priori un certain nombre de paramètres pour réduire l'espace des solutions à explorer.

Cette approche mène à la détermination d'un processus de dimensionnement précis pour le modèle de la structure donnée, réutilisable pour redimensionner le même dispositif avec le même type de cahier des charges, mais qui est à réitérer pour un autre modèle. C'est ce qu'on a nommé plus haut les méthodes de dimensionnement procédurales (cf. Chapitre 1 § 1.2.3).

Dimensionnement par méthode d'optimisation

Le dimensionnement par méthode d'optimisation consiste à utiliser une boucle itérative contenant une phase de calcul d'analyse du modèle et une phase de passage dans un algorithme d'optimisation. L'optimisation, comme le calcul d'analyse, se fait donc sur un modèle orienté.

Mais la méthode d'optimisation peut être difficile à mettre en œuvre lorsque le modèle est complexe, ou fait intervenir de nombreux paramètres à optimiser.

Souvent, on n'effectue le dimensionnement qu'en faisant varier deux ou trois paramètres bien choisis, car on ne dispose pas d'outil de conception spécifique pour manipuler tout le modèle.

Par exemple dans l'article de W. Baran [BARA 83] qui concerne un accouplement magnétique, on laisse varier les dimensions des aimants et le nombre de pôles. On effectue une série d'évaluations du modèle avec différentes valeurs de ces paramètres, et on en déduit un dimensionnement optimal parmi ceux calculés. On aurait pu effectuer l'optimisation en ne faisant varier continûment que ces paramètres. L'intérêt est de travailler sur un espace des solutions très réduit, moins difficile et moins long à explorer.

Ainsi l'utilisation des algorithmes d'optimisation déterministes est elle aussi souvent exclue, car elle nécessite le calcul des gradients, bien souvent impossible sans outil d'automatisation, alors qu'ils peuvent être très performants avec un modèle analytique, en général dérivable symboliquement.

2.1.3 Intérêt des outils d'aide à la conception

Les méthodes usuelles pour le concepteur d'utilisation des modèles analytiques pour le dimensionnement montrent leurs limites. Il apparaît clairement un besoin d'outils dédiés.

En effet, sans outil d'aide à la conception, le dimensionnement est difficile à optimiser. L'idéal est de pouvoir faire varier tous les paramètres non fixes. Sans outil, ceci n'est possible que si l'on utilise des méthodes d'optimisation stochastiques ou directes [KONE 93] car l'évaluation des dérivées est trop complexe pour être effectuée.

En outre, même la simple évaluation du modèle peut être difficile à réaliser sans outil adéquat. Les modèles de l'électronique analogique par exemple, basés sur le calcul des fonctions de transfert, deviennent très vite impossibles à gérer avec plus d'une dizaine d'interrupteurs [WAMB 96]. On est alors obligé de tronquer le modèle et sa lisibilité – dépendance entre les paramètres – diminue beaucoup. Les "gros" modèles analytiques sont difficiles à manipuler pour le concepteur.

Il faut ajouter à cela que le problème inverse n'est pas facile à résoudre sans outils. Le modèle ne peut être en général calculé que dans le sens donné par l'orientation des équations. Si on modifie le jeu de paramètres d'entrées, on risque de se retrouver avec des systèmes implicites à résoudre.

De plus, le concepteur n'exploite pas dans les modèles analytiques la possibilité du calcul de sensibilité systématique, idée qui a été développée par F Wurtz dans la méthodologie Pascosma [WURT 96], pour l'optimisation ou même seulement pour la

connaissance de la sensibilité d'un paramètre par rapport à un autre qui peut aider le concepteur à diriger sa démarche de dimensionnement, et qui est rapide et exact.

Il faut donc des outils pour manipuler le modèle et ce, de façon optimale. L'outil doit donc aider le concepteur dans le processus de dimensionnement. Ainsi, ce type d'outil doit permettre une manipulation aisée de l'information nécessaire pour le concepteur, le guider dans ses choix, et l'aider dans l'élaboration de la stratégie de dimensionnement. En soulageant le concepteur des calculs fastidieux et source d'erreurs, qui doivent être automatisés, il doit permettre un gain de temps et une amélioration de la qualité de la solution obtenue. Des manipulations jusqu'alors impossibles "à la main" peuvent ouvrir de nouvelles possibilités pour le concepteur.

Les outils dédiés devraient permettre les manipulations suivantes sur les modèles analytiques :

- L'évaluation du modèle directe, sans avoir à simplifier le modèle et en intégrant le problème de systèmes d'équations à résoudre, c'est à dire de cycles dans le modèle.

  Cette fonctionnalité entre en jeu dans la phase d'analyse.
- Le calcul de sensibilité juste, en automatisant ce calcul, car il est source d'erreurs lorsqu'il est effectué "à la main"

  Cette fonctionnalité entre en jeu dans phase d'optimisation.
- La propagation de contraintes, c'est à dire de pouvoir fixer n'importe quel jeu de paramètres et d'essayer d'en déduire les autres. Il faudrait donc intégrer la possibilité d'inverser symboliquement une équation.

  Cette fonctionnalité entre en jeu à la fois dans la phase de calcul et dans celle d'optimisation, car elle permet à la fois de calculer le modèle, en l'orientant, mais permet aussi d'explorer l'espace des solutions.

2.2 Outils et méthodologies de dimensionnement à l'aide de modèles analytiques

2.2.1 Les outils existants

Des recherches ont déjà été menées autour de logiciels utilisant les modèles analytiques de façon générique, c'est à dire qu'ils permettent d'utiliser des modèles différents, en laissant le traitement symbolique ou numérique à l'ordinateur et offrent ainsi des outils pour le concepteur.

Les outils que nous avons pu trouver dans la littérature n'intègrent souvent pas, dans leur approche de la manipulation des modèles analytiques pour le dimensionnement, le problème de l'optimisation. Ils n'intègrent pas en général non plus le calcul de sensibilité.

CADACS

Cette dernière remarque s'applique à CADACS [SALD 88], qui est un logiciel de manipulation des équations algébriques orienté objet, et qui s'intéresse surtout au problème de propagation de contraintes dans le processus de conception. Son approche intéressante est de coder les équations du modèles sous forme d'objets, ce qui permet une grande souplesse dans le parcours du graphe ainsi créé, puisqu'il peut être parcouru dans tous les sens. Il permet d'enregistrer l'enchaînement des équations résolues et donc le processus de dimensionnement du modèle, ce qui est très pédagogique. Saldahna s'est intéressé au problème de résolution symbolique des systèmes d'équations mais n'intègre pas la résolution numérique des cycles rencontrés dans le cas où la résolution symbolique est impossible. Ce travail s'intégrait dans un projet d'environnement de conception, avec en amont la génération, par un système à base de connaissances du modèle analytique à traiter [SALD 87].

TK!Solver

On peut aussi citer un logiciel très utilisé dans l'industrie, TK!Solver [KONO 84], qui s'occupe de la résolution des modèles analytiques, en intégrant la résolution numérique et de nombreuses fonctionnalités pratiques, comme la possibilité d'utiliser des tables et d'effectuer des interpolations de fonctions empiriques. Il intègre aussi un mécanisme de propagation de contraintes, moins abouti que celui de CADACS, c'est à dire qu'il permet de propager les valeurs affectées, mais on ne peut pas vraiment parler ici d'outil de dimensionnement. Il est lui aussi plus ou moins orienté objet de par son implémentation en Lisp, qui lui permet de manipuler les listes – et donc les tables – aisément.

Un de ses grands intérêt est d'intégrer grâce à cela la notion de paramètre discret (qui prend ses valeurs dans une table). Un autre est de prendre en compte les complexes, sous la forme d'un couple de réels. Il possède aussi des librairies de modèles de problèmes connus et souvent rencontrés dans différents domaines de la physique et notamment des modèles de génie électrique.

Cependant, sa vocation première est de résoudre le système d'équations constituant le modèle analytique. C'est un outil d'analyse.

DONALD

DONALD [SWIN 91] est un outil développé dans le domaine de l'électronique. Il a pour but de manipuler les modèles analytiques des fonctions de transferts symboliques des circuits analogiques. Mais il peut tout aussi bien manipuler n'importe quel modèle analytique. Il intègre la propagation de contrainte sur ces modèles et grâce à son codage sous forme de graphe bipartite, orienté objet, il permet de parcourir celui-ci dans n'importe quel sens. Par rapport à CADACS, il permet en plus de résoudre les systèmes numériquement. Il n'effectue pas le calcul de sensibilité mais donne la dépendance des paramètres les uns par rapport aux autres.

En ce qui concerne le problème de l'optimisation, et donc d'un dimensionnement intéressant, aucun des outils cités ci-dessus, y compris DONALD, n'en intègre la possibilité. Cependant DONALD vise à s'intégrer dans un environnement de conception avec d'autres outils développés par les mêmes auteurs, notamment un outil d'optimisation nommé OPTIMAN [GIEL 90] utilisant la méthode stochastique du recuit simulé. Il n'exploite donc pas l'intérêt des dérivées partielles justes que l'on peut obtenir avec un modèle analytique.

Pascosma

Les travaux effectués par F. Wurtz [WURT 96] ont donné naissance à la méthodologie et à l'outil de dimensionnement Pascosma, pour Programme d'Analyse, de Synthèse, de Conception et d'Optimisation de Systèmes Modélisables Analytiquement. Comme son nom l'indique, cet outil utilise le modèle analytique d'un dispositif à concevoir pour son dimensionnement. L'algorithme d'optimisation utilisé est du type gradient et la génération des dérivées symboliques est effectuée de manière automatique. Cet outil offre donc des facilités pour la manipulation automatique des modèles, pour la programmation de ceux-ci, mais aussi pour le calcul formellement exact des dérivées partielles.

Cet outil permet le calcul du modèle seulement selon l'orientation de départ des équations constituant celui-ci. On ne peut modifier l'orientation du modèle de manière automatique. La propagation de contraintes n'y est donc pas possible.

Ainsi, si la plupart des outils existants se sont intéressés au problème d'aide à la conception en utilisant des modèles analytiques, il nous semble que toutes les possibilités de ces modèles ne sont pas exploitées dans chacun des cas. Il nous paraît intéressant de pouvoir offrir au concepteur un outil polyvalent, lui permettant d'effectuer toutes les tâches fastidieuses à la main, mais lui offrant aussi de nouvelles possibilités grâce aux progrès de la

technologie informatique. La nécessité d'une méthodologie de dimensionnement unificatrice se fait jour.

2.2.2 Intérêt d'une méthodologie unificatrice

Le dimensionnement des systèmes électromagnétiques nécessite une méthodologie de conception pour exploiter au mieux les possibilités des modèles analytiques, que nous considérons comme une des informations de "base" pour le concepteur.

Fonctionnalités

Nous voulons offrir un outil permettant au concepteur d'effectuer la propagation de contraintes, l'évaluation du modèle et l'optimisation – sous contraintes – à partir d'un modèle analytique.

Comme nous l'avons vu précédemment, aucun outil existant ne permet de faire tout le travail de dimensionnement à l'aide de modèles analytiques. En outre, la communication entre ces différents logiciels n'est souvent pas intégrée. Il serait donc intéressant de pouvoir effectuer ces tâches au sein d'un outil unique, plate-forme de développement pour le concepteur.

Pour ce faire, l'outil doit intégrer et automatiser les différentes fonctionnalités que nous avons spécifiées au Chapitre 1 § 2.1.3, à savoir :

- L'évaluation directe du modèle, en intégrant le problème de cycles dans le modèle, et ce, selon différentes orientations.
- Le calcul de sensibilité juste, en automatisant ce calcul.
- La propagation de contraintes.

Il faut ajouter à cela la possibilité de lier un algorithme d'optimisation, de type gradient ou non pour effectuer l'optimisation.

Modularité et extensibilité

Cependant, notre outil doit rester assez ouvert pour pouvoir intégrer d'autres outils d'analyse et/ou d'optimisation, et pouvoir communiquer avec eux. Comme nous l'avons signalé plus haut, le dimensionnement effectué à l'aide d'un modèle analytique peut être affiné par la suite à l'aide d'un modèle plus précis, de type numérique. Il faut donc prévoir de pouvoir transmettre les données sur les paramètres à l'extérieur. On veut donc un outil pilotable et qui permette le dialogue.

Il doit être modulaire, et manipuler des concepts génériques, qui pourront ensuite être déclinés :

- modularité des objets manipulés : on pourra intégrer autre chose que des équations
- modularité des algorithmes utilisés, à la fois pour la résolution des systèmes d'équations que l'on peut avoir à traiter et pour l'optimisation (on peut même envisager de pouvoir intégrer des algorithmes stochastiques même si ce n'est pas le but premier, afin d'offrir un maximum de possibilités au concepteur)

L'outil doit aussi être extensible. Il s'appuie certes sur l'information contenue dans un modèle analytique, mais ne doit pas empêcher le concepteur d'intégrer des objets qui ne sont pas des équations analytiques. L'environnement prenant en charge tous les aspect liés aux modèles analytiques sert pour nous de base à un environnement plus polyvalent, où le concepteur, moyennant un certain travail de sa part, pourra intégrer d'autres objets comme des morceaux de modèle numérique.

Conclusions

Dans le contexte de la conception en génie électrique, nous nous sommes placés dans la phase de dimensionnement d'une structure donnée, ce qui est le travail le plus courant du concepteur. Nous avons choisi d'effectuer ce dimensionnement à l'aide de modèles analytiques car ils nous semblent contenir de nombreuses informations pour la conception et, en outre, leur exploitation est moins lourde et plus rapide que celle des modèles numériques en général. Nous avons montré, sur cette base, la nécessité d'une méthodologie de conception et d'outils adaptés pour l'aide à la conception.

En examinant les outils existants, qui utilisent des modèles analytiques pour la conception, une approche unificatrice de ces méthodologies nous a semblé un objectif intéressant à atteindre. Il s'agit d'offrir au concepteur le moyen d'utiliser au mieux l'information contenue dans le modèle analytique dont il dispose pour le dimensionnement et ce, en créant un environnement permettant d'intégrer au mieux toutes les fonctionnalités liées à ces modèles, tout en restant ouvert à l'intégration d'objets et de méthodes extérieurs.

Pour spécifier au mieux cet outil, nous avons basé notre travail sur l'examen d'un des outils déjà existant, que nous avons présenté : la méthodologie Pascosma, afin d'en extraire les limites et problèmes inhérents à la manipulation de modèles analytiques. C'est cette étude que nous trouvons au Chapitre 2.

Chapitre 2. Un exemple de dimensionnement avec la méthodologie Pascosma : limites mises en évidence

Au travers de la présentation d'un exemple de concret de modèle analytique, celui d'un accouplement magnétique, et de son dimensionnement à l'aide d'un des outils présentés dans le Chapitre 1 § 2.2.1, la méthodologie Pascosma, nous allons extraire les difficultés soulevées par la manipulation des modèles analytiques, en soulignant les limites rencontrées par l'approche étudiée. Cet outil nous a semblé intéressant à étudier en détails car il est le seul qui mette en œuvre une méthode d'optimisation de type gradient pour le dimensionnement, en effectuant la génération automatique des dérivées partielles symboliques. Dans un premier temps, nous présenterons Pascosma, en nous appuyant sur l'exemple d'un accouplement magnétique, puis nous mettrons en avant les apports et limites de cette méthodologie pour l'exemple considérés, mais aussi à travers d'autres exemples que nous avons pu étudier.

I - Pascosma : une méthodologie d'optimisation sous contraintes

La méthodologie Pascosma se propose de traiter le dimensionnement sous contraintes de machines électriques à l'aide d'un modèle analytique, en s'appuyant sur une méthode d'optimisation de type gradient. Il se place dans la même problématique que notre approche et constitue une méthodologie mettant en œuvre l'optimisation intéressante à étudier. Après avoir étudié les problèmes soulevés par l'optimisation sous contraintes, nous allons présenter la structure et le fonctionnement de la méthodologie Pascosma.

1.1 Méthodologie d'optimisation sous contraintes de modèles analytiques

Comme nous l'avons vu au Chapitre 1 § 1.2.4, le problème de dimensionnement sous contraintes est équivalent à un problème d'optimisation contrainte [KONE 93b]. Nous avons vu la formulation générique de ce problème. Nous allons maintenant l'étudier plus en détails dans le cas de modèles analytiques.

1.1.1 Problème d'optimisation contrainte

Dans le cas d'un modèle analytique orienté, comme ceux utilisés dans Pascosma, le problème de dimensionnement sous contraintes est décrit par un ensemble d'équations. Le modèle est décrit par :

$$Ps_i = f_i(Pe_1, Pe_2, \cdots, Pe_n), \quad i = 1, m \qquad \text{(EQ II-1)}$$

où f_i, i=1,m sont les fonctions mathématiques reliant les paramètres de sortie Ps_i, i=1,m aux paramètres d'entrée Pe_j, j=1,n. Chaque variable est fixée ou contrainte à appartenir à un certain intervalle :

$$Pe\min_f \leq Pe_f \leq Pe\max_f \quad f = 1, q \qquad \text{(EQ II-2)}$$

$$Pe_e = Peconst_e \quad e = 1, q+1, n \qquad \text{(EQ II-3)}$$

$$Ps\,min_g \leq Ps_g \leq Ps\,max_g \quad g = 1, v \qquad \text{(EQ II-4)}$$

$$Ps_h = Psconst_h \quad h = v+1, m \qquad \text{(EQ II-5)}$$

où q est le nombre de contraintes d'inégalité et (n-q), le nombre de contraintes d'égalité sur les paramètres d'entrée, et v est le nombre de contraintes d'inégalité et (m-v), le nombre de contraintes d'égalité sur les paramètres de sortie.

Pour diriger la recherche dans l'espace des solutions et se limiter à la solution la plus intéressante, on ajoute une fonction objectif fob qui va être minimisée durant l'optimisation. C'est une fonction mathématique de même nature que les fonctions f_i, i=1,m. Elle dépend donc aussi des paramètres d'entrée.

Le problème contraint peut alors être formulé comme suit : Trouver les paramètres d'entrée Pe_j, j=1,n, en sorte que les contraintes (EQ II-2), (EQ II-3), (EQ II-4) et (EQ II-5) soient satisfaites et que la fonction objectif fob(Pe_1,Pe_2,...,Pe_n) soit minimisée.

Ici, les fonctions f_i, i=1,m, ainsi que la fonction objectif, sont analytiques.

On remarque que l'on sépare les paramètres du système en deux catégories : les paramètres de sortie et les paramètres d'entrées, en fonction desquels s'expriment les paramètres de sortie, c'est à dire que le modèle analytique est orienté.

1.1.2 Algorithme d'optimisation

La formulation d'un problème d'optimisation contrainte est résoluble à l'aide d'algorithmes numériques.

Les fonctions étant analytiques, les expressions des dérivées des paramètres de sortie par rapport aux paramètres d'entrée peuvent être fournies par des logiciels de calcul symbolique et calculées de la même manière que les fonctions f_i, i=1,m. Ces dérivées donnent au concepteur la sensibilité du système d'une manière très précise.

Ainsi l'utilisation d'algorithmes d'optimisation de type gradient semble assez « naturelle » dans cette problématique, et ce, pour trois raisons :

- comme ils utilisent la connaissance des dérivées partielles pour trouver une direction de recherche, ils doivent théoriquement se montrer plus performant en termes de rapidité et de précision de convergence [WURT 96],
- ils emploient des critères rigoureux d'optimalité [KONE 93],
- on peut disposer des dérivées symboliques exactes des paramètres de sortie, ce qui permet de profiter de toute la puissance des algorithmes du type gradient (Chapitre 1 § 1.2.4).

Cependant, cette méthode implique le calcul des dérivées des paramètres de sortie, qui est impossible à effectuer, avec la plupart des modèles analytiques utilisés, sans outil informatique dédié.

La prise en compte des contraintes peut se faire par une méthode de pénalité (Chapitre 1 § 1.2.4), ou par l'algorithme d'optimisation lui-même s'il est adéquat.

Cette méthode de dimensionnement est applicable à tout modèle analytique, du moment que l'on dispose d'un outil pour effectuer les calculs et les calculs des dérivées partielles. La méthodologie Pascosma offre ce service. Elle formalise les étapes de la méthode et crée les liens entre modèle, cahier des charges, calculs et algorithme d'optimisation. Elle n'est cependant qu'un choix d'implantation spécifique de cette méthode, et n'est donc pas forcément la meilleure méthodologie d'optimisation.

1.2 Implantation de la méthodologie Pascosma

Nous allons décrire l'implantation de la méthodologie d'optimisation dans Pascosma.

1.2.1 Structure de Pascosma

La méthodologie Pascosma se propose d'assurer la génération automatique d'un logiciel de dimensionnement convivial, à partir d'un modèle analytique de la machine à concevoir. Cette approche se scinde en deux niveaux :

1er niveau – Le cadre méthodologique qui assure la génération du logiciel de dimensionnement. C'est cette partie qui nous intéresse le plus au niveau méthodologique. Nous allons l'étudier plus en détails par la suite.

2ème niveau – Le cadre méthodologique qui met en place les mécanismes génériques assurant une utilisation souple et conviviale de ce logiciel, c'est à dire:

- la souplesse : ici la possibilité de modifier le cahier des charges posé sur l'ensemble des paramètres d'entrée et de sortie, que ce soit par exemple modifier les valeurs minimales et maximales des intervalles des contraintes d'inégalité, ou transformer une contrainte d'égalité en contrainte d'inégalité (Fig. II-1).

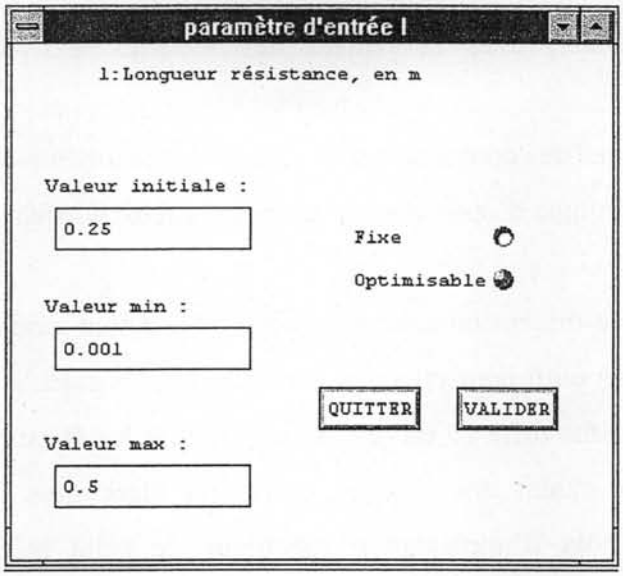


Fig. II-1 : Une fenêtre de l'interface utilisateur de Pascosma permettant de modifier la contrainte sur un paramètre d'entrée

- l'utilisation transparente pour l'utilisateur de l'algorithme d'optimisation. L'outil Pascosma se charge des procédures de normalisation, de tri, de reformulation du cahier des charge en vue du passage dans l'algorithme d'optimisation.
- la convivialité à l'aide d'une interface simple et fonctionnelle, qui comporte une aide et qui est autoconfigurable en fonction du modèle analytique implanté. (Fig. II-2)

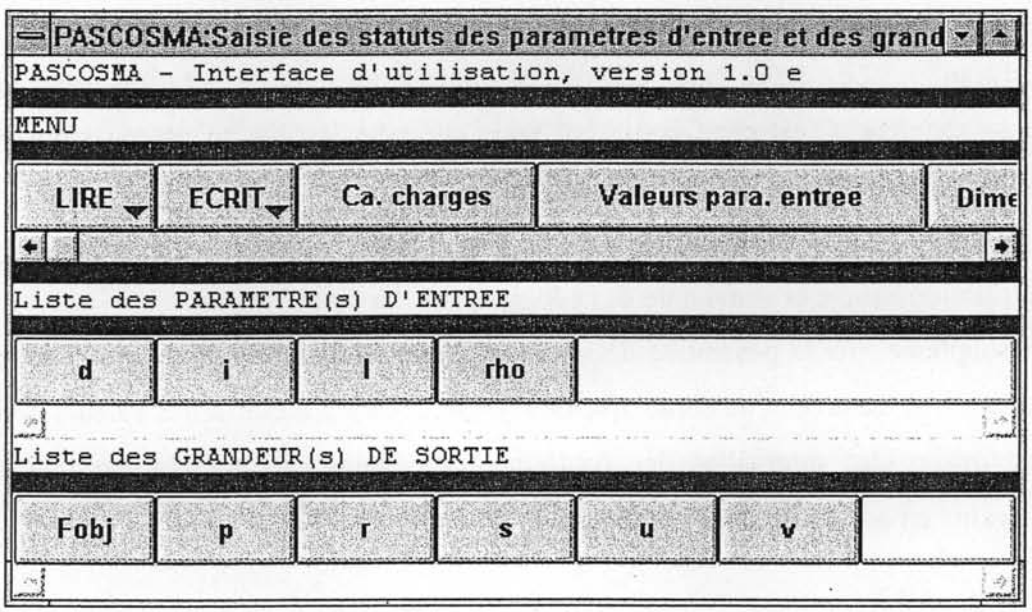


Fig. II-2 : Fenêtre principale de l'interface graphique utilisateur autoconfigurable de Pascosma

1.2.2 Génération du logiciel de dimensionnement

Nous allons décrire plus précisément les sept étapes que comprend la méthodologie Pascosma pour la génération automatique du logiciel de dimensionnement, et notamment pour se ramener à la formulation du problème d'optimisation sous contraintes donnée ci-dessus :

étape 1 : *Saisie des équations du modèle analytique de la machine à dimensionner.*

Cette saisie se fait dans un formalisme précis. On doit donner l'expression d'une variable en fonction d'autres variables. En effet, la méthodologie d'optimisation s'appuie sur un modèle analytique orienté.

exemple : $u = r \times i$

$p = u \times i$

étape 2 : *Recherche des paramètres d'entrée et de sortie du modèle.*

En reprenant les notations du paragraphe précédent, on va chercher quels sont les paramètres Pe_i, i=1,n et les paramètres Ps_j, j=1,m. Les paramètres de sortie Ps_j sont ceux qui apparaissent à gauche dans les équations du modèle entré (u et p dans l'exemple précédent), les paramètres d'entrée Pe_i, sont tous les autres (r et i dans l'exemple précédent).

étape 3 : *Calcul des expressions symboliques des paramètres de sortie et de la fonction objectif en fonction des paramètres d'entrée.*

Dans cette étape, on exprime les paramètres de sortie en fonction uniquement des paramètres d'entrée, en remplaçant dans les expressions de ces paramètres les autres paramètres de sorties qui interviennent par leur expression. On obtient alors la formulation donnée dans l'équation (EQ II-1).

exemple : si on reprend l'exemple ci-dessus, on va remplacer le paramètre u par son expression dans l'expression de p et on obtiendra le nouveau modèle :

$u = r \times i$

$p = r \times i^2$

étape 4 : *Programmation automatique des paramètres de sortie et de la fonction objectif.*

Les expressions symboliques des paramètres de sortie trouvées dans l'étape précédente sont automatiquement programmées sous forme de routines FORTRAN grâce au logiciel Macsyma [MACSY], qui possède un module de traduction en FORTRAN. Le programme d'analyse (évaluation du modèle) est donc généré.

étape 5 : *Calcul symbolique des dérivées partielles.*

Cette étape est elle aussi effectuée grâce aux possibilités de Macsyma, qui permet la dérivation symbolique. C'est une opération complexe et longue, quasiment impossible à

effectuer "à la main" pour des modèles élaborés. L'automatisation permet d'obtenir des expressions de dérivées sans erreurs de calcul.

étape 6 : *Programmation automatique des dérivées partielles.*

Elle se fait de la même manière que la programmation automatiques des paramètres de sortie. Le programme de calcul de sensibilité est ainsi généré.

étape 7 : *Edition de liens avec un algorithme d'optimisation*

Les programmes d'analyse et de calcul de sensibilité générés lors des étapes 4 et 6 sont couplés avec l'algorithme d'optimisation sous contraintes choisi, par un compilateur FORTRAN classique. On obtient alors le logiciel de dimensionnement automatique.

L'algorithme utilisé est l'algorithme de minimisation séquentielle quadratique VF13AD, issu de la bibliothèque Harwell [HARWE].

Un des intérêts principaux de la méthodologie PASCOSMA est donc de permettre la détermination et l'utilisation dans un algorithme d'optimisation de type gradient des expressions symboliques des dérivées partielles, même avec un modèle complexe comportant de nombreux paramètres d'entrée et de sortie. Ces expressions sont en général impossibles à obtenir "à la main". On évite ainsi l'évaluation numérique des dérivées, qui donne de moins bons résultats dans la boucle d'optimisation [SING 92]. En outre, le calcul de dimensionnement s'effectue rapidement car l'évaluation des expressions symboliques est rapide et le logiciel de dimensionnement s'obtient assez rapidement lui aussi, et de façon automatisée.

II – Exemple de dimensionnement : accouplement magnétique

2.1 Présentation de l'application

Afin de présenter les aspects méthodologiques de Pascosma au mieux, et de mettre en évidence des problèmes soulevés par la manipulation de modèles analytiques pour le dimensionnement, nous allons présenter une application : un accouplement magnétique.

2.1.1 Accouplement magnétique synchrone coaxial à aimants permanents

Les accouplements magnétiques à aimants permanents permettent de transmettre un couple entre les deux parties de l'accouplement : l'une entraînant l'autre. Ils peuvent être utilisés dans les enceintes contenant des liquides ou des gaz et transmettre le couple à travers un mur de séparation. Ils peuvent aussi servir à protéger des surcharges ou transmettre les mesures de capteurs depuis des enceintes sous vide ou à haute pression [FELL 79].

L'accouplement magnétique étudié est un accouplement synchrone à aimants permanents, à pôles multiples (Fig. II-3). Les deux parties de l'accouplement tournent à la même vitesse. Le couple transmis est indépendant de la vitesse de rotation, il ne dépend que de l'angle de déplacement entre les deux parties de l'accouplement – θ sur la figure II-3.

Plus précisément, l'accouplement synchrone étudié est de type coaxial : les deux moitiés de l'accouplement sont disposées de manière concentrique. L'instabilité radiale est importante, mais dans la position centrée, les forces radiale et axiale sont nulles. Ce genre d'accouplement est utilisé en présence de cloisons séparatrices cylindriques et il permet de transmettre des couples importants.

Les dimensions de l'accouplement sont définies par 9 paramètres géométriques :

- Rm le rayon moyen
- e l'entrefer
- αp l'épanouissement polaire – donné par αp=π/npp, avec npp, nombre de paires de pôles
- β1 et β2 largeur des aimants
- ea1 et ea2 épaisseurs des aimants
- ef1 et ef2 épaisseurs des culasses en fer

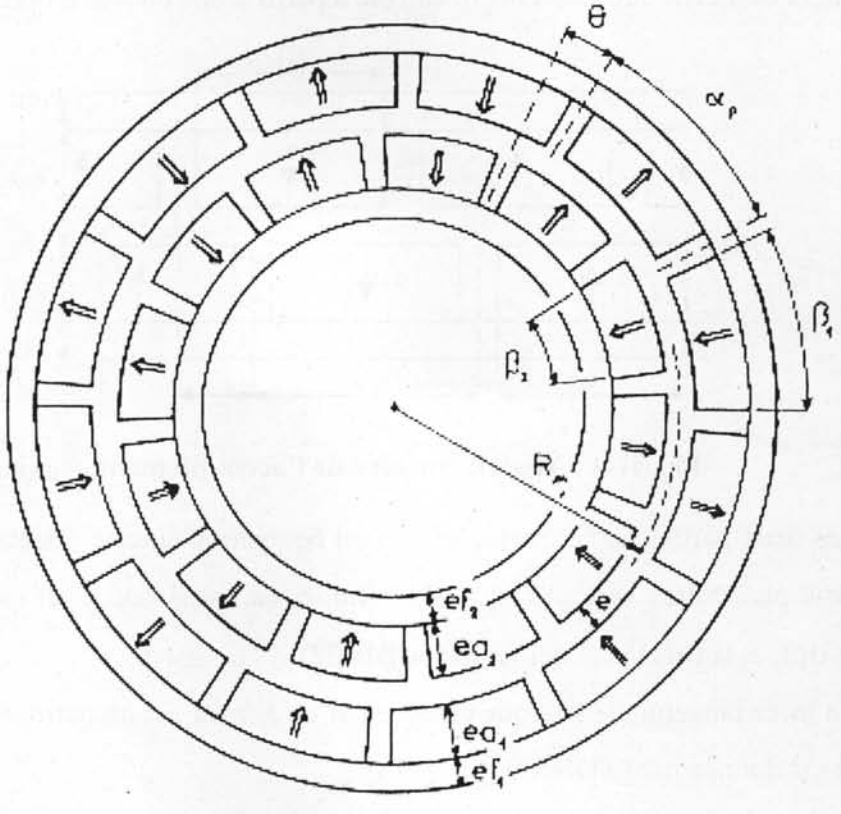


Fig. II-3 : Section de l'accouplement coaxial étudié

- L la longueur axiale de l'accouplement

Pour décrire totalement le système, on doit ajouter à ces paramètre des paramètres magnétiques :

- J1 et J2 la magnétisation de chaque moitié de l'accouplement (représentée par les flèches sur le Figure II-3)

2.1.2 Modèle analytique de l'accouplement implanté dans Pascosma

Récemment, les formules de calcul de force entre deux aimants développées, que ce soit en deux ou en trois dimensions [YONN 92, AKOU 84], ont permis de construire des modèles analytiques adaptés au calcul des accouplements magnétiques.

Présentation du modèle

Le modèle implanté dans Pascosma est un modèle 2D, basé sur le calcul de l'interaction entre deux aimants face à face pour un modèle linéaire (Fig II-4) [YONN 98]. L'effet de la culasse est pris en compte par l'application du théorème des images magnétiques équivalentes aux aimants, les culasses étant considérées comme des équipotentielles magnétiques. Pour prendre en compte la courbure, qui induit une diminution de la force, on applique un coefficient représentatif η, calculé à partir d'une courbe empirique.

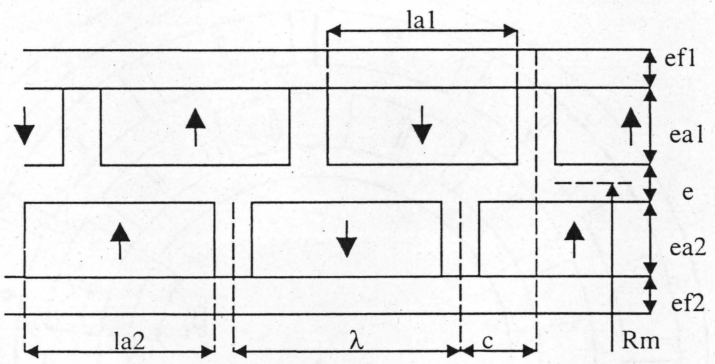


Fig. II-4 : Modèle linéaire de l'accouplement coaxial

Les deux parties de l'accouplement étant équipées d'aimants identiques, le système est décrit par 8 paramètres en tout (Fig II-4) : Rm, e, ea (=ea1=ea2), ef (=ef1=ef2), L, λ (qui remplace αp), et la (=la1=la2 qui remplace $\beta 1=\beta 2$).

La force tangentielle linéique exercée par un aimant sur un autre, parallèle et identique (Fig II-5) est donnée par [YONN 92] :

$$F(c,d)=\frac{f(c,d)}{L}=2\cdot g(c,d)-g(c,d-2b)-g(c,d+2b) \qquad \text{(EQ II-6)}$$

avec :

$$g(x,y)=\frac{J^2}{4\pi\mu_0}\cdot\left\{(x+2a)\cdot ln((x+2a)^2+y^2)\right.$$
$$-2x\,ln(x^2+y^2)+(x-2a)\cdot ln((x-2a)^2+y^2) \qquad \text{(EQ II-7)}$$
$$\left.+2y\left(Arctg\left(\frac{x+2a}{y}\right)-2Arctg\left(\frac{x}{y}\right)+Arctg\left(\frac{x-2a}{y}\right)\right)\right\}$$

Cette force est importante pour les petits déplacements mais devient négligeable pour un déplacement de trois ou quatre fois la largeur de l'aimant. Ainsi, la force totale exercée sur un aimant dans le modèle linéaire peut être obtenue très précisément en considérant cinq aimants de part et d'autre de l'aimant considéré, c'est à dire en effectuant une somme de onze termes. Cette force F (EQ II-6) dépend de la distance entre les centres des aimants. Sur la Figure II-5, l'abscisse c correspond à l'angle de déplacement relatif des deux parties de l'accouplement (θ sur la Figure II-3). F est maximale pour c=λ/2.

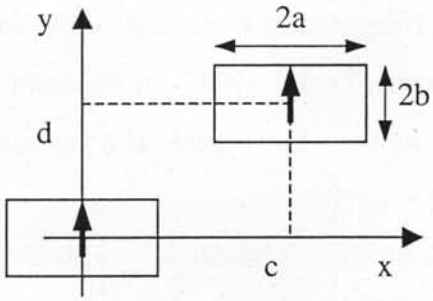


Fig. II-5 : Interaction entre deux aimants élémentaires identiques

Les équations principales du modèle analytique de l'accouplement magnétique sont :

$$\lambda=\frac{\pi\cdot Rm}{npp}, \qquad \text{(EQ II-8)}$$

$$d=e+2\cdot ea, \qquad \text{(EQ II-9)}$$

$$\eta=1-\frac{0.18}{npp}-\frac{1.2036}{npp^2}, \qquad \text{(EQ II-10)}$$

$$\Gamma max=(2\cdot npp\cdot Rm)\cdot F\left(-\frac{\lambda}{2},d\right)\cdot\eta\cdot L, \qquad \text{(EQ II-11)}$$

$$mv/\Gamma=\frac{(4\cdot la\cdot ea\cdot L\cdot npp)}{\Gamma max}, \qquad \text{(EQ II-12)}$$

où Γmax est le couple maximal, et mv correspond à la masse d'aimants.

On a ajouté à ce modèle une équation donnant le rayon intérieur de l'accouplement Rint, qui doit être contraint à une valeur minimale, en fonction de l'épaisseur minimale de la culasse :

$$Rint = Rm - \frac{e}{2} - ea - ef \qquad \text{(EQ II-13)}$$

et l'épaisseur minimale de la culasse ef est donnée par :

$$ef = \frac{ea \cdot la \cdot J}{B\,max \cdot (2ea + e)} \qquad \text{(EQ II-14)}$$

où Bmax est l'induction maximale admise dans la culasse et est imposée à 1.5T.

Nous cherchons ici à maximiser le couple massique, Γ/mv, c'est à dire) minimiser la fonction objectif mv/Γ, donnée par l'équation (EQ II-12).

Validation du modèle

Le modèle analytique utilisé a été validé par comparaison avec les valeurs de couple obtenues pour un accouplement identique, avec d'autres modèles analytiques.

Les dimensions de l'accouplement sont : Rm=100 mm, e=2 mm, npp=10, L=3 m, la=31 mm, ea=10 mm, J=1 T (magnétisation correspondant aux aimants KOERMAX 22). Pour ces valeurs, le couple maximal Γmax calculé par Pascosma est égal à 33.04 kN.m, et le rapport Γ/mv vaut 0.9 10^6 N/m². La Figure II-6 montre la géométrie initiale de l'accouplement étudié.

	Notre modèle	Modèle 1	Modèle 2	Modèle 3
Γmax (N.m)	33043	34656	33653	33447
Γ/mv (10^6 N/m²)	0.9179	0.9316	0.9046	0.8991

Tableau II-1 : Comparaison du modèle analytique utilisé dans Pascosma avec d'autres modèles [BARA 83]

La comparaison de notre modèle avec trois modèles utilisés dans [BARA 83] est donnée dans le Tableau II-1. On voit que le modèle utilisé dans Pascosma donne des valeurs très proches de celles obtenues pour la même géométrie avec ces modèles.

En outre, une vérification avec le logiciel éléments finis FLUX2D [FLUX2D] donne une valeur de 33.9 kN.m pour le couple maximal, ce qui corrobore nos résultats.

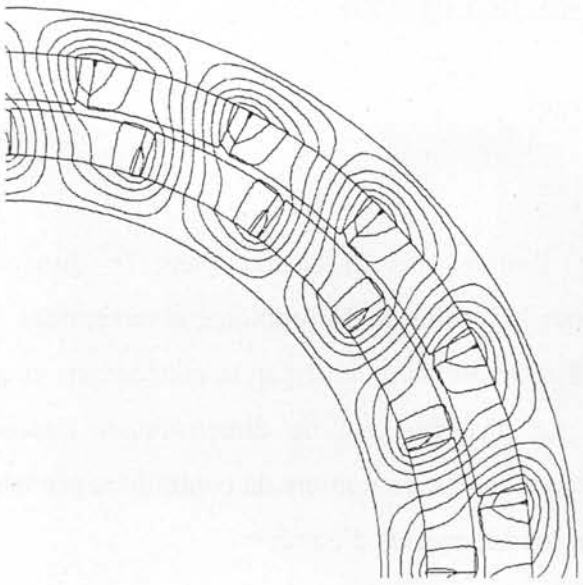


Fig. II-6 : Géométrie initiale de l'accouplement magnétique étudié

Implantation dans Pascosma

On obtient donc un modèle d'environ une dizaine d'équations (avec les expressions des fonctions f et g (EQ II-6 , EQ II-7), et comportant aussi une dizaine de paramètres. Comme on peut le constater en regardant l'équation (EQ II-6), les fonctions analytiques en jeu sont assez complexes et difficiles à dériver sans outil informatique adapté.

On peut noter ici un problème dû à la structure même de Pascosma et que l'on peut mettre en évidence avec cet exemple : le problème de la substitution symbolique. En effet, comme on l'a vu dans le paragraphe 1.2.2 de ce chapitre, Pascosma va systématiquement substituer les expressions symboliques des paramètres de sorties dans les équations. Les équations finales des paramètres de sortie peuvent devenir très grosses, même si cela est géré par Pascosma, et on peut ainsi rencontrer des problèmes de lenteur de génération du code, ou même d'explosion du code généré.

Ainsi, si l'on reprend l'équation (EQ II-6) du modèle de l'accouplement magnétique, on voit bien, que si l'on remplace systématiquement l'expression de la fonction g donnée par (EQ II-7), on peut arriver à des expressions de grande taille, difficiles à manipuler, d'autant plus si les paramètres de la fonction g sont eux-mêmes des paramètres de sortie et qu'on doit leur substituer une expression symbolique. Il ne faut pas oublier que les expressions ainsi substituées sont ensuite dérivées.

2.2 Dimensionnement de l'exemple étudié

2.2.1 Cahier des charges

Position

Pour montrer l'intérêt de Pascosma dans le dimensionnement des systèmes électromagnétiques pour la résolution du problème inverse, nous avons posé un problème de conception, difficile à résoudre sans outil par le concepteur, et qu'il peut être typiquement amené à rencontrer. Le problème est de dimensionner l'accouplement pour un couple maximal donné, en posant un certain nombre de contraintes géométriques.

Contraintes sur les paramètres d'entrée :

- entrefer : e = 5 mm
- rayon moyen : Rm = 100 mm
- magnétisation : J = 1 T
- nombre de paires de pôles : $8 \leq npp \leq 60$ et valeur initiale 10
- longueur axiale : $0.5\ m \leq L \leq 5\ m$ et valeur initiale 3 m
- largeur des aimants : $1\ mm \leq la \leq 31\ mm$ et valeur initiale 31 mm
- épaisseur des aimants : $1\ mm \leq ea \leq 26\ mm$ et valeur initiale 10 mm

Contrainte sur les paramètres de sortie :

- couple maximal : $\Gamma max = 10\ kN.m$

Paramètres discrets

Le nombre de paires de pôles, npp est un paramètre qui évolue de façon discrète. Dans Pascosma, on se place dans un espace réel continu, ce qui est le cas de la plupart des problèmes d'électrotechnique. Le problème des paramètres discrets est contourné en les considérant dans un premier temps comme continus et en les imposant ensuite à la valeur discrète la plus proche. On admet avec cette méthode que l'optimum du problème « discret » correspond à la valeur la plus proche de l'optimum « continu », ce qui n'est pas du tout prouvé. Cette entorse à la théorie donne en pratique de bons résultats compte tenu des équations du domaine.

Contraintes structurelles

Pour optimiser cet accouplement, il nous faut nous assurer au cours du dimensionnement que la géométrie obtenue est cohérente.

Il existe une contrainte sur le système reliant deux paramètres. Elle impose que la largeur des aimants soit inférieure à l'épanouissement polaire :

$$\lambda \geq la \qquad \text{(EQ II-15)}$$

Dans le modèle décrit précédemment, aucune équation ne relie explicitement les paramètres λ et la, donc nous n'avons aucune garantie qu'au cours de l'optimisation, cette inégalité soit respectée.

Ce type de contrainte est un problème classique dans le dimensionnement : imposer un paramètre en fonction d'un autre. Ceci revient à rajouter une équation au modèle, liant les deux paramètres en question, et à rajouter une contrainte au cahier des charges. Dans Pascosma, on est obligé de prendre en considération ce type de contrainte dès la phase de saisie du modèle, car on ne peut rajouter d'équation une fois que le logiciel de dimensionnement est généré. On voit donc tout l'intérêt qu'il y aurait à pouvoir rentrer les équations dynamiquement.

On a donc rajouté au modèle une équation qui prend en compte la contrainte géométriques de réalisation :

$$\varepsilon = \lambda - la \qquad \text{(EQ II-16)}$$

et ε, qui correspond à l'espace entre deux aimants, est contraint à être positif.

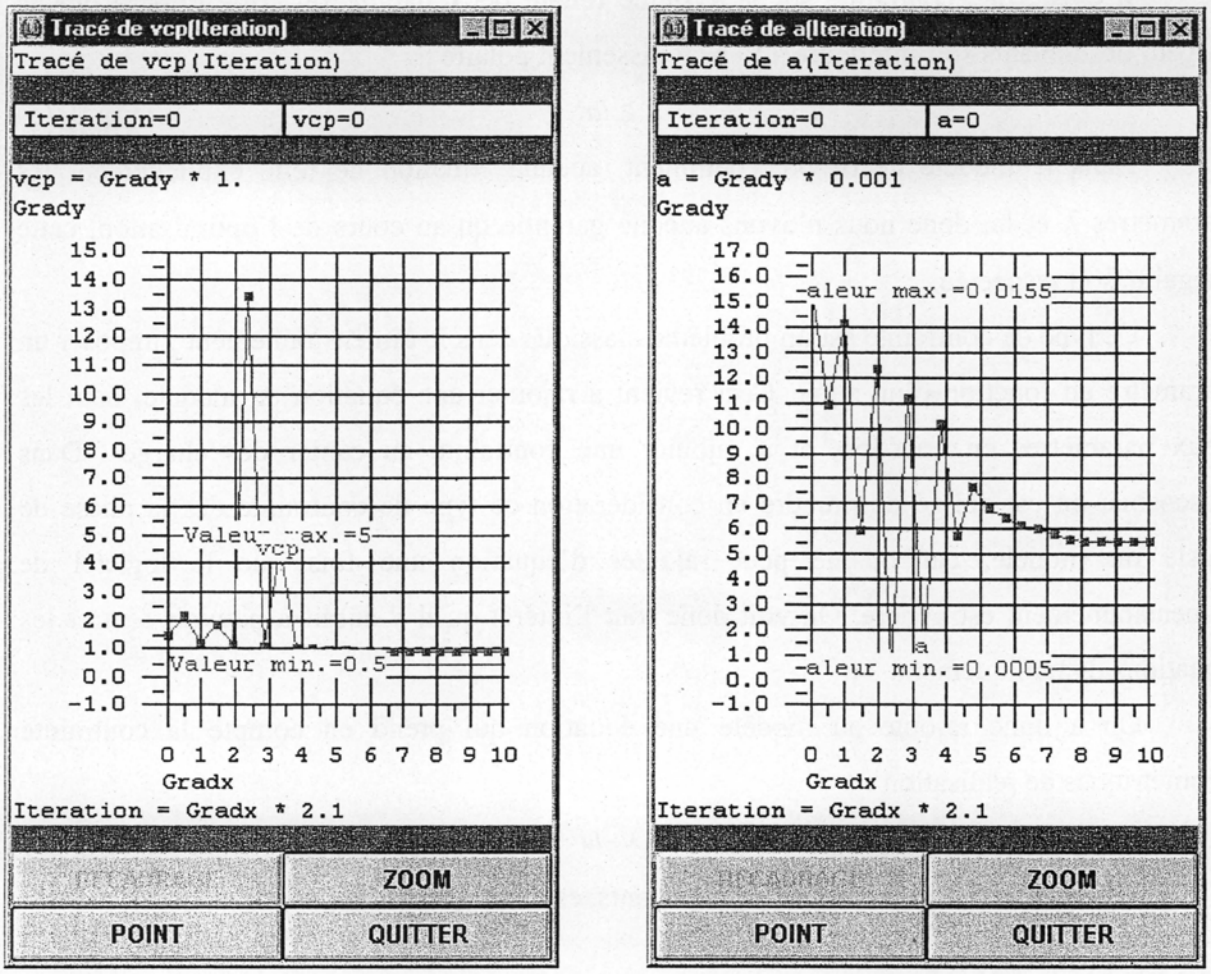
2.2.2 Dimensionnement

Optimisation directe

Le dimensionnement de l'accouplement a été effectué avec Pascosma avec le cahier des charges donné ci dessus (Chapitre 2 § 2.2.1) [COUT 99].

L'algorithme d'optimisation converge en 21 itérations, en moins d'une seconde, vers une solution avec un nombre de paire de pôles non entier (20,3454). La figure II-7 montre l'évolution de la fonction objectif et de la largeur des aimants au cours de l'optimisation.

En fixant npp=20, on obtient ensuite en 6 itérations : la=11.3 mm, ea=3.4 mm et L=3 m. Il est normal que cette dernière grandeur ne varie pas puisque le modèle analytique utilisé est 2D. L'accouplement optimisé est donné Fig II-8. Le couple maximal obtenu Γmax est de 9998.81 N.m, et le rapport couple sur masse d'aimants est Γ/mv=1.083 10^6N/m². Il était de 0.9179 10^6N/m² pour la géométrie initiale mais les contraintes géométriques n'étaient pas les mêmes, l'entrefer étant plus petit (2mm).



(a) (b)

Fig. II-7 : Evolution de la fonction objectif vcp=Γ/mv (a) et de la demi-largeur des aimants a=la/2 (b) au cours de l'optimisation directe

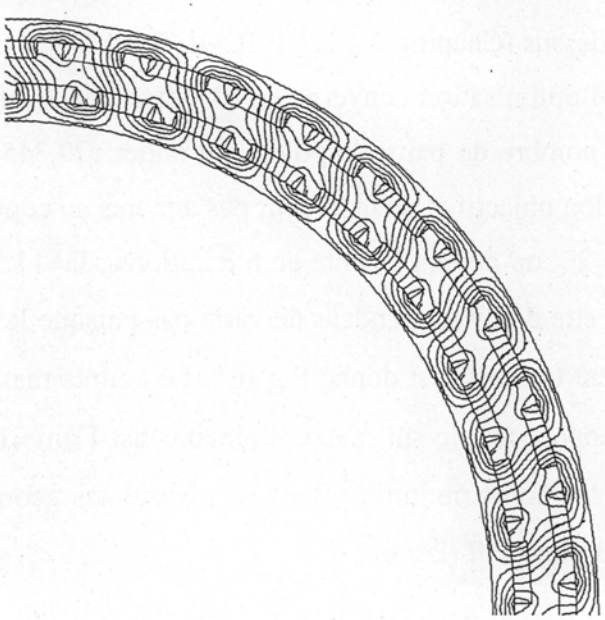


Fig. II-8 : Géométrie de l'accouplement obtenu par optimisation directe

Pascosma permet donc de résoudre le problème inverse par l'optimisation, et ce, de manière rapide et efficace.

Comparaison avec un dimensionnement classique [BARA 83]

Pour les dimensions de l'accouplement étudié, nous avons utilisé des valeurs prises dans l'article de W. Baran.

Ce travail présente une comparaison de plusieurs modèles analytiques, utilisés pour le dimensionnement d'un accouplement magnétique, identique à celui étudié ici. Cependant, l'auteur ne disposant pas d'outil informatique pour le dimensionnement, il a effectué une série de calculs, pour différentes valeurs de nombre de paires de pôles npp, et de dimensions des aimants, la largeur la et l'épaisseur ea. Le rayon moyen Rm et l'entrefer e sont fixes.

Pour comparer ces résultats avec notre approche, nous avons effectué une série d'optimisations avec différentes valeurs de npp, et en laissant varier à chaque fois la et ea, puis une optimisation en laissant varier npp, la et ea. Ces optimisations sont très rapides, elles durent moins d'une seconde sur un Pentium 200Mhz.

On obtient la valeur suivante pour le nombre de paires de pôles : npp=21.58. On effectue alors de nouveau une optimisation, en fixant npp à 22, et on obtient les valeurs suivantes : la=8.4 mm, ea= 2.9 mm, Γmax=12858 N.m et Γ/mv=2 10^6 N/m².

Cette manipulation nous permet de constater que la meilleure largeur pour les aimants est égale à 2/3 de l'épanouissement polaire.

Cette constatation était impossible à faire pour Baran, car il ne peut explorer, même avec une série de calculs, tout l'espace des solutions et connaître précisément l'optimum. En outre, il fixait toujours une valeur proche de l'épanouissement polaire λ pour le paramètre la.

Comme le laissaient supposer les calculs de Baran, l'optimum tend vers un grand nombre de paires de pôles. Nos optimisations nous ont permis néanmoins de voir que cette optimum est assez plat (valeur proche de 2 10^6 N/m² pour Γ/mv) et que le nombre de paire de pôles optimal est vers 22.

Ainsi, l'automatisation du processus de dimensionnement permet de manipuler de manière pratique pour le concepteur, le modèle analytique, et d'en déduire des informations utiles plus facilement.

III – Apports et limites de l'approche Pascosma

Au regard de cet exemple, des autres cas que nous avons pu étudier, et de l'étude que nous avons menée sur le problème de dimensionnement sous contraintes, nous allons mettre

en évidence les apports et limites de la méthodologie d'optimisation Pascosma, en différenciant les problèmes génériques et ceux dus à l'implantation choisie dans cette approche.

3.1 Apports et limites de la méthodologie d'optimisation présentée

L'étude de Pascosma et de la méthodologie mise en œuvre dans cette approche, nous permet de tirer quelques conclusions sur l'optimisation à l'aide de modèles analytiques, et l'utilisation d'une méthodologie pour l'effectuer.

3.1.1 Apports

La méthodologie d'optimisation présentée offre des possibilités intéressantes pour le concepteur, dans la mesure où elle lui permet d'effectuer de manière souple et conviviale le dimensionnement de machines électrotechniques. La résolution du problème inverse, qui peut devenir quelque chose d'impossible à effectuer sans outil informatique est un de ces gros points forts.

Comme nous l'avons souligné au Chapitre 1 § 2.1.3, des outils et méthodologies adaptés deviennent indispensables pour aider le concepteur dans le travail du dimensionnement. En appréhendant ce problème de manière générique, on permet d'offrir un outil évitant de réitérer les mêmes opérations à chaque application traitée, faisant gagner du temps et aussi rendant plus sûr le travail, car il est effectué sur des outils qui ont déjà été testés sur d'autres applications. En outre, ce type d'approche permet de prendre du recul par rapport à l'application traitée. En offrant par exemple des possibilités comme le calcul de sensibilité d'un paramètre, ou la connaissances des interdépendances dans le systèmes, ces outils délivrent des données précieuses pour le concepteur.

Cependant, la méthodologie d'optimisation ne peut résoudre tous les problèmes, et connaît donc certaines limites.

3.1.2 Limites liées à l'utilisation de modèles analytiques

L'approche présentée est basée sur l'utilisation de modèles analytiques.

Orientation

Comme on l'a vu au § 1.1.1, le problème d'optimisation contrainte traite un modèle analytique orienté, en séparant les paramètres en paramètres de sortie et paramètres d'entrée. Dans notre approche, nous souhaitons mettre en œuvre non seulement une méthodologie d'optimisation, mais aussi la propagation de contraintes. Dans ce contexte, nous souhaitons

pouvoir manipuler le modèle analytique selon différentes orientations, ce qui sous-entend l'utilisation d'une méthodologie plus amont de manipulation des modèles analytiques. Ainsi, la méthodologie d'optimisation proposée ne peut prétendre à satisfaire tous les besoins du dimensionnement à l'aide de modèles analytiques. La propagation de contraintes notamment est impossible dans ce contexte.

Systèmes implicites

Un problème rencontré, non avec l'exemple étudié ici, mais avec d'autres, dans le cadre du dimensionnement à l'aide de modèles analytiques est le problème des Systèmes implicites dans le modèle. Ce sont des cycles, qui empêchent d'exprimer les paramètres de sortie du modèle en fonction des paramètres d'entrée seuls, de manière symbolique, et donc d'orienter le modèle.

Ce problème est néanmoins pris en compte dans la méthodologie Pascosma, mais pas de manière générique. Il est à prendre en compte dans toute méthodologie d'optimisation. Nous verrons au Chapitre 3 ce problème plus en détail, et comment il est pris en compte dans Pascosma et dans une nouvelle approche que nous proposerons.

3.1.3 Limites liées à l'algorithme d'optimisation

La méthodologie d'optimisation étudiée est liée à l'algorithme d'optimisation utilisé dans son cœur. Ainsi, les limites inhérentes aux algorithmes de type gradient apparaissent dans l'outil :

- le domaine des variables est continu et réel, et les paramètres discrets, comme on l'a vu dans l'exemple, ne peuvent être traités par l'algorithme de façon rigoureuse. Il en est de même pour les paramètres complexes. On ne peut non plus traiter des paramètres normalisés dont la valeur appartient à une liste donnée (tels le diamètre des fils de cuivre dans une machine par exemple, qui appartient à une certaine gamme proposée par le fournisseur)
- La solution trouvée par l'algorithme peut être un optimum local. Pour pallier ce problème, il faudrait utiliser un algorithme d'optimisation qui ne soit pas trompé par les minima locaux, comme par exemple un algorithme génétique, voire coupler les deux types d'algorithmes [DEVA 94].

3.2 Apports et limites de l'implantation Pascosma

Pascosma est un exemple d'implantation d'une méthodologie de dimensionnement par optimisation de modèle analytiques.

Nous avons listé ci-dessus les limites rencontrées par ce type d'approche, nous allons ici nous intéresser aux limites inhérentes à la structure de Pascosma.

3.2.1 Apports

Pascosma nous a permis, par son étude, de mettre en évidence une méthodologie d'optimisation sous contraintes de modèles analytiques, et les problèmes soulevés par ce type d'approche. Cet outil est une implantation intéressante de cette méthodologie et remplit le rôle qu'il s'était fixé : effectuer le dimensionnement sous contraintes, et résoudre le problème inverse sur un modèle donné, pour un cahier des charges fixé. De plus, à notre connaissance, c'est le seul outil qui se propose d'implanter ce type d'approche, en automatisant le processus de génération des expressions des équations et de leurs dérivées.

3.2.2 Modification des équations

Comme on l'a vu dans l'exemple de l'optimisation directe, un changement dans le modèle est impossible de manière dynamique dans Pascosma : le graphe des équations est figé et orienté. On ne peut pas le réorienter. Cependant, l'impossibilité de modifier le modèle, sans refaire toute la génération du code de l'application, inhérente à l'implantation de Pascosma pose de grandes limitations.

Ainsi on ne peut dans le cahier des charges imposer un paramètre en fonction d'un autre, ce qui revient à ajouter une équation dans le modèle, si on n'a pas pris en compte cela dès le début (cf Chapitre 2 § 2.2.1 sur les contraintes structurelles).

De même, pour modifier la fonction objectif, il faut générer à nouveau l'application.

Le simple ajout ou la suppression d'une équation dans le modèle peuvent entraîner des manipulations très lourdes.

3.2.3 Substitution symbolique

Les modèles analytiques utilisés par les concepteurs tendent à devenir de plus en plus gros, car, effectuant la même démarche que pour les modèles utilisés en éléments finis, on cherche à être de plus en plus précis et à prendre en compte tous les phénomènes intervenant dans les systèmes étudiés : phénomènes magnétiques mais aussi thermiques, mécaniques, vibratoires, etc.

On a mis en évidence au § 2.1.2 avec l'exemple de l'accouplement un autre problème dû à l'implantation de Pascosma : le problème de la substitution, qui entraîne des problème de taille dans les expressions mathématiques manipulées, et un problème d'explosion de code à terme.

En effet, pour chaque paramètre de sortie, m+1 routines (m étant le nombre de paramètres d'entrée du modèle), sont générées, pour son calcul et celui de ses dérivées partielles par rapport à tous les paramètres d'entrée, et ce, de façon systématique. Pour n paramètres de sorties, on obtient donc $n \times (m+1)$ routines FORTRAN générées, ce qui devient très vite trop gros, notamment pour les compilateurs FORTRAN existants [WATCOM].

Par exemple, pour un modèle moyen de 200 équations, comportant 100 paramètres, dont 80 de sortie, ce qui est assez courant, on obtient $80 \times (20+1) = 1680$ routines générées. On voit donc qu'on peut arriver très vite à un nombre important de routines FORTRAN à écrire, compiler et lier.

En outre, cette méthode de calcul fige définitivement l'orientation du modèle analytique étudié.

Pour éviter ce problème, il faudrait par exemple effectuer les calculs par composition, sans substituer symboliquement les expressions des paramètres de sortie dans le modèle.

Conclusions

Au vu des limites que nous avons listées au travers de l'exemple de l'accouplement magnétique et de celles que nous pouvons encore rencontrer dans d'autres problèmes, il apparaît qu'une nouvelle approche est nécessaire. Celle-ci doit permettre de surpasser ces limites et offrir au concepteur un outil polyvalent, offrant les possibilités de la méthodologie d'optimisation qu'implante Pascosma, mais aussi des possibilités nouvelles au niveau de l'utilisation des modèles analytiques dans la conception, pour pouvoir assurer le calcul du modèle selon différentes orientations et la propagation de contraintes. Il est ainsi apparu la nécessité de réfléchir à une autre structure pour une meilleure gestion des équations du modèle.

Mais avant de pouvoir nous pencher sur cette nouvelle architecture, il nous faut résoudre le problème de la prise en charges des systèmes implicites dans les modèles analytiques. Nous ne pouvons reprendre celle qu'effectuait Pascosma, basée sur un modèle orienté et ne permettant pas le calcul simple du modèle. Ce problème se pose en outre presque fatalement dès que l'on cherche à modifier l'orientation du modèle.

La troisième partie montre donc une méthode pour la prise en charge des paramètres implicites dans les modèles analytiques.

Dans la quatrième, nous présentons alors la structure d'une nouvelle architecture proposée pour le dimensionnement à l'aide de modèles analytiques, issue de l'analyse effectuée sur la méthodologie Pascosma et des réflexions sur les problèmes de manipulation des modèles analytiques dans le cadre du dimensionnement sous contraintes. Nous présenterons ensuite l'implantation informatique réalisée.

Chapitre 3. Prise en charge des paramètres implicites

Dans l'optique de conception à l'aide de modèles analytiques, ceux-ci peuvent poser un certain nombre de problèmes, que nous avons listés dans le chapitre précédent. Dans ce chapitre, nous allons évoquer un problème rencontré dans la manipulation d'équations analytiques : les systèmes d'équations implicites, que nous avons cité au Chapitre 2 § 3.1.2. Dans un premier temps, nous allons présenter ce problème dans le contexte de l'optimisation sous contraintes. Nous proposons ici ensuite deux méthodes pour le prendre en charge, que nous allons comparer sur un exemple significatif.

I – Position du problème des paramètres implicites

Nous allons présenter le problème des paramètres implicites et dans quelles circonstances on le rencontre, ainsi que la manière dont il intervient dans le problème d'optimisation sous contraintes d'un système modélisé analytiquement.

1.1 Présentation des paramètres implicites

1.1.1 Qu'est ce qu'un système d'équations implicites ?

Pour calculer un modèle analytique ou effectuer une optimisation à l'aide de ce modèle (Chapitre 1 § 2.1.2), il faut orienter le modèle, c'est à dire séparer les paramètres du modèle en deux catégories : les paramètres d'entrée et les paramètres de sortie, que l'on cherche à exprimer analytiquement en fonction des paramètres d'entrée. Cependant, certains paramètres de sortie peuvent ne pas pouvoir s'exprimer explicitement en fonction de ces paramètres d'entrée.

Par exemple , la variable X, dans l'équation (EQ III-1) :

$$F(P,X)=exp(X)+cos(X+P)+1=0 \qquad \text{(EQ III-1)}$$

ne peut pas s'exprimer en fonction de la variable P, paramètre d'entrée, bien que la fonction F soit analytique.

On appellera X, paramètre implicite et F, équation implicite.

On peut ainsi trouver des systèmes d'équations implicites, c'est à dire que dans le modèle analytique apparaissent des systèmes d'équations qui ne peuvent être résolues que simultanément, car les équations ne sont pas indépendantes. Par exemple, le système d'équations (EQ III-2) est un système d'équations implicites :

$$\begin{cases} X1=2.P1+sin(X2) \\ X2=P2+3.cos(X1+P3) \end{cases} \qquad \text{(EQ III-2)}$$

avec P1, P2 et P3, paramètres d'entrée. X1 et X2 ne peuvent s'exprimer analytiquement et explicitement en fonction de P1, P2 et P3 seulement, ils sont liés l'un à l'autre.

On appellera un tel système, un système implicite.

1.1.2 Un problème courant dans la problématique de conception à l'aide de modèles analytiques en génie électrique

Le problème des systèmes d'équations implicites peut se poser :

- avec un modèle déjà orienté :

Des modèles, type réseau de réluctances non-linéaires, qui prennent en compte la saturation, par exemple, sont des modèles où il est impossible d'éviter les boucles et les systèmes implicites, et ce, même en choisissant un autre jeu de paramètres d'entrée. Ce cas est courant en électrotechnique, où l'on rencontre des problèmes de caractéristiques non-linéaires des matériaux, de saturation, de non-linéarités dans l'électronique de puissance, etc...

- en ré-orientant un modèle :

On doit orienter le modèle pour effectuer la résolution ou l'optimisation. Mais dans une phase de propagation de contraintes, on peut vouloir modifier le jeu des paramètres d'entrée, et donc l'orientation du graphe. Cette manipulation peut induire des systèmes implicites – de même elle peut les faire disparaître. Ainsi, en reprenant l'exemple du système (EQ III-2), si on choisit X1, X2 et P3 comme paramètres d'entrée, le système n'est plus implicite car P1 et P2 peuvent s'exprimer explicitement en fonction de ces paramètres :

$$\begin{cases} P1 = (X1 - \sin(X2))/2 \\ P2 = X2 - 3.\cos(X1 + P3) \end{cases} \qquad \text{(EQ III-3)}$$

1.1.3 Résolution des systèmes implicites

Les systèmes implicites nécessitent une résolution numérique. Certains systèmes d'équations acceptent une résolution symbolique : c'est le problème de "missing propagation path" (MPP), étudié par Saldanha dans sa thèse [SALD 88], où l'on cherche dans le système à résoudre à remplacer certains paramètres par leur expression symbolique, et à la propager. Ces systèmes ne sont pas des systèmes implicites. Nous nous sommes intéressés ici seulement au problème de la résolution numérique des systèmes d'équations implicites.

La prise en charge des systèmes d'équations implicites ne pose normalement pas de problème avec des algorithmes dédiés à la résolution numérique des systèmes non-linéaires,

que l'on peut trouver dans la littérature. Ainsi le calcul d'analyse – ou solver – peut être effectué aisément sur un modèle analytique contenant des équations implicites.

Dans ce cas, la fonction G (EQ III-4), qui permet d'exprimer X de (EQ III-1) en fonction de P, et qui est numérique, est appelée fonction implicite.

$$X = G(P) \quad tq \quad F(P, G(P)) = exp(G(P)) + cos(G(P) + P) + 1 = 0 \qquad \text{(EQ III-4)}$$

Nous allons voir maintenant comment intervient ce problème lorsque l'on veut effectuer un dimensionnement du système étudié, c'est à dire une optimisation sous contraintes, à l'aide d'un algorithme de type gradient, puisque nous nous sommes placés dans ce contexte.

1.2. Problème d'optimisation contrainte avec un modèle orienté contenant des équations implicites

L'optimisation nécessite une partie d'analyse (cf. Chapitre 1 § 2.1.2), qui correspond à l'évaluation du modèle, effectuée normalement par le solver. Dans notre approche, nous souhaitons pouvoir utiliser des algorithmes d'optimisation de type gradient, et nous devons donc pouvoir évaluer les dérivées partielles des paramètres implicites dans la boucle d'optimisation. Il nous faut donc nous intéresser au problème du calcul des dérivées partielles en présence d'équations implicites.

1.2.1 Formulation du problème d'optimisation contrainte en présence de paramètres implicites

Reprenons la formulation du problème d'optimisation contrainte donnée au Chapitre 2 § 1.1.1, en y incluant un système d'équations implicites. On considère donc le modèle décrit par l'ensemble des m équations explicites:

$$Ps_i = f_i(Pe_1, Pe_2, \cdots, Pe_n), \quad i = 1, m \qquad \text{(EQ III-5)}$$

où f_i, i=1,m sont les fonctions mathématiques reliant les paramètres de sortie Ps_i, i=1,m aux paramètres d'entrée Pe_j, j=1,n, et d'un système de s équations implicites :

$$F_k(Pe_1, Pe_2, \cdots, Pe_n, X_1, \cdots, X_s) = 0, \quad k = 1, s, \qquad \text{(EQ III-6)}$$

où X_k, k=1,s sont les paramètres implicites de ce système, qui sont à résoudre, en fonction des paramètres d'entrée Pe_j, j=1,n et F_k, k=1,s sont les fonctions analytiques reliant ces paramètres.

Les contraintes sur les paramètres d'entrée, de sortie et implicites sont :

$$Pemin_f \le Pe_f \le Pemax_f \quad f = 1, q \qquad \text{(EQ III-7)}$$

$$Pe_e = Peconst_e \quad e = q+1, n \qquad \text{(EQ III-8)}$$

$$Ps\,min_g \le Ps_g \le Ps\,max_g \quad g = 1, v \qquad \text{(EQ III-9)}$$

$$Ps_h = Psconst_h \quad h = v+1, m \qquad \text{(EQ III-10)}$$

$$X\,min_r \le X_r \le X\,max_r \quad r = 1, w \qquad \text{(EQ III-11)}$$

$$X_t = Xconst_t \quad t = w+1, s \qquad \text{(EQ III-12)}$$

où q est le nombre de contraintes d'inégalité et (n-q), le nombre de contraintes d'égalité sur les paramètres d'entrée, où v est le nombre de contraintes d'inégalité et (m-v), le nombre de contraintes d'égalité sur les paramètres de sortie et où w est le nombre de contraintes d'inégalité et (m-w), le nombre de contraintes d'égalité sur les paramètres implicites.

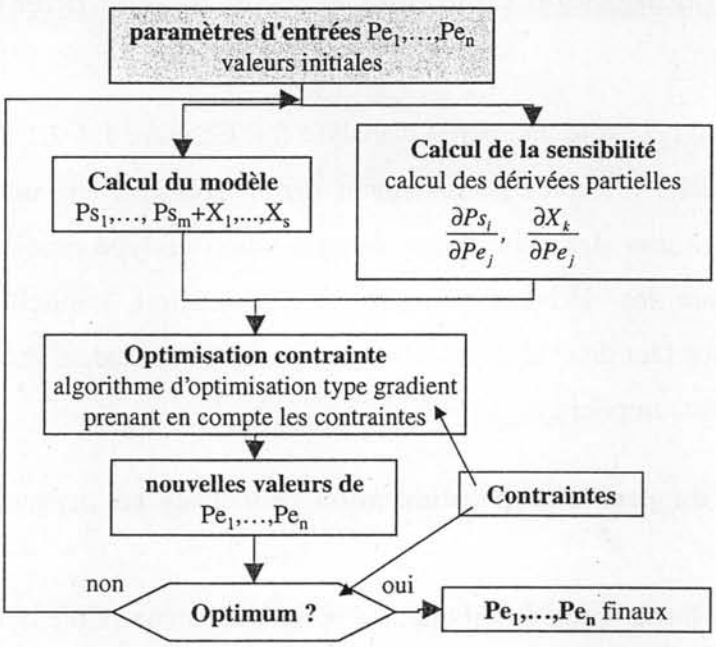


Fig III-1 : Problème d'optimisation contrainte d'un modèle comportant un système d'équations implicites

La figure III-1 présente le problème tel qu'on aimerait le traiter, c'est à dire pouvoir considérer les paramètres X_k, k=1,s, comme des paramètres de sortie, en ayant la possibilité de les évaluer dans la phase de calcul du modèle et de calculer leurs dérivées partielles par rapport aux paramètres d'entrée, dans la phase de calcul de la sensibilité.

1.2.2 Calcul des dérivées partielles des paramètres implicites

Les paramètres implicites X_k, k=1,s ne s'expriment pas analytiquement en fonction des paramètres d'entrée du modèle. Les expressions symboliques des dérivées partielles de ces paramètres ne sont donc, a priori, pas disponibles. Pourtant, nous avons besoin de les calculer pour l'algorithme d'optimisation.

On pourrait envisager, une dérivation numérique des fonctions implicites G_k, k=1,s (EQ III-13), qui résolvent numériquement les équations implicites F_k, k=1,s :

$$X_k = G_k(Pe_1, Pe_2, \cdots, Pe_n), \quad k = 1, s, \qquad \text{(EQ III-13)}$$

par une différence finie.

Cette méthode consiste, pour calculer la dérivée d'une fonction G_k, à utiliser la formule [SING 92] :

$$\frac{\partial G_k}{\partial Pe_i}(Pe_1, \cdots, Pe_i, \cdots, Pe_n) = \lim_{h \to 0} \frac{G_k(Pe_1, \cdots, Pe_i + h, \cdots, Pe_n) - G_k(Pe_1, \cdots, Pe_i, \cdots, Pe_n)}{h}, \qquad \text{(EQ III-14)}$$
$$k \in 1, s \; i \in 1, n$$

Pour calculer cette dérivée, on utilise en général un h très petit, mais non nul évidemment et l'on obtient qu'une approximation de la dérivée, voire une valeur très erronée, car la fonction peut avoir un comportement divergent au voisinage de 0.

Des études ont déjà montré que les algorithmes de type gradient sont moins performants lorsque le calcul des dérivées est effectué de cette manière [RAMA 73]. Des imprécisions sur les calculs se glissent au cours de l'évaluation des dérivées, et peuvent induire en erreur l'algorithme de type gradient dans la boucle d'optimisation.

En appliquant le théorème mathématique des fonctions implicites (cf. Annexe A), on peut calculer les dérivées partielles des paramètres implicites de façon juste, à partir d'expressions analytiques, du moment que l'on possède justement une expression symbolique de la fonction implicite, ce qui est le cas dans notre approche. On garde ainsi tous les avantages d'un algorithme de type gradient, que l'on a décrits au Chapitre 1 § 1.2.4.

Cependant, la méthodologie Pascosma (cf. Chapitre 2), qui utilise un algorithme de type gradient permet de prendre en charge les paramètres implicites d'une autre façon, tout en utilisant des dérivées partielles explicites. Il nous faut donc comparer ces deux méthodes afin d'utiliser la moins pénalisante en terme de convergence et de temps de calcul, et en terme de facilité de mise en œuvre également.

II- Présentation des deux méthodes de résolution pour le problème des paramètres implicites dans l'optimisation contrainte

Pour présenter les deux méthodes, nous les replaçons dans le contexte du dimensionnement d'un modèle analytique, en reprenant la formulation générique du problème d'optimisation contrainte présentée dans la partie précédente. Ensuite, en nous appuyant sur

cette formulation, nous présentons le fonctionnement des deux méthodes, comment est effectué le calcul des dérivées partielles pour l'optimisation dans chaque cas et discuterons de leurs avantages et inconvénients respectifs.

2.1. Première méthode : méthode utilisée dans la méthodologie Pascosma

2.1.1 Présentation

La première méthode étudiée est celle qui a été proposée notamment par F. Wurtz [WURT 96] dans le cadre de la méthodologie Pascosma, et consiste, en présence d'une ou plusieurs équations implicites, à créer des paramètres fictifs qui seront contraints à être nuls. La résolution du système est donc intégrée à l'optimisation et on est obligé d'utiliser une optimisation pour résoudre simplement les équations.

Par exemple le système (EQ III-15) :

$$\begin{cases} X1 = 2.P1 + sin(X2) \\ X2 = P2 + 3.cos(X1 + P3) \end{cases} \quad \text{(EQ III-15)}$$

devient :

$$\begin{cases} Z1 = 2.P1 + sin(X2) - X1 \\ Z2 = P2 + 3.cos(X1 + P3) - X2 \end{cases} \quad \text{(EQ III-16)}$$

où Z1 et Z2 sont les paramètres créés, qui sont des paramètres de sortie, contraints à être nuls. X1 et X2 deviennent des paramètres d'entrée, avec P1, P2 et P3.

En reprenant la formulation générique du problème d'optimisation contrainte donnée précédemment, on crée donc des paramètres de sortie $Cimp_k$, k=1,s tels que :

$$Cimp_k = F_k(Pe_1, Pe_2, \cdots, Pe_n, X_1, \cdots, X_s), \quad k = 1, s \quad \text{(EQ III-17)}$$

Les paramètres X_k, k=1,s, deviennent alors des paramètres d'entrée.

Et on doit rajouter les contraintes d'égalité (EQ III-18) sur les paramètres $Cimp_k$, k=1,s.

$$Cimp_p = 0 \quad p = 1, s \quad \text{(EQ III-18)}$$

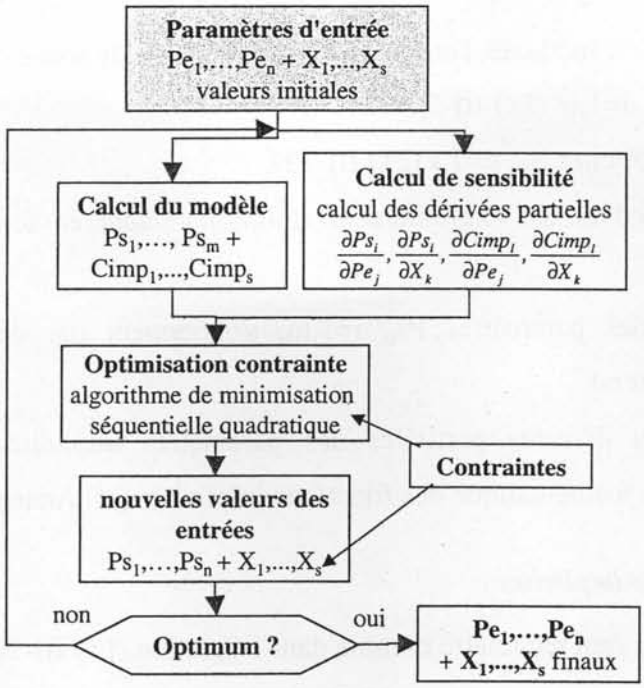


Fig III-2 : Processus d'optimisation contrainte avec la première méthode

2.1.2 Calcul des dérivées partielles

Avec la première méthode, l'ensemble des paramètres de sortie, dont on souhaite calculer les dérivées par rapport aux paramètres d'entrée, est constitué :

- des paramètres Ps_i, i=1,m (EQ III-5),
- et des paramètres $Cimp_k$, k=1,s (EQ III-17).

L'ensemble des paramètres d'entrée est constitué :

- de l'ensemble des paramètres Pe_j, j=1,n,
- et des paramètres implicites X_k, k=1,s du système implicite inclus dans le modèle.

Les paramètres de sortie s'expriment de façon explicite en fonction des paramètres d'entrée (EQ III-5, III-13). Les dérivées partielles des paramètres de sortie sont donc obtenues par dérivation symbolique sans problème particulier.

2.1.3 Mise en œuvre

Dans les deux méthodes étudiées, la routine d'optimisation est une routine de minimisation séquentielle quadratique, issue de la librairie Harwell et nommée VF13AD [HARWE]. Elle nécessite le calcul des paramètres de sortie et de leurs dérivées.

Dans les deux méthodes, nous avons utilisé l'environnement de Pascosma pour la génération automatique des routines FORTRAN de calcul, ainsi que pour l'édition de lien et le pilotage des exécutables obtenus.

2.2.2 Calcul des dérivées partielles

Dans la deuxième méthode, l'ensemble des paramètres de sortie est constitué :

- des paramètres Ps_i, i=1,m (EQ III-5),
- des paramètres implicites X_k, k=1,s (EQ III-19),

qui s'expriment respectivement explicitement et implicitement en fonction des paramètres d'entrée Pe_j, j=1,n.

Les dérivées des paramètres Ps_i, i=1,m, s'obtiennent par dérivation symbolique, comme dans le cas général.

On calcule les dérivées partielles des paramètres implicites de façon juste en appliquant le théorème mathématique des fonctions implicites (cf. Annexe A),

théorème des fonctions implicites

Dans le cas d'un seul paramètre comme dans l'équation (EQ III-20) :

$$F(P,X)=exp(X)+cos(X+P)+1=0, \qquad \text{(EQ III-20)}$$

où X est le paramètre implicite et P celui d'entrée, et si on note G la fonction implicite telle que :

$$X=G(P), \qquad \text{(EQ III-21)}$$

en dérivant (EQ III-20) par rapport à P, on obtient :

$$\frac{dF}{dP}(P,X)=\frac{\partial F}{\partial P}(P,G(P))+\frac{\partial F}{\partial X}(P,G(P)).\frac{\partial G}{\partial P}(P)=0, \qquad \text{(EQ III-22)}$$

De cette équation, on peut déduire la dérivée de X=G(P) par rapport à P :

$$\frac{\partial G}{\partial P}(P)=-\frac{\frac{\partial F}{\partial P}}{\frac{\partial F}{\partial X}}(P,G(P))=\frac{sin(G(P)+P)}{exp(G(P))-sin(G(P)+P)}, \qquad \text{(EQ III-23)}$$

Dans le cas général, on peut exprimer les dérivées partielles des paramètres implicites par rapport aux paramètres d'entrée, en fonction des dérivées partielles des fonctions implicites à annuler constituant le système implicite à résoudre (cf. Annexe A). On doit alors résoudre, pour un système d'équations implicites de degré s, une équation matricielle pour trouver J, la matrice de dimension s×n des dérivées partielles recherchées :

$$J=-Q^{-1}\cdot M \qquad \text{(EQ III-24)}$$

avec $J_{q,j}=\frac{\partial G_q}{\partial P_j}, Q_{q,k}=\frac{\partial F_q}{\partial X_k}, M_{q,j}=\frac{\partial F_q}{\partial P_j}$ et q=1,s, j=1,m et k=1,s et avec les notations de la formulation du problème d'optimisation contrainte donnée plus haut.

M et Q se calculent à l'aide des expressions symboliques des dérivées des fonctions analytiques F_k, k=1,s

Q^{-1} peut être calculé à l'aide d'un algorithme d'inversion de matrice, et peut même être exprimé symboliquement pour des systèmes de taille inférieure à 3.

On peut également envisager d'utiliser un algorithme de résolution des systèmes linéaires pour résoudre les s systèmes linéaires donnés par l'équation matricielle :

$$Q \cdot J = -M \qquad \text{(EQ III-25)}$$

2.2.3 Mise en œuvre

Le calcul des paramètres implicites X_k, k=1,s s'effectuera grâce à une routine Harwell de résolution de systèmes non linéaires à n inconnues (routine NS02AD) qui est une version modifiée de l'algorithme 'dog-leg' de Powell [POWE 70, HARWE]. Cet algorithme utilise notamment le calcul des équations implicites (EQ III-6) et de leurs dérivées partielles. Les équations F_k, k=1,s (EQ III-6), entrées par l'utilisateur et leurs dérivées devront donc être aussi codées dans la deuxième méthode.

Ainsi qu'on l'a précisé précédemment, nous avons utilisé ce type d'algorithme, utilisant les dérivées partielles car il doit converger plus facilement vers la solution, de la même manière que les algorithmes de type gradient sont plus performants dans l'optimisation, ceci grâce au fait que les dérivées partielles guident la recherche dans l'espace des solutions. Le calcul des dérivées étant possible, et un algorithme de ce type disponible dans la bibliothèque Harwell, nous avons décidé de l'utiliser car nous voulions employer les algorithmes les plus performants possibles dans notre approche. Par contre, cet algorithme demande des valeurs initiales pour les paramètres implicites X_k, k=1,s à trouver.

Les dérivées des équations implicites F_k, k=1,s (EQ III-6) sont aussi utiles pour le calcul des dérivées partielles des paramètres implicites X_k, k=1,s car elles interviennent dans la formule (EQ III-25) qui permet d'obtenir ces dernières, dans les matrices Q et M. Pour la résolution de l'équation (EQ III-25) qui est un système matriciel, on aurait pu envisager d'utiliser un algorithme de résolution des systèmes linéaires, mais nous avons choisi de calculer la matrice inverse de Q, à l'aide d'un simple algorithme de pivot complet de Gauss, lorsqu'elle est de dimension supérieure à 3.

Pour intégrer l'utilisation de l'algorithme de résolution des systèmes non-linéaires, nous avons développé un module séparé à l'aide de Macsyma [MACSY], qui génère le code FORTRAN des fonctions nécessaires, que l'on peut ensuite lier aux fonctions créées par Pascosma, lors de l'édition de lien (cf. Annexe B).

2.2.4 Avantages / inconvénients

Aucun paramètre ni aucune contrainte n'est rajouté au cahier des charges exprimé sous forme de contraintes par rapport au problème de départ décrit dans le premier paragraphe (cf. § 1.2.1).

On veut que le calcul du modèle soit possible sans devoir effectuer d'optimisation, et ce pour pouvoir assurer la fonctionnalité solver, c'est à dire pouvoir évaluer le modèle orienté, de façon simple et efficace. Ceci est possible dans cette méthode. En outre, lors de l'optimisation, le jeu de paramètres donné à l'algorithme d'optimisation correspond à un système cohérent, et donc le dimensionnement parcourt bien l'espace des solutions cohérentes.

Cependant, cette méthode est plus lourde et plus difficile à mettre en œuvre pour l'optimisation. Elle induit a priori plus d'appels aux algorithmes et fonctions utilisés, puisqu'elle nécessite le passage dans deux algorithmes, donc deux boucles itératives imbriquées, et plus de fonctions codées.

De plus, elle nécessite des valeurs initiales pour les paramètres implicites, qui sont dans cette méthode des paramètres de sortie, et qui donc ne devraient pas en avoir besoin. Cependant, on peut noter que la première méthode nécessite aussi des valeurs initiales pour ces paramètres car ils apparaissent alors comme des paramètres d'entrée.

III – Mise en œuvre sur un exemple : actionneur linéaire

Pour comparer les deux approches, nous avons étudié plusieurs exemple :

- une fonction implicite simple afin de valider l'implantation sur un cas très simple à gérer
- un modèle analytique d'une dizaine d'équations, comportant un paramètre implicite
- un modèle analytique complexe comportant plusieurs systèmes implicites (modèle par réseau de réluctances).

Les deux premiers cas tests [COUT 99c] nous ont permis de valider l'implantation mais sans donner de résultats vraiment concluant en terme de temps de calcul (les deux modèles étudiés étaient trop simples), ou en terme de convergence de l'algorithme d'optimisation. Nous nous sommes donc intéressés à un exemple plus significatif : un actionneur linéaire, modélisé par un réseau de réluctances [COUT 99b].

3.1. Présentation de l'application

Nous allons dans un premier temps présenter l'application et le cahier des charges posé.

3.1.1 Modèle de l'actionneur : modèle analytique par réseau de réluctances

L'application étudiée est un actionneur magnétique linéaire de type électroaimant (Fig. III-4), avec un noyau central cylindrique mobile. Quand la bobine est alimentée par un courant, la force F créée dans l'entrefer fait bouger le noyau en position "Fermé". La course du noyau est d'environ 1mm. Le système est modélisé dans la position "Ouvert". Dans cette position, la force F est la plus faible, et doit être suffisante pour déplacer le noyau.

Cet actionneur a fait l'objet d'une étude interne au laboratoire d'Electrotechnique de Grenoble, effectuée par M. C. Chillet, dans le cadre d'une collaboration industrielle avec la Société Européenne de Propulsion.

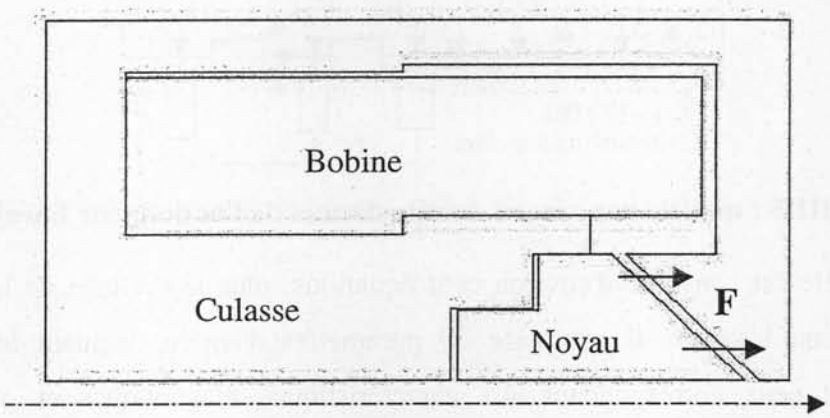


Fig. III-4 : Coupe axi-symétrique de l'actionneur linéaire

Le modèle analytique de cet actionneur est basé sur un réseau de réluctances (Fig.III-5). Le matériau magnétique FeCo utilisé est caractérisé par une courbe H(B) non linéaire, donnant le champ magnétique en fonction de l'induction. Cette courbe a été obtenue à partir de mesures expérimentales déjà effectuées sur ce matériau. Le Fer-Cobalt a été choisi car il possède une valeur de saturation élevée (2.4T), c'est à dire que le volume nécessaire pour obtenir une certaine force F sera plus petit qu'avec un matériau à saturation plus faible. Ce matériau, assez cher, est utilisé quand le volume ou la masse sont des paramètres critiques comme, par exemple, dans les applications spatiales. C'est la non-linéarité du matériau qui induit l'apparition de paramètres implicites, ainsi que le calcul de mailles dû au modèle par réseau de réluctances. On cherche ici à optimiser le volume total de l'actionneur.

Le modèle utilisé est simplifié et ne prend pas en compte tous les phénomènes intervenant, notamment les phénomènes thermiques. Il a néanmoins été validé électromagnétiquement sous FLUX2D [FLUX2D], logiciel d'éléments finis.

Le réseau de réluctances modélisant le système est représenté sur la Figure III-5. Les différentes réluctances correspondent aux flux magnétiques circulant dans l'actionneur. Les flux de fuite ne sont pas négligeables ici. Huit équations implicites correspondant aux huit mailles du réseau apparaissent dans ce modèle. Le système à résoudre est donné par l'équation matricielle non-linéaire suivante :

$$\Re(\phi, p)\cdot\phi = N\cdot I\,, \qquad \text{(EQ III-26)}$$

où $\Re$ est la matrice des réluctances, qui dépendent à la fois du vecteur des 8 flux implicites ϕ et du vecteur des paramètres géométriques du modèle p. N.I représente le vecteur des ampères-tours dans chaque branche du réseau considérée.

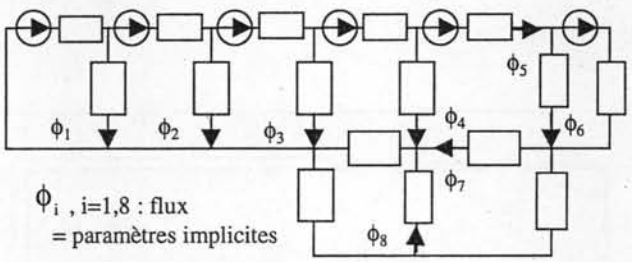


Fig. III-5 : modèle par réseau de réluctances de l'actionneur linéaire

Le modèle est constitué d'environ cent équations, plus le système de huit équations implicites donnant les flux. Il comporte 29 paramètres d'entrée, incluant les paramètres géométriques et ceux correspondant aux caractéristiques magnétiques et physiques des matériaux (densité, résistivité, etc..) et 40 paramètres de sortie, qui correspondent en général aux performances du système, comme la force F, l'induction et les ampères-tours dans les différentes parties de l'actionneur, la masse, le volume, les pertes Joule et quelques paramètres géométriques de sortie.

La force F dépend des flux ϕ, et des paramètres géométriques p. La masse et le volume, qui est la fonction objectif, ne dépendent que des paramètres p.

3.1.2 Cahier des charges

Pour le dimensionnement de cet actionneur, un certain nombre de contraintes sont posées sur le modèle.

La fonction objectif est le volume total de l'actionneur. Celui-ci est optimisé dans la position "Ouvert", c'est à dire, juste avant que le noyau mobile se mette en mouvement. Dans

cette position, la force F est minimale et doit être suffisante pour mettre en mouvement le noyau.

Les contraintes principales imposées par le constructeur sont :

- les dimensions extérieures : l'actionneur doit être contenu dans un cylindre de rayon 30 mm et de longueur 53 mm.
- le poids : le poids total de l'actionneur doit être inférieur à 300g, et le poids du noyau, inférieur à 20g.
- la valeur de la force F : elle doit être supérieure à une valeur minimale fixée (voir Tableau III-1) dans la position "Ouvert". Elle doit être suffisante pour mouvoir le noyau.
- la densité de courant dans le cuivre : elle doit être comprise entre 10^6 et 10^7 A/m2.
- les pertes Joule, qui doivent être inférieures à 6W.

Quelques autres contraintes annexes sont posées sur certains paramètres géométriques d'entrée pour garantir la faisabilité de l'actionneur (jeux internes). Les inductions dans chaque partie, sont imposées inférieures à 2.4 T pour éviter la saturation du Fer-Cobalt.

Les paramètres implicites sont traités à part, suivant la méthode utilisée. Dans la première méthodes, ils sont paramètres d'entrée alors que dans la seconde ils sont paramètres de sortie. Dans les deux cas, des intervalles de variation leur sont imposés.

Dans la première méthode, huit paramètres artificiels, nommés $Cimp_k$, k=1,8, égaux aux équations implicites considérées sont ajoutés et contraints à être nuls.

3.2. Résultats et conclusions

3.2.1 Critères de comparaison

Le fichier du programme exécutable obtenu dans la seconde méthode est cinq fois plus gros que dans la première (5,6 Mo contre 1.15 Mo). Ceci est dû au nombre important de fonctions et routines nécessaires dans la deuxième méthode par rapport à la première.

Les deux méthodes ont été comparées sur les critères suivants :

- nombre d'itérations : de la routine d'optimisation seulement dans la première méthode, et des deux routines – optimisation et résolution de système non-linéaire – dans la seconde.
- temps de calcul.

- qualité de convergence : pour la mesurer, on regarde la précision des résultats obtenus par rapport aux contraintes demandées. Par exemple, la précision sur les paramètres artificiels $Cimp_k$, k=1,8 créés dans la première méthode, qui doivent être nuls. On va donc les calculer aussi dans la deuxième méthode afin de comparer.

La précision des résultats, nommée E dans les tableaux III-1, III-2 et III-3, correspond donc à la précision sur ces paramètres artificiels $Cimp_k$, k=1,8, qui doivent être nuls, c'est à dire que la précision considérée est la valeur absolue maximale parmi ceux-ci. Une précision plus grande que 10^{-6} n'est pas acceptable, au vu de l'ordre de grandeur des flux dans l'actionneur, qui est de 10^{-5} - 10^{-6}.

En outre, les routines utilisées acceptent toutes deux un paramètre de réglage : la précision souhaitée sur les calculs. Cette précision correspond surtout, dans le cas de la routine d'optimisation, à la précision sur la fonction objectif, qui détermine l'arrêt de cet algorithme, les autres paramètres contraints n'étant pas toujours calculés avec cette précision.

Comme la résolution des systèmes non-linéaires s'effectue de manière itérative comme l'optimisation, le nombre d'itérations de cet algorithme est aussi compté. Les deux boucles sont imbriquées. C'est à dire qu'une résolution du système est faite à chaque itération de la boucle d'optimisation. Le nombre d'itérations de la routine de résolution des systèmes non-linéaires, nommé D dans les tableaux III-2 et III-3, correspond au cumul des itérations effectuées à chaque pas de l'optimisation.

3.2.2 Calcul simple ou solver

Comme on l'a dit précédemment, le calcul simple du modèle sans effectuer d'optimisation n'est possible que dans la deuxième méthode. Le calcul simple correspond à la fonctionnalité solver.

Dans la première, il faut effectuer une optimisation, en imposant seulement comme contraintes les paramètres $Cimp_k$, k=1,8 à zéro, et en ne laissant varier en entrée que les paramètres implicites ϕ_i, i=1,8. Il faut ensuite calculer le modèle avec le jeu de paramètres d'entrée optimal trouvé par la boucle d'optimisation, ce qui correspond au temps de calcul mentionné dans le tableau III-1. Comme on peut le constater, ce temps est très petit, ce qui confirme l'intérêt des modèles analytiques pour la phase d'évaluation dans une boucle d'optimisation.

Les Tableaux III-1 et III-2 présentent le résultats de tels calculs pour les deux méthodes. Ces calculs ont été effectués sur un Pentium II 150 Mhz. La légende de ces tableaux est la suivante :

- A : précision de l'algorithme d'optimisation,
- B : précision de l'algorithme de résolution de systèmes non-linéaires,
- C : nombre d'itérations de l'algorithme d'optimisation,
- D : nombre d'itération de l'algorithme de résolution des systèmes non-linéaires,
- E : précision des résultats.

Nous garderons ces notations par la suite.

On remarque que pour une précision égale (10^{-6}) sur l'algorithme d'optimisation et sur celui de résolution non-linéaire, le calcul est plus rapide avec le second algorithme, ce qui était prévisible car cet algorithme est plus simple que l'algorithme d'optimisation et qu'il est plus approprié au problème. Ainsi, on peut remarquer que la résolution est meilleure dans la seconde méthode.

A	temps d'optimisation	temps de calcul	C	E
10^{-6}	0.38 s	0.06 s	3	10^{-6}
10^{-8}	0.38 s	0.06 s	4	10^{-7}

Tableau III-1 : Calcul simple du modèle avec la première méthode

B	temps de résolution	D	E
10^{-6}	0.22 s	9	10^{-7}
10^{-8}	0.22 s	16	10^{-9}

Tableau III-2 : Calcul simple du modèle avec la deuxième méthode

3.2.3 Optimisation

Pour pouvoir comparer les deux méthodes sur plusieurs cas, nous avons modifié la contrainte sur la force F, sans toucher au reste du cahier des charges et en prenant toujours les mêmes valeurs initiales pour les paramètres d'entrée. La force est en effet un paramètre critique de l'application.

Quelques résultats sont donnés dans le Tableau III-3. Les calculs ont été effectués sur un Pentium II 300 MHz.

Pour les différents cahiers des charges testés, la seconde méthode est toujours plus précise que la première, et converge en général en moins d'itérations de l'algorithme d'optimisation.

Par exemple la ligne 8 du Tableau III-3 nous montre que pour la contrainte F>250 N, et avec une précision de l'algorithme d'optimisation de 10^{-5} (ou $<10^{-5}$), le dimensionnement ne

converge pas avec la première méthode. Ainsi, dans tous les tests effectués, si la précision demandée est trop contraignante, l'algorithme d'optimisation n'arrive pas à trouver une solution, ou le fait en beaucoup plus d'itérations que la deuxième méthode. Celle-ci semble converger plus facilement.

n°	méth.	contrainte sur F (N)	F calc. (N)	fonct obj (10^{-5} m^3)	A	B	C	D	E	temps (s)
1	1	F > 150	150	5.78519	10^{-6}	-	153	-	10^{-8}	6.54
2	1	F > 150	150	5.9	10^{-4}	-	94	-	10-6	3.46
3	2	F > 150	149.985	5.78741	10^{-4}	10^{-8}	30	640	10^{-10}	2.91
4	2	F > 150	149.995	5.78515	10^{-5}	10^{-8}	34	644	10^{-9}	3.40
5	1	F > 200	200	6.61418	10^{-6}	-	170	-	10^{-8}	6.64
6	1	F > 200	200.001	6.82446	10^{-4}	-	91	-	10^{-6}	4.01
7	2	F > 200	199.998	6.61415	10^{-5}	10^{-8}	31	538	10^{-11}	2.68
8	1	F > 250	-	-	10^{-5}	-	>200	-	-	-
9	1	F > 250	249.999	8.25076	10^{-4}	-	196	-	10^{-7}	6.97
10	2	F > 250	249.999	7.3881	10^{-5}	10^{-8}	34	464	10^{-9}	2.91
11	1	F > 300	299.999	8.13393	10^{-6}	-	81	-	10^{-8}	3.63
12	2	F > 300	299.918	8.13322	10^{-4}	10^{-8}	62	1326	10^{-12}	5.61
13	1	F > 350	350	8.88458	10^{-6}	-	36	-	10^{-9}	1.43
14	2	F > 350	349.994	8.88449	10^{-5}	10^{-8}	26	63194	10^{-11}	2.70
15	1	F > 400	400	9.72773	10^{-5}	-	87	-	10^{-8}	4.40
16	2	F > 400	400	9.72781	10^{-5}	10^{-8}	39	5472	10^{-9}	4.12
17	1	F > 450	450.002	10.8044	10^{-6}	-	106	-	10^{-9}	3.95
18	2	F > 450	450	10.8044	10^{-5}	10^{-8}	33	1005	10^{-9}	3.11
19	1	F > 500	499.999	11.9118	10^{-5}	-	44	-	10^{-8}	1.98
20	2	F > 500	499.996	11.9102	10^{-5}	10^{-8}	31	667	10^{-9}	2.69

tableau III-3 : Résultats de dimensionnement de l'actionneur linéaire pour différents cahiers des charges avec les deux méthodes

Le temps de calcul est inférieur à 7 s dans tous les cas, et dans les deux méthodes. Ce n'est donc pas un critère significatif de comparaison ici, ces temps étant très courts. En outre, les temps de calcul sont comparables pour les deux méthodes.

Le Tableau III-3 montre que, dans la deuxième méthode, comme les problèmes d'optimisation et de résolution de système non-linéaire sont découplés et traités séparément par des algorithmes appropriés, ils sont mieux gérés et résolus. La résolution du système implicite nécessite de nombreuses itérations de l'algorithme, mais comme celui-ci est plus adapté et plus simple donc plus rapide que l'algorithme d'optimisation, les paramètres artificiels $Cimp_k$, k=1,8 sont mieux annulés. C'est à dire que la solution est meilleure et cela ne prend pas plus de temps qu'avec l'algorithme d'optimisation seul.

Dans la première méthode, l'objectif principal de l'algorithme d'optimisation est de minimiser la fonction objectif. Le respect des contraintes est secondaire, notamment celles sur l'annulation des paramètres $Cimp_k$, k=1,8. Par contre, dans la deuxième méthode, on peut, en jouant sur la précision de l'un ou l'autre algorithme, affiner la précision de l'optimisation, ou du calcul des paramètres implicites. Les deux problèmes sont indépendants, ce qui est souvent le cas dans les problèmes d'électrotechnique où les paramètres implicites portent souvent sur des calculs de flux ou d'induction, alors que la fonction objectif porte sur des volumes ou des coûts. Ainsi, la précision requise de l'algorithme d'optimisation pour converger vers une solution satisfaisante est moins grande dans la deuxième méthode que dans la première. La solution est d'ailleurs souvent meilleure (au vu de la fonction objectif) avec la deuxième méthode.

Conclusions

Il s'avère que la deuxième méthode est plus satisfaisante en termes de convergence des calculs, même si sa mise en œuvre est plus lourde. Cette méthode, en découplant le problème d'optimisation et le problème de résolution des systèmes implicites et en les traitant chacun par un algorithme approprié, permet de mieux les résoudre et d'obtenir de meilleurs résultats, plus précis.

Le temps de calcul reste raisonnable (< 7s).

En outre, cette méthode permet d'effectuer le calcul simple du modèle sans passer par une boucle d'optimisation. A tous points de vue, elle nous semble plus intéressante à mettre en œuvre dans notre nouvelle approche présentée dans les chapitres suivants. Elle nous permet d'intégrer la fonction solver, en traitant la présence de systèmes implicites et cette fonction solver peut ensuite être utilisée dans la boucle d'optimisation pour l'évaluation du modèle, sans problème de convergence ou de temps de calcul a priori.

On voit donc ici tout l'intérêt d'utiliser le théorème des fonctions implicites pour résoudre le problème du calcul des dérivées partielles des paramètres implicites, car il permet

d'utiliser un algorithme du type gradient, sans perte de convergence du fait d'imprécision sur ces dérivées. Les résultats sont identiques, comme on a pu le constater, à une approche "tout analytique".

Conclusion du Chapitre

Nous avons présenté dans ce chapitre deux méthodes, que nous avons comparées, pour la prise en charge des paramètres implicites dans les modèles analytiques. Ce problème, souvent rencontré en électrotechnique, notamment à cause de l'utilisation de matériaux à caractéristiques non-linéaires, se pose particulièrement dans la nouvelle approche que nous proposons pour le dimensionnement des systèmes électromagnétiques. En effet, nous souhaitons pouvoir modifier l'orientation du modèle analytique du système considéré, et cela entraîne souvent l'apparition de systèmes d'équations implicites dans le modèle. La deuxième méthode proposée pour prendre en charge les modèles analytiques s'est avérée plus performante dans le contexte de l'optimisation sous contraintes s'appuyant sur un algorithme de type gradient. Cette méthode est, en outre, plus aisée à mettre en œuvre dans le contexte de notre nouvelle approche. Nous pensons donc l'intégrer à celle-ci.

Nous allons maintenant pouvoir présenter la structure d'un nouvel environnement que nous proposons pour le dimensionnement des systèmes électromagnétiques, s'appuyant sur la modélisation analytique, et permettant d'effectuer la résolution du modèle pour une orientation donnée, la propagation de contrainte sur les équations de celui-ci et l'optimisation sous contraintes.

Chapitre 4. Modélisation d'une nouvelle architecture pour le dimensionnement à l'aide de modèles analytiques

Dans ce chapitre, nous allons présenter l'architecture proposée pour l'environnement de dimensionnement que nous souhaitons développer. Pour ce faire, nous avons établi un cahier des charges, donnant les spécifications que devra remplir cet environnement, pour pouvoir effectuer les manipulations sur un modèle analytique entré par l'utilisateur. Après avoir présenté ce cahier des charges, nous nous intéresserons à l'architecture de l'environnement en nous penchant plus particulièrement sur l'implantation des fonctionnalités de manipulation d'un modèle analytique, et à la structuration des données.

I – Cahier des charges

Pour établir la structure à donner à notre environnement, nous avons besoin de définir le cahier des charges que devra remplir celui-ci.

L'objectif global de notre approche est d'offrir au concepteur un outil pour la conception, lui permettant d'exploiter toutes les possibilités des modèles analytiques dans la phase de dimensionnement d'un système électromécanique.

Dans cette optique, nous avons dégagé au chapitre 1 § 2.1.2 trois fonctionnalités principales à mettre en œuvre sur un modèle analytique : la propagation de contrainte, le calcul d'analyse ou solveur et l'optimisation sous contraintes.

Dans un premier temps, nous allons examiner la gestion qui doit être faite du modèle, puis nous nous pencherons sur la satisfaction des fonctionnalités désirées.

1.1 Gestion du modèle

1.1.1 Constitution du modèle

Eléments du modèle

Le modèle est constitué d'équations et de paramètres. Dans la structure de base de notre environnement, nous nous intéressons exclusivement aux équations analytiques, continues et dérivables, telles que nous les avons définies au Chapitre 1 § 2.1.2. Les paramètres sont considérés réels.

Pour illustrer notre propos, nous allons utiliser le modèle analytique simple d'une résistance, donné au Chapitre 1 § 2.1.2, composé de cinq équations :

$$Eq1 : U = R \times I \qquad \text{(EQ IV-1)}$$

$$Eq2 : R = \frac{\rho \times L}{S} \qquad \text{(EQ IV-2)}$$

$$Eq3 : P = R \times I^2 \qquad \text{(EQ IV-3)}$$

$$Eq4 : S = \frac{\pi \times D^2}{4} \qquad \text{(EQ IV-4)}$$

$$Eq5 : V = L \times S \qquad \text{(EQ IV-5)}$$

où R est la résistance, U, la tension électrique appliquée, I, le courant qui la traverse, ρ, la résistivité du matériaux, P, la puissance dissipée, L, la longueur, S, la surface, D, le diamètre et V, le volume total.

Représentation du modèle : le graphe

Pour travailler sur le modèle, nous avons choisi de le représenter sous forme d'un graphe bipartite, dont les nœuds sont les équations et les paramètres (Fig. IV-1). Cette représentation est également celle utilisée dans DONALD [SWIN 91].

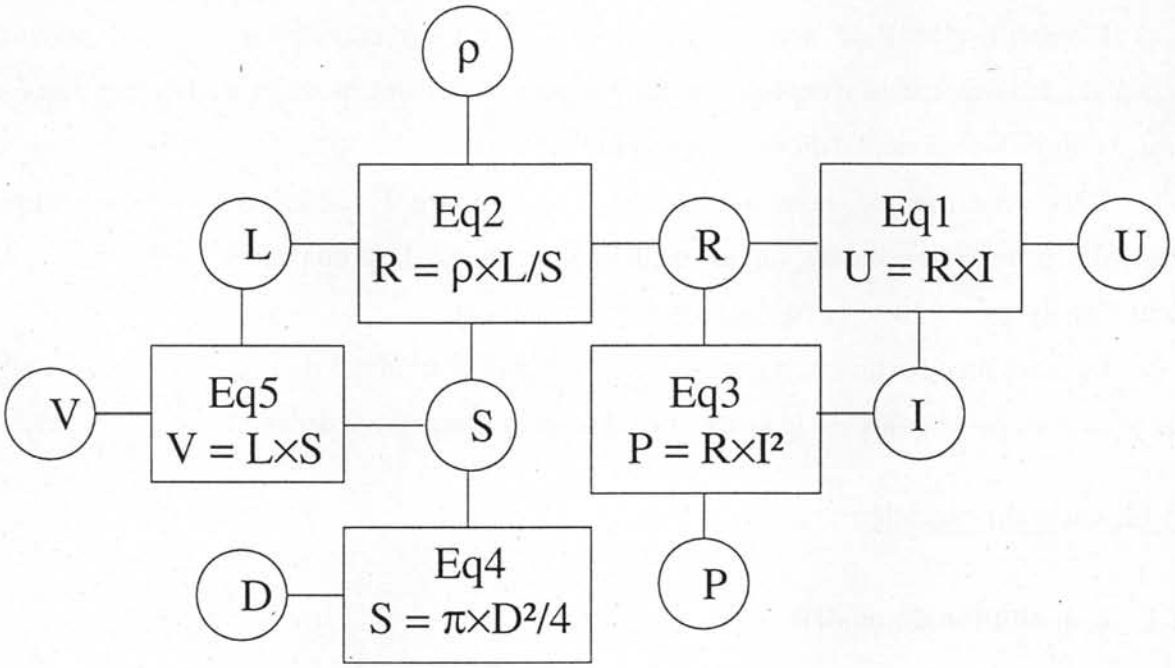


Fig. IV-1 : Graphe du modèle de la résistance

Cette représentation nous permet d'illustrer notre réflexion, mais elle ne peut pas vraiment servir au concepteur, car le graphe devient très vite illisible pour des modèles plus compliqués.

1.1.2 Opérations élémentaires sur le modèle

Nous devons envisager de pouvoir créer et supprimer des éléments du modèle considéré, c'est à dire que le modèle n'est pas figé. Cette souplesse doit permettre à l'utilisateur d'utiliser l'environnement pour développer son modèle

Le modèle est constitué de l'information entrée par l'utilisateur, c'est à dire les équations. Mais dans le processus de dimensionnement, on utilise des informations extraites de ces équations, comme par exemple les dérivées partielles ou encore les différents sens de parcours possibles des équations. Toutes ces informations ne peuvent être données par l'utilisateur. C'est à l'environnement de les déduire. Nous devons donc envisager des mécanismes pour extraire l'information d'une équation entrée par l'utilisateur.

Avec différents modèles analytiques que nous avons pu étudier, nous avons eu la demande de pouvoir évaluer plusieurs jeux de paramètres en parallèle, par exemple les points de fonctionnement au démarrage, nominal et à vide d'une machine asynchrone, afin d'optimiser le modèle pour ces trois points. Il nous faut donc introduire la possibilité d'évaluer la même équation avec plusieurs jeux de paramètres différents en parallèle.

Par exemple, en reprenant l'exemple de la résistance, on peut vouloir calculer la tension U et la puissance dissipée pour différentes valeurs de courant I . On dupliquera dans ce cas les équations (EQ IV-1) et (EQ IV-3), et l'on créera deux jeux U1, P1, I1 et U2, P2 et I2 de paramètres.

1.2. fonctionnalités

Nous allons décrire les fonctionnalités que nous désirons mettre en œuvre sur le modèle des équations analytiques et les besoins au niveau du graphe des équations qu'elles engendrent.

1.2.1 Propagation de contraintes

La propagation de contraintes, comme on l'a vu au Chapitre 1 § 2.1.3, entre en jeu à la fois dans la phase d'analyse, car elle permet de calculer le modèle en l'orientant, et dans la phase de dimensionnement, car elle permet de parcourir l'espace des solutions par tâtonnement. Le calcul d'analyse, ou solveur, et l'optimisation s'effectuent, eux, sur un modèle orienté. Nous allons donc distinguer ici la fonction d'orientation du modèle et la fonction de propagation de contraintes proprement dite.

Orientation du modèle

La fonctionnalité d'orientation sur le graphe induit un certain nombre de fonctions à remplir sur le modèle :

- fixer des paramètres qui deviennent dès lors des paramètres d'entrée :

Cela veut dire que l'on doit distinguer les paramètres d'entrée et de sortie. Il faut donc enregistrer cette information quelque part. Les paramètres doivent aussi accepter une valeur, que l'on doit stocker.

Le fait de fixer des paramètres d'entrée implique que l'on oriente le modèle. Les paramètres qui peuvent se déduire de ceux qu'on a fixés deviennent des paramètres de sortie. Une orientation correspond au choix d'une liste de paramètres d'entrée pour le modèle, qui soit cohérente. Par exemple sur la Figure IV-2, les paramètres d'entrée sont : ρ, L, I et D et les paramètres de sortie : U, R, V, S et P. L'orientation, utilisée dans la phase de calcul et dans celle d'optimisation, est une information à stocker.

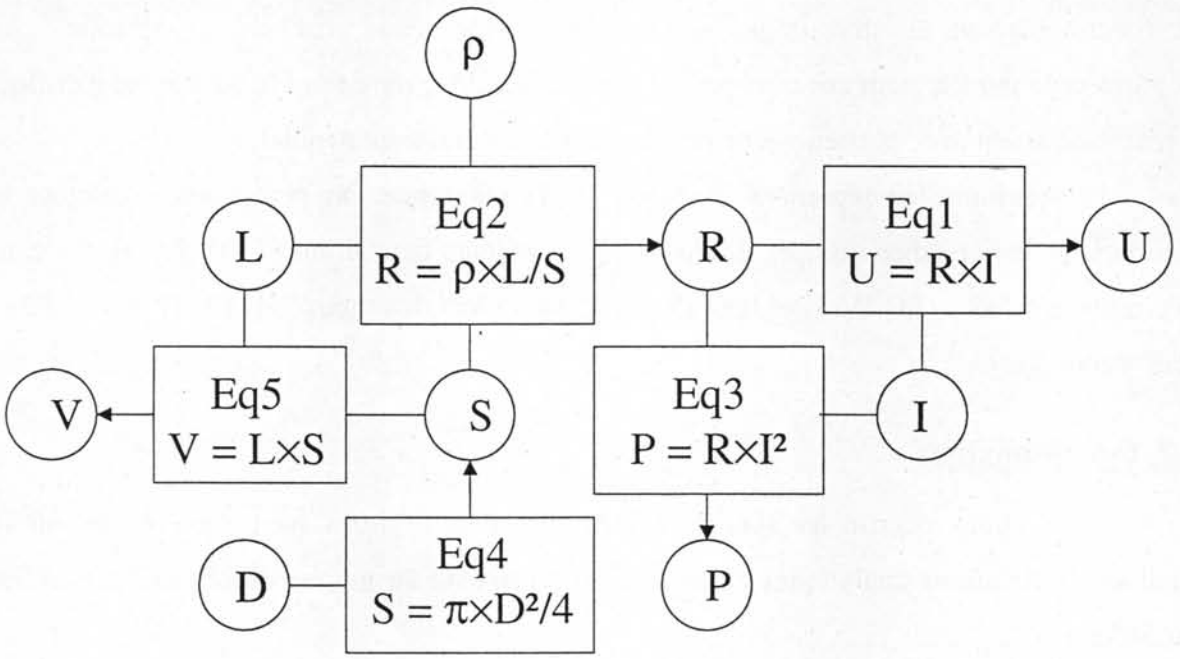


Fig. IV-2 : Graphe orienté du modèle de la résistance

- calculer les degrés de liberté du modèle:

Ceci revient à savoir quel est le nombre de paramètres à fixer en entrée, pour pouvoir calculer le modèle. Le nombre de degrés de liberté est donné par le nombre total de paramètres, moins le nombre d'équations.

Dans l'exemple de la résistance, le nombre d'équations est 5, le nombre de paramètres 9. On doit donc fixer 4 paramètres pour pouvoir déterminer tous les autres. Cependant, on ne peut pas fixer n'importe lesquels. Si par exemple, on choisit de fixer à la fois U, R et I, on peut s'attendre à ce que l'équation (EQ IV-1) – U=R×I – ne soit pas cohérente. C'est pour cela qu'on choisit les paramètres d'entrée un par un, en en déduisant à chaque fois les paramètres de sortie que l'on peut calculer. Le calcul des degrés de liberté au fur et à mesure, donné par le

nombre de paramètres non déterminés moins le nombre d'équations non résolues, détermine le nombre de paramètres d'entrée qu'il faut encore fixer.

Propagation

La fonctionnalité de propagation de valeurs ou de contraintes, telle que nous souhaitons en disposer dans notre environnement, nous a amenés à lister un certain nombre de fonctions à remplir par le graphe :

- pouvoir parcourir les équations dans tous les sens possibles

Pour propager les valeurs suivant l'orientation choisie, et pouvoir modifier cette orientation de manière simple, il est souhaitable de pouvoir évaluer les équations dans tous les sens possibles, suivant les paramètres que l'on fixe en entrée. Par exemple, pour l'équation (EQ IV-1) – U=R×I –, chaque variable U, R ou I est calculable à partir du moment où les deux autres sont fixées.

L'évaluation numérique pour calculer un des paramètres est toujours possible, par un algorithme de résolution de fonction non-linéaire, même si, dans ce cas, l'unicité de la solution n'est pas du tout garantie. C'est pourquoi nous avons choisi d'utiliser la possibilité du calcul symbolique liée à l'utilisation d'équations analytiques, pour effectuer la propagation des calculs. Quand une équation est calculable symboliquement dans un sens donné, qui permet de calculer une variable X, on dira que l'équation est inversible par rapport à X, et par extension que le paramètre X est inversible. Les sens d'inversibilité d'une équation et les expressions correspondantes sont des informations que ne peut rentrer l'utilisateur, ce serait trop fastidieux. C'est donc à l'environnement de déduire, à partir d'une équation, quelles sont les variables inversibles et de calculer symboliquement leur expression.

- avoir une algorithmique de résolution des équations

A chaque paramètre fixé, il nous faut déterminer si une ou plusieurs équations sont calculables pour en déduire les paramètres de sortie correspondants. Il faut donc un algorithme pour résoudre le modèle, au fur et à mesure.

Il nous faut également pouvoir revenir en arrière, ce qui est le propre d'un algorithme de propagations de contraintes, et prendre en compte la modification d'une valeur d'entrée, ou le changement de statut d'un paramètre.

- prendre en compte un cahier des charges sur le modèle

On veut pouvoir signaler à l'utilisateur si les valeurs fixées en entrée et si celles obtenues en sorties sont compatibles avec le cahier des charges sur le modèle. Il faut donc pouvoir enregistrer les contraintes sur les paramètres.

1.2.2 Solveur

Le solveur s'effectue sur un modèle orienté. On suppose donc qu'on a déjà effectué une propagation de contraintes, en fixant les paramètres d'entrée les uns après les autres. On a enregistré l'orientation. On peut éventuellement enregistrer également le parcours de résolution des équations que l'on peut réutiliser dans le solveur, pour effectuer l'enchaînement des calculs de manière rapide.

On souhaite intégrer la résolution des systèmes d'équations implicites, car c'est un problème qui se rencontrera forcément, comme on l'a vu, dans des problèmes d'électrotechnique, et particulièrement si l'on cherche à modifier l'orientation du modèle. Il faut donc que l'algorithme de propagation que l'on a décrit dans le paragraphe précédent soit capable de détecter de tels systèmes.

1.2.3 Optimisation

De la même manière que le solveur, l'optimisation s'effectue sur un graphe orienté.

L'optimisation, comme on l'a vu précédemment est constituée d'une boucle : évaluation du modèle et des dérivées partielles, passage dans l'algorithme d'optimisation, test sur l'optimalité du jeu de paramètres d'entrée trouvé, sinon réévaluation, etc...

On utilise le solveur dans la partie analyse de l'optimisation.

On intercale en général un algorithme de normalisation avant et après le passage dans l'algorithme d'optimisation.

Il nous faut ici, pour un algorithme de type gradient comme nous souhaitons en utiliser, le calcul des dérivées partielles des paramètres de sortie en fonction des paramètres d'entrée. Etant donné que les équations peuvent avoir différents sens de parcours selon l'orientation adoptée, il faudra de même pouvoir évaluer les dérivées symboliques dans les sens possibles d'inversion des équations.

On veut effectuer une optimisation sous contraintes, il faudra donc pouvoir prendre en compte un cahier des charges, comme pour la propagation de contraintes. Ce cahier des charges porte sur des paramètres aussi bien d'entrée que de sortie du modèle orienté.

L'optimisation nécessite une fonction objectif, qui est minimisée par l'algorithme d'optimisation et permet de diriger la recherche dans l'espace des solutions (cf. Chapitre 1 § 1.2.3). Cette fonction objectif est en général rajoutée au modèle comme une équation du modèle, comme par exemple le calcul du coût de la masse d'aimants dans un moteur synchrone à aimants. Mais ce peut être une équation faisant déjà partie du modèle, comme le calcul du couple. En pratique, l'utilisateur choisit le paramètre de sortie correspondant à la grandeur qu'il veut minimiser (coût ou inverse du couple par exemple) comme fonction objectif. Il arrive souvent que le concepteur souhaite effectuer plusieurs optimisations avec différentes fonctions objectif. Nous souhaitons donc pouvoir permettre à l'utilisateur de changer de fonction objectif, et de choisir en pratique n'importe quel paramètre en fonction objectif de l'algorithme d'optimisation.

Toutes les spécifications listées dans ce cahier des charges sont les spécifications principales de notre environnement. D'autres services comme la sauvegarde, l'interface, sont à étudier par la suite.

En perspective, on cherchera à intégrer dans l'environnement des objets plus complexes que des équations, ou à prendre en compte des paramètres non réels, ou encore à modifier les algorithmes utilisés de manière simple.

II - Structure détaillée de l'architecture

Pour mettre en œuvre les éléments et fonctionnalités décrits ci-dessus, nous devons mettre en place une structure adaptée et performante. Nous allons présenter ici cette structure, d'abord globale, puis plus en détails avec la structure de donnée choisie pour la représentation du modèle où viennent se greffer les fonctionnalités et services.

Pour créer notre environnement, il a fallu effectuer un certain nombre de choix. L'un de ces choix principaux est celui d'une structure orientée objet. Après avoir expliqué la raison de ce choix, nous présenterons l'architecture générale.

2.1. Choix d'une représentation objet

Par rapport à une approche procédurale, comme c'est le cas par exemple dans la méthodologie Pascosma, la programmation orienté objet nous semble offrir de nombreux avantages pour construire notre environnement de conception.

Structuration objet

Tout d'abord le grand intérêt des langages à objets est de permettre la création de « moules », c'est à dire les classes, qui sont réutilisables autant de fois que nécessaire. Ainsi, si l'on définit le moule type d'une équation, qui consiste en ses propriétés spécifiques et les fonctions qu'elle peut accomplir, on pourra réutiliser ce moule pour toutes les équations du modèle, et l'on saura manipuler n'importe quelle équation car les fonctions utiles seront définies de manière générique dans le moule.

Il nous suffit de définir la structure générique des objets qui doivent intervenir dans l'environnement et les relations qui les lient, pour ensuite utiliser cet environnement avec différents modèles, implémentant différents jeux de ces objets.

On peut aussi facilement rajouter ou modifier des objets, pour peu que les fonctions qu'ils offrent soient celles définies par le moule générique, même si elles sont effectuées de manière différente dans l'objet modifié.

Polymorphisme

Notre environnement est dédié avant tout à l'intégration des modèles analytiques, et donc des équations analytiques, mais nous envisageons de le laisser suffisamment souple pour y intégrer des objets plus hétéroclites, pouvant intervenir dans un modèle utilisé pour la conception.

Déjà, il serait souhaitable de pouvoir y intégrer des objets : "systèmes d'équations", c'est à dire pouvoir intégrer dans un modèle un sous-modèle, ou encore des systèmes implicites.

Mais on pourrait envisager d'intégrer des objets numériques, permettant de calculer très précisément un ou plusieurs paramètres, comme par exemple une application FLUX2D [FLUX2D] permettant de calculer l'induction dans l'entrefer d'une machine. On pourrait aussi vouloir intégrer des fonctions d'interpolations de mesures, représentées sous forme de tableaux, ou d'autres types de calculs, comme des minimisations par les moindres carrés, des fonctions affines par morceaux [SAUV 99, SAUV 99b], et pourquoi pas encore des équations différentielles, très courantes en électrotechnique.

Ceci est gérable dans un environnement objet, comme nous le verrons par la suite, car il suffit de créer une classe mère « Objet de Calcul » générique, dont héritera la classe « Equation ».

On utilise ici le principe de l'héritage, ou polymorphisme. C'est à dire que la classe Equation possédera les mêmes fonctions que la classe Objet de Calcul, mais ne les remplira pas de la même manière, elle effectuera des manipulations spécifiques aux équations..

Nous ne proposons pas ici de résoudre tous ces problème mais de créer un environnement suffisamment ouvert pour pouvoir y intégrer, au fur et à mesure de l'utilisation et des évolutions, des objets dans le modèle qui ne soient pas forcément des équations analytiques, en prenant en compte bien sûr les limitations qui en découlent pour le calcul et l'optimisation.

Souplesse et ré-utilisabilité

La modification et les rajouts sont plus simples dans un langage objet si les objets sont bien faits. En effet, si le "moule" de l'objet est bien défini, et si on modifie l'objet en restant dans ce moule, le reste du programme n'est pas à toucher. On voit donc aussi l'avantage en terme de maintenabilité et d'évolutivité de l'environnement créé.

Dans notre environnement, nous envisageons d'utiliser différents algorithmes d'optimisation, ou de résolution. On peut envisager de les interchanger de manière simple si le "moule" correspondant à l'algorithme d'optimisation ou de résolution est bien défini.

Un autre intérêt des objets est qu'ils peuvent être réutilisés. Ainsi, le travail effectué pour différents aspects de l'environnement peut être capitalisé et réutilisé ensuite dans d'autres outils. De même, pour un besoin particulier, on peut récupérer des classes déjà existantes, qui y répondent.

2.2. Structure générale

Comme nous l'avons décrit dans la partie précédente, le cahier des charges de l'environnement que nous voulons réaliser s'articule autour de la gestion du modèle, des fonctionnalités propagation de contraintes, solveur et optimisation et des services de sauvegarde, persistance et interfaçage. Le modèle sous forme d'objet Graphe constituera le noyau de l'architecture, sur lequel viendront se greffer les fonctionnalités et les services.

2.2.1 Représentation du modèle : l'objet Graphe

La base de notre approche est le modèle analytique, il nous a donc fallu réfléchir à une représentation du modèle. Dans ce chapitre, § 1.1.1, nous avons associé une représentation graphique au modèle que nous avons appelé graphe. Nous allons décrire maintenant la structure objet correspondant au modèle et contenant le graphe des équations : l'objet Graphe.

L'objet Graphe créé, représenté schématiquement sur la Figure IV-3, contient le modèle, c'est à dire les équations et les paramètres, ainsi que toute l'information liée à ces éléments, comme par exemple les sens d'inversion d'une équation ou les expressions

symboliques des dérivées, et les fonctions de base de gestion du modèle, qui servent à manipuler directement le graphe des équations. Le Graphe assure la gestion des liens entre les objets du modèle, la création et suppression de ces objets, le calcul et le stockage de caractéristiques concernant le modèle complet, comme le nombre de degrés de liberté ou l'orientation. Ainsi le modèle est englobé dans une structure permettant de le manipuler facilement depuis l'extérieur.

Les équations et paramètres sont stockés sous forme d'objets, ou de pointeurs sur des objets, dans l'objet Graphe. La structure des objets constituant le modèle, sera décrite plus précisément dans la suite.

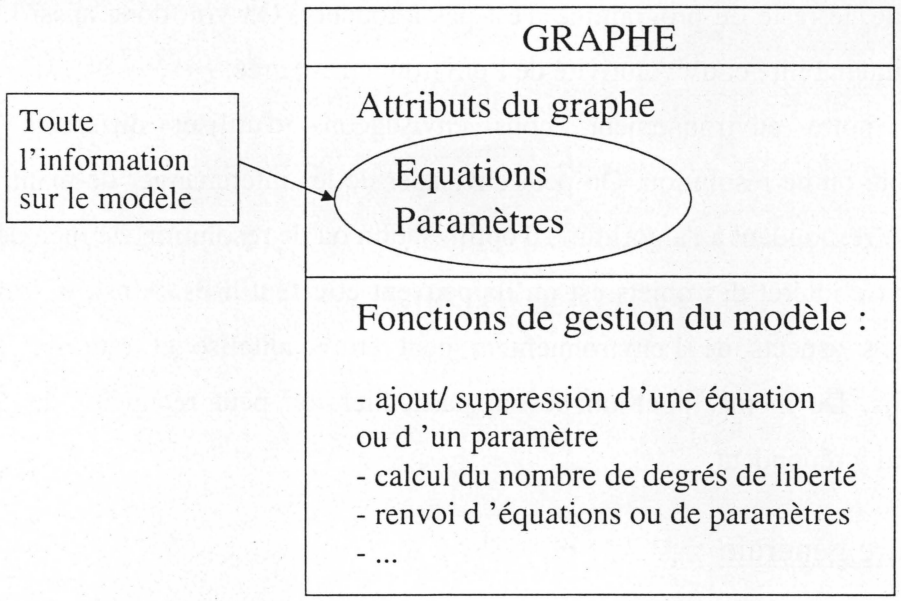


Fig. IV-3 : Structure objet contenant le modèle : le Graphe

2.2.2 Architecture

L'objet graphe, noyau de l'architecture est entouré d'une couche d'objets qui permettent la propagation de contraintes, le solveur et l'optimisation, ainsi que différents services.

La Figure IV-4 présente cette architecture.

Nous avons distingué l'orientation du modèle, et la propagation de contraintes proprement dite. Les services sont prévus pour assurer les fonctions annexes, développées par la suite, comme par exemple la sauvegarde.

Les différentes fonctionnalités ainsi que les services utilisent les fonctions fournies par le graphe et les objets du graphe. Chaque fonctionnalité est décrite par un objet générique, qui sera ensuite implanté de façon spécifique selon les algorithmes utilisés.

Nous avons prévu d'insérer une couche « Interface non-graphique » entre les objets de l'environnement et l'interface graphique, de manière à pouvoir découpler celle-ci, ou à pouvoir piloter l'environnement via un autre logiciel.

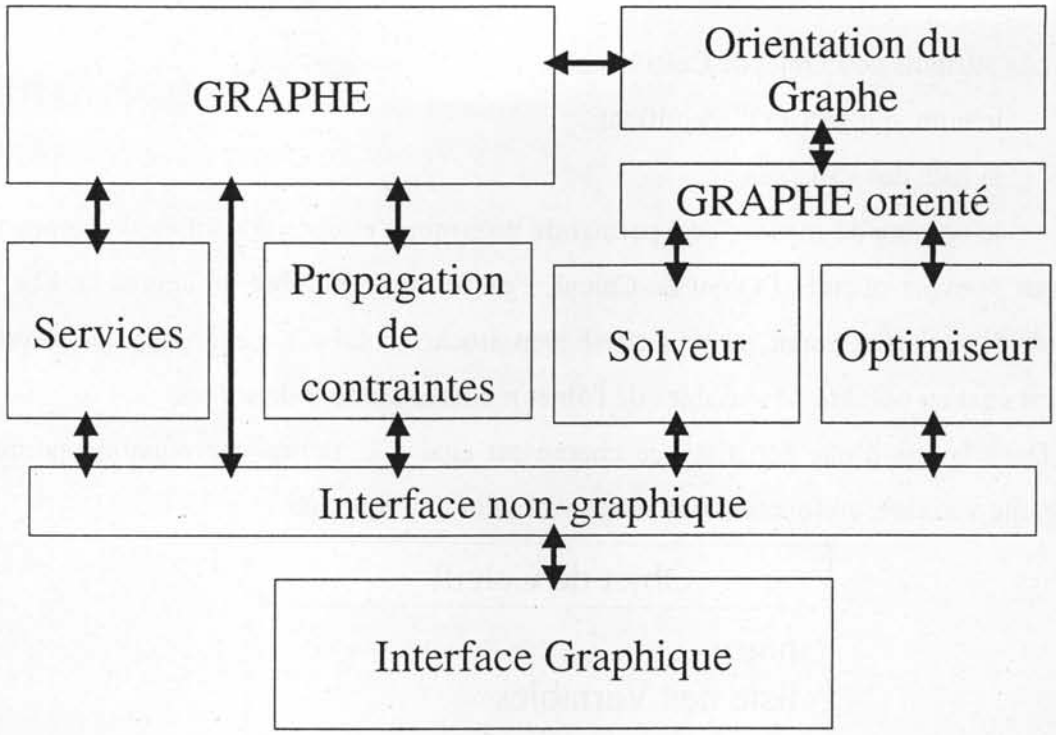


Fig. IV-4 : schéma de l'architecture proposée

2.3. Structure des objets du Graphe

Nous allons décrire ici plus précisément l'objet servant à la représentation du modèle : le Graphe, et les structures objet associées aux différents éléments composant le modèle analytique.

2.3.1 Objets de calcul et Equations

L'information de base du modèle analytique est constituée des équations. Nous avons donc créé des objets correspondants. Cependant, nous souhaitons pouvoir par la suite intégrer à notre environnement des objets qui ne soient pas des équations, comme des systèmes d'équations ou des objets effectuant un calcul numérique. C'est pourquoi nous avons pensé à la structure d'« Objets de calcul ».

Un Objet de calcul est un objet générique, un moule, qui permet ensuite de venir implanter un objet tel qu'une équation. Il définit des attributs et fonctions, qui sont communs à tout objet pouvant être utilisé dans le modèle. Pour l'instant, nous n'avons défini que l'objet de calcul générique et l'objet Equation, qui hérite du précédent. On pourra venir créer par la

suite d'autres objets héritant de l'objet de calcul générique, qui devront implanter les fonctions décrites dans celui-ci (Fig. IV-6).

Nous allons décrire le moule des Objets de Calcul, que nous illustrons sur la Figure IV-5 .

Les attributs de l'Objet de Calcul sont :

- le nom, qui servira d'identifiant,
- la liste des variables,
- le nombre de sorties : ceci permet de déterminer ensuite le nombre de paramètres à fixer pour pouvoir calculer l'Objet de Calcul, c'est à dire le nombre de degrés de liberté de l'Objet de Calcul (On aurait pu tout aussi bien stocker celui-ci). Le nombre de degrés de liberté est égal au nombre de variables de l'objet moins le nombre de sorties.

Dans le cas d'une équation, ce champ est égal à 1, puisqu'une équation permet de calculer une variable, en fonction des autres variables de l'équation.

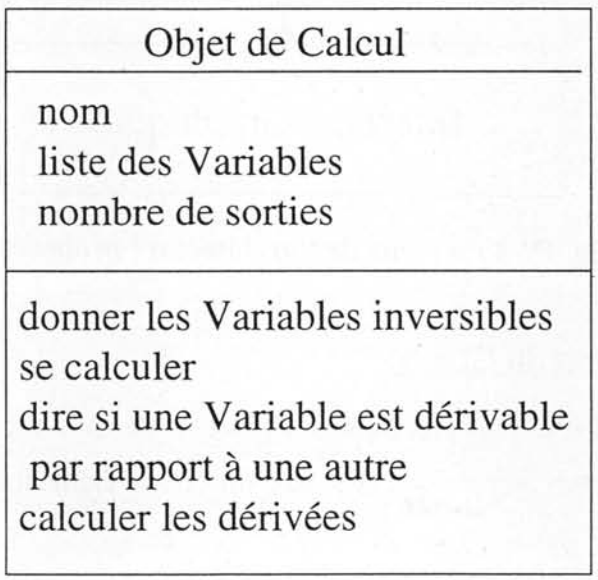


Figure IV-5 : Structure objet de l'Objet de Calcul

Les objets héritant d'Objet de Calcul doivent implanter les fonctions décrites dans cet objet. L'Objet de Calcul doit déclarer ce qu'il sait faire, et pouvoir le faire. Le « contrat » à remplir par un objet héritant de l'Objet de Calcul est :

- dire dans quels sens il est calculable, c'est à dire quels sont les jeux de paramètres de sortie qui peuvent être exprimés,

Par exemple pour une Equation, elle doit pouvoir donner les variables inversibles comme on les a définies au § 1.2.1.

- se calculer, dans les sens possibles,

Pour une Equation, on calculera symboliquement les variables inversibles.

- dire si les paramètres de sorties sont dérivables par rapport aux paramètres d'entrée, dans les sens possibles,
- calculer les dérivées partielles des paramètres de sortie en fonction des paramètres d'entrée, dans tous les sens possibles.

La figure IV-6 représente la façon dont fonctionne la hiérarchie des Objets de Calcul. L'Objet de Calcul générique définit le moule commun, les attributs que possèdent les sous-classes et les méthodes. Ces méthodes seront implantées de manière propre à chacune des classes dérivées.

Ainsi, la méthode appelée calcul dans Objet de Calcul sera implantée dans chaque sous-classe, comme Equation, mais de manière spécifique. La tâche remplie est la même mais elle se fait de manière particulière à l'Objet de calcul considéré. Pour l'utilisateur, ou le programme, l'appel à la fonction calcul d'un Objet de Calcul, fera appel à la bonne implantation de cette méthode dans la sous-classe correspondante. On peut donc manipuler des Objets de Calcul de manière générique, sans se préoccuper des calculs qui sont effectivement effectués derrière.

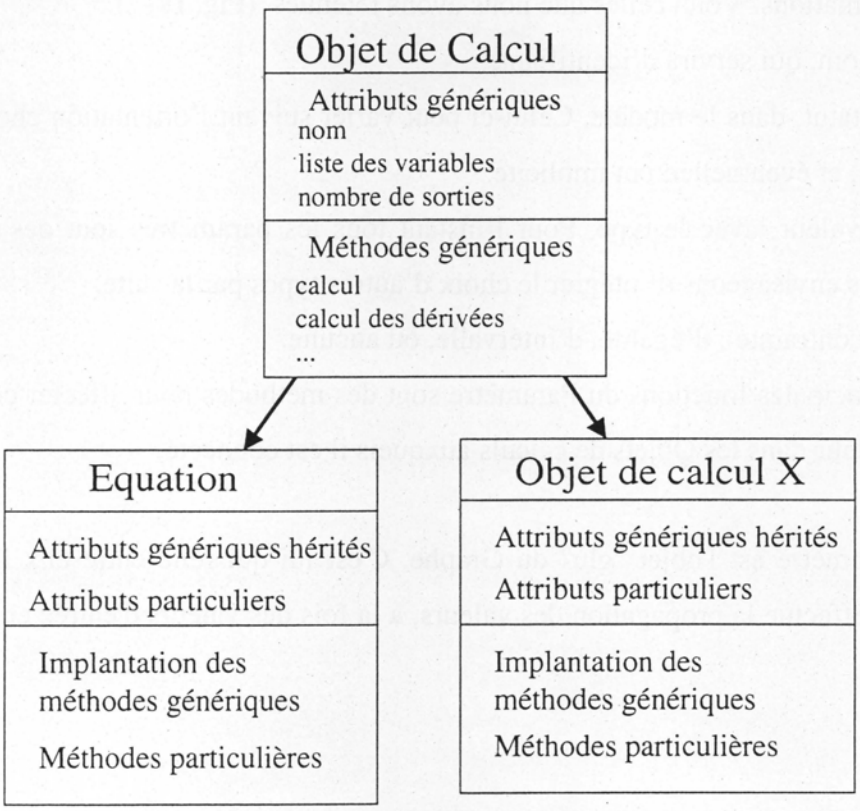


Figure IV-6 : Hiérarchie d'Objets de Calcul

Certaines des méthodes d'Objet de Calcul peuvent même être "abstraites", c'est à dire qu'elles ne possèdent pas de corps, et ne consistent en fait qu'en un contrat que doivent remplir toutes les classes héritées d'Objet de Calcul.

La façon dont sont faits les calculs par l'Objet de Calcul est donc peu importante au niveau de l'environnement. Seul lui importe que ces fonctions existent. Il existe un éventail de solutions techniques pour mettre en œuvre les calculs, ne serait-ce que pour les équations. Nous présenterons nos solutions par la suite, car elles dépendent pour une part des choix techniques effectués, notamment du langage informatique choisi.

2.3.2 Paramètres

La deuxième entité qui semble naturellement apparaître dans un modèle, après les équations, ou objets de calcul, est le paramètre. D'ailleurs, dans la plupart des modèles utilisés dans les différents outils cités dans le Chapitre 1 § 2.2.1, comme TK!Solver [KONO 84], Donald [SWIN 91] ou Cadacs [SALD 88], qui sont eux aussi des environnements objets, ce sont ces deux éléments qui sont considérés.

Comme nous l'avons signalé dans le cahier des charges, le paramètre doit porter plusieurs informations. Voici celles que nous avons retenues (Fig. IV-7) :

- le nom, qui servira d'identifiant,
- le statut, dans le modèle. Celui-ci peut varier suivant l'orientation choisie : entrée, sortie, inconnu, et éventuellement implicite.
- La valeur, avec le type. Pour l'instant tous les paramètres sont des réels double précision. Nous envisageons d'intégrer le choix d'autres types par la suite.
- La contrainte : d'égalité, d'intervalle, ou aucune.

Les principales fonctions du Paramètre sont des méthodes pour affecter ces champs et propager la valeur dans les Objets de calculs auxquels il est connecté.

Le Paramètre est l'objet "glu" du Graphe. C'est lui qui relie entre eux les Objets de Calcul, et qui effectue la propagation des valeurs, à la fois des valeurs d'entrée et de celles qui sont calculées.

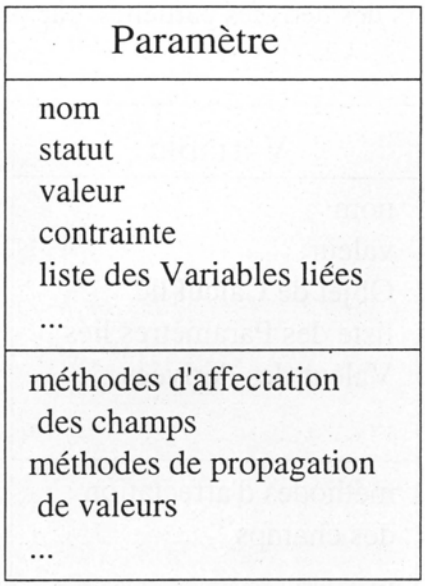


Figure IV-7 : Structure Objet du Paramètre

2.3.3 Variables

Plusieurs problèmes se sont posés cependant avec la seule structure Objets de calcul / Paramètres. Les paramètres peuvent être liés à plusieurs Objets de calcul, et cela pose des problème de gestion au niveau du passage des valeurs.

En effet, un paramètre peut être un paramètre de sortie du modèle tout en étant localement un paramètre d'entrée pour un Objet de calcul considéré. Par exemple, si l'on considère les équations $R=\rho \times L/S$ et $U=R \times I$. R est un paramètre de sortie de la première et un paramètre d'entrée de la seconde. Il faut donc distinguer les paramètres d'entrée/sortie du modèle et ceux, au niveau local, de l'équation considérée.

Ensuite, nous désirons pouvoir calculer une Equation, ou un Objet de Calcul, seuls, indépendamment du modèle, ce qui implique de découpler l'Objet de calcul et le Paramètre.

C'est ce que nous avons fait, en créant les objets Variables, liens entre les Objets de calcul et les Paramètres. La Variable permet de stocker temporairement l'information sur le calcul de l'Objet de calcul. Elle est intimement liée à l'Objet de Calcul et n'existe qu'à travers lui.

Les principaux attributs de l'objet Variable sont :

- le nom
- la valeur
- l'objet de calcul auquel appartient cette Variable
- la liste des Paramètres auxquels elle est liée.

- un tableau des valeurs des dérivées partielles, par rapport aux variables d'entrée de l'objet de calcul.

Variable
nom valeur Objet de Calcul lié liste des Paramètres liés Valeur des dérivées ...
méthodes d'affectation des champs ...

Figure IV-8 : Structure objet de la Variable

De même que pour le Paramètre, les principales fonctions de cet objet sont des fonctions pour fixer ou récupérer les valeurs de ses attributs.

Ainsi on rajoute comme attribut à l'objet Paramètre la liste des Variables auquel il est lié, de même que chaque Objet de calcul possède en attribut la liste de ses Variables. Les objets Variables liés à un Objet de calcul, seront créés automatiquement avec cet objet.

2.3.4 Graphe

Structure Objet

Comme nous l'avons dit plus haut, l'objet Graphe est l'objet pilote du modèle. Il contient donc l'ensemble des Objets de calcul, des Paramètres et des Variables du modèle, plus des attributs du modèle et des fonctions de gestion de celui-ci. Nous avons déjà donné plus ou moins sa structure objet (Figure IV-3), nous allons la re-préciser ici plus en détail (Figure IV-9).

Ses attributs sont :

- son nom, qui sert d'identifiant au modèle étudié,
- la liste des Objets de calcul,
- la liste des Paramètres.
- l'orientation du graphe si elle a été déterminée (nous détaillerons au chapitre suivant l'objet Orientation).

GRAPHE
nom liste des Objets de Calcul liste des Paramètres orientation ...
Ajouter / supprimer une équation ou un paramètre Calcul du nombre de degrés de liberté Donner la liste des Paramètres Donner la liste des Objets de Calcul Renvoyer un Objet de Calcul à partir de son nom Renvoyer un Paramètre à partir de son nom Donner l'orientation ...

Figure IV-9 : Structure objet du Graphe

On voit que la liste des Variables n'apparaît pas dans les champs, mais elle peut être retrouvée par le biais des Objets de Calcul, ou des Paramètres. En outre, les Variables sont transparentes pour l'utilisateur, qui ne les voit que comme connecteurs entre les Objets de calcul et les Paramètres, qui l'intéressent.

Les fonctions sur le Graphe sont typiquement la suppression ou l'ajout d'un Objet de Calcul ou d'un Paramètre. Le Graphe peut aussi donner des informations sur le modèle complet comme le nombre de degrés de liberté ou l'orientation du modèle. Il permet de récupérer et manipuler les objets depuis l'extérieur, c'est à dire par les autres objets de l'environnement.

Représentation

En résumé, la représentation graphique que nous avions donnée au § 1.1.1 pour le modèle simple de la résistance (EQ IV-1) donnait une assez bonne idée de l'objet Graphe lui-même, des objets qu'il contient et des liens qui lient ces objets.

Nous devons cependant y ajouter les objets Variables, liés aux Objets de Calcul, et donc aux Equations dans notre exemple (Figure IV-10). Sur notre graphe, nous avons noté les Variables avec la lettre minuscule du nom du Paramètre et le numéro de l'Objet de Calcul auxquels elles sont liées. Elles sont les liens, représentés par des lignes, qui peuvent être

orientées, entre les Objet de Calcul, représentés par des rectangles, et les Paramètres, représentés par des ronds.

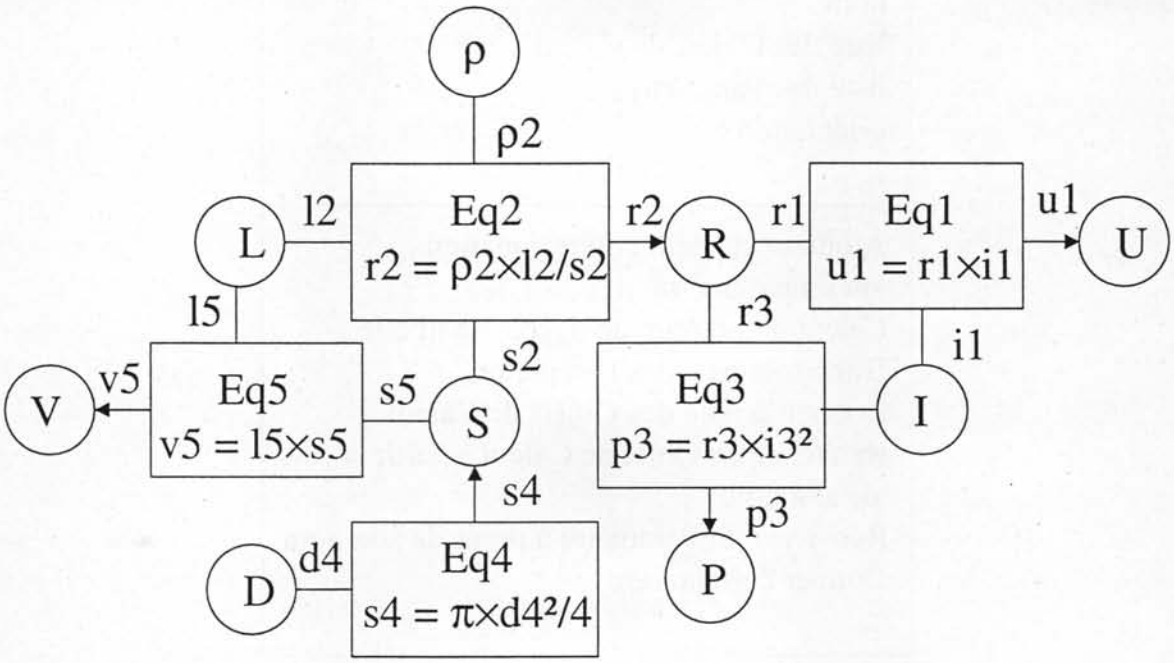


Fig. IV-10 : Représentation de l'objet Graphe, avec orientation

Les Equations ne voient que les Variables, et effectuent le calcul sur elles.

On voit bien ici, que la Variable est liée à un seul Objet de Calcul et à un seul Paramètre, alors que le Paramètre peut être lié à plusieurs Variables.

Conclusions

Nous avons présenté dans cette partie le cahier des charges décrivant les spécifications de notre environnement, c'est à dire les fonctions à réaliser pour stocker l'information sur le modèle et pour le manipuler en vue de le calculer et d'effectuer la propagation de contrainte, et l'optimisation. Nous avons ensuite décrit l'architecture générale objet proposée pour mettre en œuvre ce cahier des charges, en s'attardant sur la structure des données utilisée, car le modèle analytique est le cœur de notre approche. Des choix techniques sont à effectuer maintenant pour l'implantation effective de cette architecture. Nous allons donc présenter dans le chapitre suivant ces choix et leurs justifications, ainsi que l'implantation effectuée.

Chapitre 5. Implantation

Après avoir formalisé les aspect théoriques de notre approche, l'objectif est de détailler les réalisations informatiques effectuées, avec un premier prototype réalisé. Dans un premier temps, nous décrirons le travail de création de ce prototype, avec les solutions d'ordre informatique et technique qui ont été choisies. Nous présenterons ensuite, sur un exemple, un aperçu du fonctionnement de ce prototype. Nous détaillerons également le travail qui est encore à effectuer pour remplir complètement le cahier des charges posé dans la première partie du chapitre 4.

I – Description de l'architecture implantée

Dans le chapitre précédent nous avons spécifié les différentes composantes de notre environnement et esquissé l'architecture générale. L'implantation effective des objets dépend cependant pour une bonne part du langage informatique choisi. Ainsi, si en C++, on peut utiliser le mécanisme de double héritage, en JAVA, l'héritage est toujours simple, et par contre, on peut utiliser la notion d'interface. C'est pourquoi dans une première partie nous décrirons les raisons de notre choix : JAVA. Nous nous arrêterons ensuite sur les objets informatiques créés, qui ont posé des problèmes d'implantation, et les algorithmes utilisés dans la première version de l'environnement.

1.1. Choix du langage informatique : JAVA

JAVA est un langage faisant partie des nouvelles technologies émergentes. Bien adapté à Internet, ses possibilités ont jusqu'à présent été surtout utilisées dans ce domaine. Néanmoins, nous pensons qu'il mérite sa place dans les langage de développement objet, pour des applications plus « sérieuses » que des petits programmes, ou applets, dans une page HTML [INFOR].

1.1.1 Un langage à objets récent

JAVA est un pari de l'équipe Conception et Diagnostic Intégrés du LEG, qui a choisi de miser sur ce langage pour ses développements actuels, et vraisemblablement futurs.

C'est un langage récent, développé au départ pour des applications de domotique – le grille-pain intelligent, et qui a trouvé presque tout de suite son application sur le Web. Il est exploité aujourd'hui de plus en plus pour des applications distribuées, via Internet.

Il s'agit d'un langage à objets. En effet, JAVA est entièrement constitué d'objets, à la différence de C++, qui autorise la programmation procédurale. On dit souvent qu'il est à la

croisée de C++ et Smalltalk, et on dit aussi qu'il a éliminé beaucoup d'inconvénients de ces langages.

Pour notre application, nous avions surtout le choix entre JAVA et C++, les autres langages orientés objets nous étant moins connus, et demandant plus d'investissement pour un travail limité dans le temps. Bien que l'équipe ait beaucoup utilisé LeLisp jusqu'à récemment, à travers SMECI et les outils ILOG [ILOG], et que ce langage soit celui utilisé dans TK !Solver [KONO 84], il nous semblait plus indiqué pour des applications de type systèmes experts que pour l'environnement que nous avions en vue. JAVA nous demandait un investissement en formation, au contraire de C++, connu des ingénieurs généralistes.

1.1.2 Un langage offrant de nombreuses facilités

JAVA offre de nombreuses facilités, qui nous ont semblé intéressantes pour un développeur débutant, non expert en informatique. La presse spécialisée tend à dire que les temps de développement d'applications avec JAVA sont notablement réduits par rapport à C++ [INFOR].

JAVA offre en effet une bibliothèque de classes très bien conçue, les API – pour Application Programming Interface, qui permet de répondre à beaucoup de problèmes déjà rencontrés. Ainsi par exemple, pour concevoir des interfaces graphiques, il existe des objets fenêtre, menu, dialogue, etc.. tous prêts dans JAVA qui permettent de créer assez rapidement une interface agréable et conviviale.

Le gros avantage de JAVA est de permettre d'obtenir du code sans erreur beaucoup plus facilement qu'avec C++ [HORS 99]. La gestion de la mémoire n'est plus à la charge de l'utilisateur : JAVA possède un ramasse-miettes qui désalloue automatiquement les emplacements mémoire occupés par des objets qui ne sont plus utilisés. En outre, on ne manipule de pointeurs dans JAVA, cette notion étant transparente, de sorte que l'écrasement malencontreux d'une zone mémoire à cause d'un pointeur erroné n'est plus possible comme c'était souvent le cas en C++.

De plus, JAVA intègre la gestion des erreurs, les « threads », c'est à dire les processus multiples, des possibilités de sauvegarde et restauration des objets dans un état donné, la documentation automatique, etc...

JAVA est aussi un langage sûrement plus rapidement appréhensible que C++, donc un langage idéal pour un non-informaticien.

Il offre également la notion de « packages », c'est à dire d'ensembles d'objets regroupés, qui peuvent être réutilisés et mis en bibliothèque. Cette notion permet d'étendre

facilement des classes existantes, et de capitaliser des objets développés. Le développement des applications est donc largement facilité.

Un autre point intéressant pour nous est la possibilité de charger dynamiquement des classes. Ceci nous permettra par exemple, de charger une nouvelle équation dans un modèle.

1.1.3 Un langage facilitant l'interopérabilité et l'hétérogénéité des applications

Comme nous l'avons signalé précédemment, JAVA est utilisé pour le développement des applications distribuées sur Internet. Pour cela, JAVA est multi-plateformes, c'est à dire qu'un code développé en pur JAVA, pourra normalement tourner aussi bien sur Windows que sur Unix. En effet, JAVA utilise la notion de machine virtuelle, la JVM (Java Virtual machine), qui interprète le code JAVA compilé sous forme de code byte. Ceci est très intéressant pour la portabilité de notre environnement. En contrepartie, JAVA est souvent plus lent à l'exécution qu'un code compilé en natif.

En outre, il intègre des protocoles d'échanges comme RMI (remote method invocation) [HORS 99], et la norme CORBA [GEIB 98] pour pouvoir utiliser dans l'application des objets distants, distribués. Dans l'optique d'intégrer des objets hétérogènes comme des Objets de calcul numériques issus d'autre logiciels, cette propriété est très intéressante.

On pourra également utiliser des algorithmes connus, développés en FORTRANn ou en C, grâce au JNI (Java Native Interface).

Toutes ces possibilités nous ouvrent des horizons pour le développement et l'extension de notre environnement. JAVA ne semble pas être un phénomène de mode et offre de réels nouveaux outils pour la conception orientée objet.

1.2. Description de la création du Graphe

Nous allons décrire ici les objets du Graphe qui ont posé des problèmes ou ont nécessité des choix d'implantation. Les classes Paramètre, Variable, et Graphe sont des classes figées, dont on peut créer des instances sans aucun problème. Nous allons décrire ici les choix techniques qui ont été effectués pour le codage de l'information contenue dans l'équation, et que nous souhaitons exploiter dans l'environnement. Nous verrons ensuite ce que ces choix ont induit pour la génération effective des équations, et leur instanciation ainsi que celle du graphe, dans l'environnement.

1.2.1 Codage des équations

L'équation est un objet très particulier, car c'est celui qui est vraiment à la base de notre approche. Il est donc important que notre environnement prenne parfaitement en charge les équations et offre des facilités de manipulation pour ces objets.

L'utilisateur a la possibilité de créer et d'intégrer d'autres objets de calcul, différents d'équations, mais dans ce cas c'est à lui de gérer leur codage, en se conformant au moule de l'Objet de calcul, c'est à dire en implantant les méthodes définies par celui-ci.

Nous avons donc défini une classe générique Equation décrit dans le chapitre précédent (cf. Chapitre 4 § 2.3.1), classe fille d'Objet de calcul, qui sera le moule pour les équations des modèles analytiques à implanter. Cependant, cette implantation peut être effectuée de nombreuses façons différentes, suivant les choix techniques.

Implantation des calculs

Pour coder effectivement les équations entrées par l'utilisateur, comme par exemple

$$U = R \times I , \qquad \text{(EQ V-1)}$$

on a le choix :

- d'interpréter l'expression.

Dans ce cas, il faut stocker seulement les expressions que l'on veut calculer dans l'objet contenant l'équation, sous forme de chaînes de caractères "u=r*i" par exemple, et posséder un objet capable d'interpréter ces chaînes de caractère comme des formules mathématiques et de les calculer. Le code à générer dans les objets contenant les équations est alors petit, et on possède un objet générique pour le calcul des expressions mathématiques. Par contre, l'interprétation peut devenir longue à l'utilisation, si l'on calcule tous les objets du graphe de cette façon. Cette opération est coûteuse en temps de calcul.

Il existe de tels objets disponibles sur Internet par exemple [LUND], mais on ne possède pas la maîtrise des sources. On pourrait envisager d'en créer un.

- de coder en dur l'expression en JAVA.

Dans ce cas, il faut coder les expressions que l'on veut calculer sous forme de méthodes JAVA, dans l'objet contenant l'équation, ou dans des objets annexes, peu importe (mis à part le fait que l'on crée de nouvelles classes). Le code généré est alors beaucoup plus long mais les calculs sont beaucoup plus rapides car on utilise du code compilé.

En outre, cette implémentation ne nécessite pas d'objet externe ou de création d'objets trop spéciaux.

Nous avons choisi de coder "en dur" toutes les formules mathématiques utilisées dans l'environnement. La raison principale est la rapidité de calcul sur le graphe, une fois que le code est généré. Ensuite, cette implantation ne nécessite pas de module externe ou de développement particulier. Cependant, nous espérons que la génération même des classes des équations, contenant les expressions mathématiques liées, ne sera pas trop pénalisante en terme de temps et de taille du code généré.

Codage des expressions inverses d'une équation

Pour une équation, comme nous l'avons dit dans le chapitre précédent (cf. Chapitre 4 § 1.2.1), nous souhaitons pouvoir calculer chaque variable de façon symbolique lorsque cela est possible, de manière à pouvoir parcourir le graphe dans tous les sens possible et à modifier l'orientation.

Deux problèmes se posent :

- comment inverser les formules symboliquement ?
- doit-on coder systématiquement les expressions inverses, ou le faire au fur et à mesure des besoins ?

On a déjà décidé que ces expressions seraient codées en dur, comme toutes les expressions mathématiques concernant une équation donnée.

Pour inverser symboliquement une équation, nous avons deux possibilités :

- utiliser un objet de calcul symbolique en JAVA : de la même manière que pour l'interpréteur de formules mathématiques, on peut trouver ce genre d'objet (sans les sources et pas toujours gratuit) sur le Web, ou le coder soi-même. Cet objet risque d'être peu performant en face d'autres calculateurs symboliques non JAVA, plus anciens et mieux développés, comme Macsyma [MACSY] ou Maple [MAPLE].
- utiliser un logiciel de calcul symbolique extérieur à JAVA, tel que Macsyma ou Maple.

Pour des raisons de performances, nous avons choisi d'utiliser Macsyma, logiciel de calcul symbolique développé dans un langage proche du Lisp, pour effectuer les calculs symboliques sur les équations. En outre, c'est un logiciel déjà utilisé précédemment et donc bien connu (cf. Chapitre 3 § 2.2.3).

Nous avons donc développé dans Macsyma un module de génération de fichier, contenant les expressions mathématiques symboliques utiles d'une équation.

Dans ce contexte, il nous a paru plus judicieux de coder tous les sens possibles d'inversion d'une équation, de manière à ne passer qu'une fois dans ce module pour une équation donnée. Ceci peut s'avérer coûteux en taille de code généré, mais comme les équations sont de taille raisonnable et que les expressions symboliques ne sont pas substituées, ceci limite les expressions générées.

Codage des dérivées

Etant donné que nous souhaitons utiliser en priorité dans notre environnement, des modèles analytiques couplés avec des algorithmes d'optimisation de type gradient, le calcul des dérivées partielles des équations est nécessaire.

Comme nous l'avons décidé, ces expressions seront codées "en dur" en JAVA.

Nous avons besoin d'un dérivateur symbolique pour évaluer les expressions analytiques des dérivées partielles. Comme nous utilisons déjà Macsyma pour calculer les expressions inverses, nous allons l'utiliser aussi pour les dérivées, et nous coderons d'un seul coup tout ce qui est nécessaire.

Cependant, nous ne sommes pas obligés de calculer systématiquement les expressions symboliques des dérivées partielles, dans tous les sens inversibles de l'équation. En effet, nous pouvons utiliser le théorème des fonctions implicites que nous avons vu au chapitre 3 § 2.2.2 et qui est donné en Annexe A. Il suffit alors de coder les dérivées partielles de l'équation implicite, par rapport à toutes ses variables. Si on reprend l'équation (EQ V-1) :

$$U = R \times I , \qquad \text{(EQ V-1)}$$

l'équation implicite est donnée par :

$$f(U,R,I) = U - R \times I . \qquad \text{(EQ V-2)}$$

Ses dérivées partielles sont :

$$\frac{\partial f}{\partial U}(U,R,I) = 1 . \qquad \text{(EQ V-3)}$$

$$\frac{\partial f}{\partial R}(U,R,I) = -I . \qquad \text{(EQ V-4)}$$

$$\frac{\partial f}{\partial I}(U,R,I) = -R . \qquad \text{(EQ V-5)}$$

On en déduit par exemple la dérivée du paramètre U par rapport au paramètre I :

$$\frac{\partial U}{\partial I}(R,I) = -\frac{\frac{\partial f}{\partial I}}{\frac{\partial f}{\partial U}}(G(R,I),R,I) = R, \qquad \text{(EQ V-6)}$$

où G est la fonction, ici explicite donnant U en fonction de R et I (EQ V-1).

On peut donc ne coder que les dérivées de la fonction *f* symboliquement, et effectuer ensuite le calcul numérique de celles-ci pour obtenir les dérivées partielles des paramètres de l'équation.

On peut noter que cette méthode ne permet cependant pas de calculer les dérivées partielles des paramètres non inversibles de l'équation, sans évaluation de ceux-ci. Ainsi, dans l'équation (EQ V-6), si G était implicite, on ne pourrait évaluer la dérivée de U par rapport à I sans effectuer une résolution numérique de l'équation.

Néanmoins, dans un premier temps, comme nous disposons des expressions symboliques des variables inversibles, les expressions symboliques de leurs dérivées partielles sont simples à obtenir à l'aide de Macsyma. Nous avons donc choisi de les coder systématiquement, et de n'utiliser le théorème des fonctions implicites que pour le calcul des dérivées des variables non inversibles, dans le cas d'une résolution numérique de l'équation concernée.

1.2.2 Création des équations et du graphe

Les choix effectués précédemment induisent le développement d'un module sous Macsyma pour le calcul symbolique. Mais les informations issues de ce module doivent ensuite être intégrées à l'environnement pour créer effectivement les objets correspondants. Le passage par fichier nous a semblé ici naturel. Il faut donc un mécanisme dans l'application pour lire ce fichier, et créer les classes des équations entrées par l'utilisateur. Ensuite, comme ces classes sont créés dans l'environnement lui-même, il faut pouvoir en créer des instances dynamiquement, pour créer le graphe des équations. Nous décrivons ici ces mécanismes.

Création des équations

L'utilisateur qui veut créer ou modifier une équation doit passer par un mécanisme implanté dans Macsyma. Ce mécanisme prend en entrée :

- l'équation, sous sa forme implicite, c'est à dire que pour créer l'équation U=R×I, il faut entrer U-R×I,

- un nom donné par l'utilisateur, sous forme de chaîne de caractères,
- un répertoire et nom de fichier où les informations doivent être stockées,

Par exemple pour le modèle de résistance donné au Chapitre 1 § 2.1.2 (réutilisée au Chapitre 4 § 1.1.1), le fichier est le suivant :

```
NG_cc_gensys(
/* noms des equations */
["EqUn","EqDeux","EqTrois","EqQuatre","EqCinq"],
/* expressions des equations */
[u-r*i, r*s-rho*l, p-r*i^2, 4*s-%pi*Di^2, v-l*s],
/* nom du fichier que (/où si il existe deja) l'on va generer */
"resistance.txt",
/* chemin du fichier ou l'on va generer sous forme de liste */
[z,travail,modeles,resistance]          );
```

Le mécanisme Macsyma donne en sortie :

- la liste des variables,
- l'expression à annuler (*f* dans (EQ V-2)),
- les expressions des variables inversibles,
- les expressions des dérivées partielles des variables inversibles,

qu'il écrit dans un formalisme défini dans un fichier.

Par exemple pour l'équation EqUn de l'exemple précédent, on génère :

```
equation = EqUn {
variable = [i,r,u];
value = u-(i*r);
expression = [i=u/r, r=u/i, u=i*r];
derivative = [i(r=-(u/pow(r,2.0)),u=1/r), r(i=-(u/pow(i,2.0)),u=1/i), u(i=r,r=i) ];
}
```

Ce fichier peut être écrit directement par l'utilisateur s'il le désire, en respectant le formalisme demandé, mais cette opération est fastidieuse et source d'erreurs.

Ce fichier est ensuite interprété par JAVA dans l'application. L'interprétation est effectuée par un objet Parser (Interpréteur de fichier basé sur la définition d'une grammaire propre), créé à l'aide de l'utilitaire Java Compiler Compiler (JavaCC) de Sun. Celui-ci crée les classes héritées d'Equation, contenant ces équations, sous forme de fichiers JAVA, et les

compile ensuite (la compilation dynamique est possible, le compilateur JAVA étant lui aussi une classe JAVA).

Ces classes sont des classes filles de la classe Equation, abstraite. Elles implantent les méthodes de calcul effectif, qui sont les expressions symboliques codées en dur, décrites par la classe mère. On manipule ensuite dans l'environnement leurs instances comme des objets Equations, voir même comme des instances d'Objet de Calcul.

Création du graphe

Pour la création du graphe lui même, c'est à dire :

- créer une instance des objets qui composent le modèle (Equations, Paramètres et Variables)
- les relier entre eux,

nous avons créé une procédure par fichier.

Cette procédure est nécessaire car on ne peut manipuler de manière générique les classes correspondant aux équations. Les classes existent, mais seul l'utilisateur connaît le graphe et sait combien d'objets de chaque classe il doit instancier, et avec quels Paramètres il doit les relier. Seul l'utilisateur peut construire effectivement le graphe. La procédure par fichier nous a semblé la plus simple dans un prototype, mais on peut envisager dans l'avenir une interface graphique, qui permettrait de choisir dans une bibliothèque les classes disponibles pour les relier à des objets Paramètres, de manière physique, avec une représentation sous forme de boîtes et de connecteurs par exemple.

Le fichier est écrit par l'utilisateur dans un formalisme précis en mode texte. Il est ensuite interprété par l'application Java, par un objet qui permet de créer les Objets de calcul et leurs variables liées, les Paramètres et de les lier ensemble. Cet objet est lui aussi un Parser, écrit à l'aide de JavaCC (cf. § précédent) .

Reprenons par exemple la classe nommée EqUn, générée par le mécanisme précédent, et qui possède trois variables u, r et i. Pour créer une instance de cette équation, et la relier à des paramètres appelés respectivement Tension, Résistance et Intensité[1] du Graphe, l'utilisateur devra écrire la ligne suivante :

EqUn equation1 [(u, Tension), (r, Résistance), (i, Intensité)]

[1] Par convention dans l'application, on a choisi de mettre les noms des Variables en minuscules, alors que la première lettre des noms de Paramètres est une majuscule.

Il existe aussi un mécanisme par défaut, qui lie automatiquement les Variables de l'Equation dont on crée une instance à un Paramètre de même nom, mais dont la première lettre est une majuscule.

Ainsi :

EqUn equation1 ;

Equivaut à :

EqUn equation1 [(u, U), (r, R), (i, I)]

Les deux mécanismes sont utiles car l'utilisateur peut vouloir donner des noms explicites à ses paramètres, tout en utilisant des noms courts lorsqu'il crée les équations, pour éviter toute lourdeur. En outre, on peut vouloir créer deux instances d'une même classe Equation, en les liant à des Paramètres différents.

Pour créer les Objets de calcul, l'interpréteur est capable d'effectuer un chargement dynamique des classes si nécessaire, c'est à dire si la classe n'est pas déjà chargée en mémoire. Le mécanisme de génération des équations, décrit dans le paragraphe précédent, implique en effet que les classes des équations générées par l'utilisateur sont écrites, compilées et chargée au sein même de l'application. Il nous a donc fallu un mécanisme, pour les charger – c'est à dire intégrer – dans l'application une fois compilées et lorsqu'on a besoin d'en créer des instances.

1.3. Description des fonctionnalités implantées

Nous avons vu comment sont générés les classes et équations et comment est créé le Graphe. Nous allons maintenant détailler les fonctionnalités implantées.

1.3.1 Orientation et propagation de contraintes

Parcours du Graphe

Pour implanter les fonctionnalités de Propagation de contraintes et d'Orientation, il nous fallait trouver un algorithme de parcours de graphe. En effet, à chaque fois que l'utilisateur fixe un paramètre en tant que paramètre d'entrée, l'application doit être capable de déterminer quelles équations sont calculables, et ainsi de propager les valeurs, et d'orienter le graphe.

Un examen systématique de chaque équation peut s'avérer coûteux, et, qui plus est, inutile.

JAVA propose le mécanisme de "threads", qui sont des processus indépendants qui se lancent dans une application : on pourrait envisager d'associer un thread à chaque équation, qui détermine si l'équation est calculable, c'est à dire, si un nombre suffisant de paramètres ont été fixés en entrée et qui déclenche le calcul lorsque c'est le cas, puis s'arrête.

Une autre méthode est d'utiliser un algorithme, de parcours de graphe par exemple, dédié à ce genre de problème, et qui permette de déterminer quelles sont les équations susceptibles d'être résolues.

Nous n'avons pas voulu utiliser le mécanisme des threads, qui n'utilise qu'une intelligence locale et ne prend pas en compte l'ensemble du graphe, et qui peut s'avérer coûteux lui aussi, si le nombre d'équations devient important. Néanmoins, cette solution peut toujours être implantée par la suite.

Algorithme utilisé : Matrice d'occurrence

L'algorithme utilisé se base sur le calcul de la matrice d'occurrence du graphe. Il est issu de travaux destinés à la résolution de problèmes de gestion de contraintes sur des systèmes d'équations dans le domaine de la mécanique [SRID 93]. La matrice d'occurrence est une matrice donnant les dépendance entre les paramètres et les équations du modèle.

Nous avons donc créé une classe contenant cette matrice d'occurrence, ainsi que les fonctions de gestion de base de celle-ci (Fig. V-1). Les lignes de la matrice d'occurrence correspondent aux équations du modèle, les colonnes, aux paramètres. L'élément (i,j) de la matrice est égal à 1 si le jème paramètre est paramètre de la ième équation. Il est nul sinon.

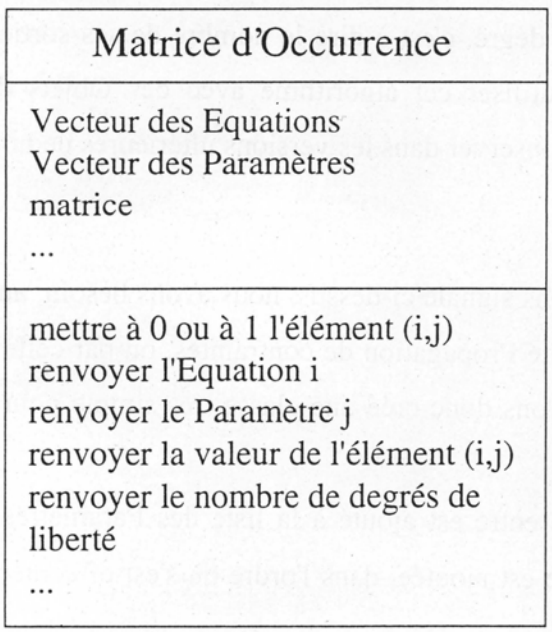


Fig. V-1 : Classe Matrice d'Occurrence

Le principe du parcours de graphe à l'aide de la matrice d'occurrence, proposé dans l'article [SRID 93], est simple : à chaque paramètre fixé, on annule la colonne correspondante, et on détermine les équations susceptibles d'être calculables. Ce sont les équations qui ont pour paramètre le paramètre qui vient d'être fixé. On n'examine donc pas systématiquement toutes les équations du modèles, mais seulement celles autour de la valeur à propager. Elles sont classée par degré de "solvabilité" et parmi ces équations, on cherche une équation effectivement calculable, en s'occupant d'abord de celle dont le degré de solvabilité est le plus élevé. Dès qu'on a trouvé une équation qui peut être calculée, on la calcule dans le cas de la propagation de contraintes et on la rajoute à l'objet contenant l'Orientation du Graphe.

En effet, l'étape de propagation ou d'orientation permet d'orienter le graphe et d'enregistrer une orientation qui sera ensuite réutilisée pour le calcul et l'optimisation. Nous avons donc créé une classe permettant de stocker l'orientation. Nous la décrirons ci-après.

Pour effectuer les étapes de l'algorithme de parcours de graphe, basé sur l'utilisation d'une matrice d'occurrence, nous avons créé un objet appelé "OccurrenceSolver", qui contient cet algorithme, et dont le constructeur prend en paramètre un objet Matrice d'Occurrence. Sa méthode principale est celle qui doit être appelée par l'application à chaque fois que l'utilisateur fixe un Paramètre d'entrée.

Nous avons étendu l'algorithme de parcours de graphe à l'aide d'une matrice d'occurrence à des objets dont le nombre de sorties est supérieur à 1, c'est à dire à des objets autres que des équations, en effectuant un test avec un système de deux équations, donc un objet dont le nombre de sorties est 2. Nous avons pour cela associé à chaque objet dans la matrice d'occurrence, son degré, c'est à dire le nombre de ses sorties justement. Nous avons donc espoir de pouvoir utiliser cet algorithme avec des Objets de Calcul autres que les Equations, et donc de le conserver dans les versions ultérieures de notre environnement.

Classe Orientation

Comme nous l'avons signalé ci-dessus, nous avons besoin, au cours de l'orientation du graphe par la fonctionnalité Propagation de contraintes, ou par celle d'Orientation, de stocker cette orientation. Nous avons donc créé une classe très simple pour effectuer cela : la classe Orientation (Fig. V-2).

Chaque paramètre entré est ajouté à la liste des Paramètres d'entrée de l'orientation. Chaque Equation calculée est ajoutée, dans l'ordre où s'est effectuée la résolution, au vecteur des Equations.

Orientation
Liste des Paramètres d'entrée Vecteur des Equations (dans l'ordre de calcul) ...
renvoyer la liste des Paramètres d'entrée renvoyer le Vecteur des Equations ajouter une Equation ajouter un Paramètre d'entrée ...

Fig. V-2 : Classe Orientation

L'orientation n'est considérée valide que lorsque toutes les équations ont été résolues. Elle est alors associée au Graphe, et le Graphe est déclaré orienté. Nous avons donc ajouté deux attributs à l'objet Graphe : un objet Orientation contenant l'Orientation du Graphe lorsqu'elle est effectuée et un booléen indiquant si le Graphe est orienté.

Toute modification dans le statut d'un des Paramètres du Graphe implique la remise à zéro de l'objet Orientation du Graphe.

1.3.2 Solveur et Optimisation

Le Solveur et l'Optimisation sont les deux fonctionnalités qui s'effectuent sur le Graphe orienté. Seul le Solveur a été implanté pour l'instant, ainsi que le calcul des gradients.

Solveur

Le solveur s'effectue sur le graphe orienté. La seule difficulté est de récupérer l'Orientation du Graphe et d'effectuer les calculs le plus rapidement possible.

Etant donné, comme on l'a vu au paragraphe précédent, que l'orientation, lorsqu'elle est faite, est stockée dans le Graphe, il suffit de récupérer les Paramètres d'entrée fournis par cette orientation, et d'en demander la valeur à l'utilisateur, pour ensuite calculer les Equations dans l'ordre indiqué par l'orientation.

Ce calcul peut être réitéré autant de fois que désiré par l'utilisateur, avec des valeurs différentes pour le jeu de Paramètres d'entrée. On effectue donc bien le calcul d'analyse.

Comme nous utilisons des équations analytiques, codées "en dur" dans JAVA, ce calcul est très rapide.

Optimisation

La fonctionnalité Optimisation n'a pas encore été implantée. Mais nous avons introduit également dans la fonctionnalité Solveur le calcul des dérivées des Paramètres de sortie par rapport aux Paramètres d'entrée du Graphe, afin de valider la méthode de propagation des valeurs des dérivées partielles dans le Graphe. Nous aurons en effet besoin de ces valeurs de dérivées pour effectuer l'optimisation du modèle à l'aide d'un algorithme de type gradient. Ce calcul est la plus grande difficulté posée par l'optimisation, à partir du moment où l'on dispose d'un algorithme. Nous pensons donc que le fait de pouvoir calculer les dérivées des Paramètres de sortie par rapport aux Paramètres d'entrée valide le fait que l'on puisse, par la suite, intégrer un algorithme d'optimisation et effectuer le dimensionnement sous contraintes du modèle analytique.

La propagation des valeurs des dérivées s'effectue à peu près de la même manière que la propagation des valeurs des Paramètres. Au niveau local, la Variable de sortie de l'Equation (qui est orientée) possède un attribut contenant la valeur de ses dérivées partielles par rapport aux autres Variables. Le Paramètre qui lui est lié va, lui, contenir les valeurs de ses dérivées par rapport aux paramètres d'entrée du Graphe, qui sont calculées par composition, suivant que les Paramètres liées aux Variables d'entrée de l'Equation sont ou non des Paramètres d'entrée du modèle (Fig. V-3).

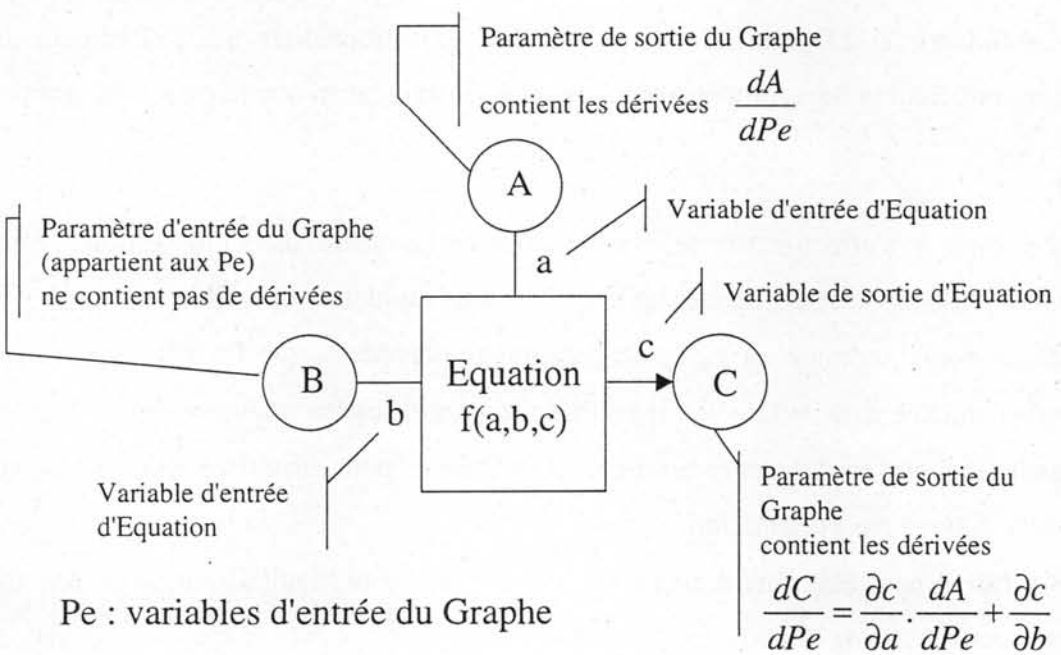


Fig. V-3 : Propagation des valeurs des dérivées dans une Equation

1.4 Architecture

Nous allons présenter en bilan de cette partie le prototype réalisé, c'est à dire l'architecture effective implantée, à partir du cahier des charges donné dans la première partie du Chapitre 4, et des choix techniques effectués que nous avons décrits dans la partie précédente. Cette architecture est présentée sur la Figure V-4.

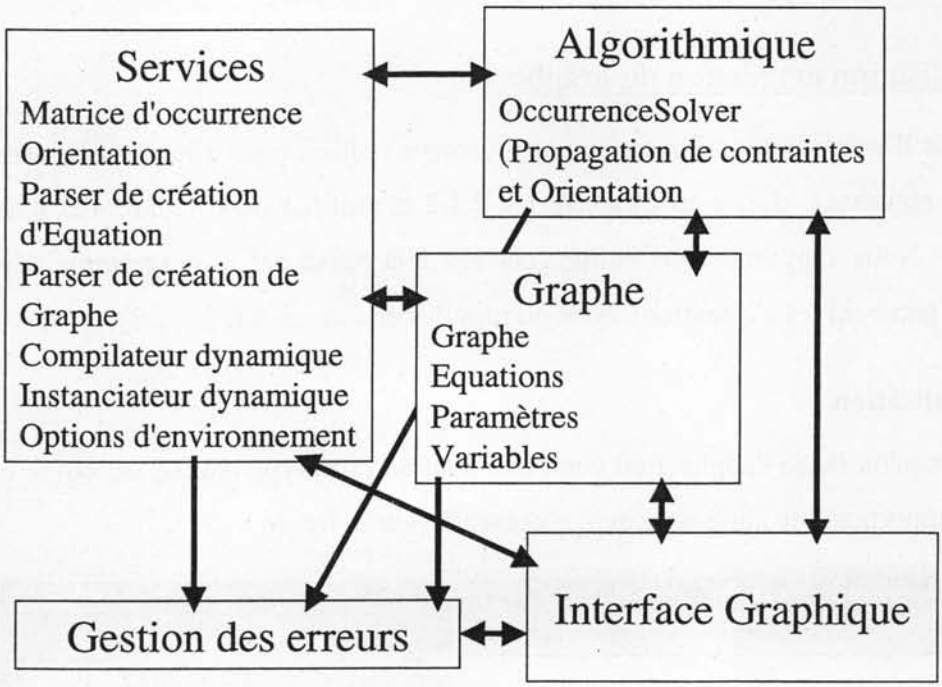


Fig. V-4 : Architecture du prototype implanté

On retrouve sur cette figure toutes les classes que nous avons décrites précédemment, au Chapitre 4 § 2.3 et dans le début de ce Chapitre.

Les Options d'environnement permettent de stocker les répertoire de génération des classes JAVA, de compilation et les répertoires utiles à la compilation en JAVA (Classpath) contenant les bibliothèques des classes spécifiques développées par SUN, l'inventeur de JAVA [SUN] utilisées dans l'application, ainsi que les noms des fichiers : celui issu de Macsyma et celui de création du Graphe.

La gestion des erreurs est spéciale dans JAVA. Chaque classe peut "lancer" des erreurs, qui doivent ensuite être prises en compte à un moment ou à un autre dans l'application. Cette gestion nous permet de faire remonter au niveau de l'interface graphique toutes les erreurs qui peuvent se produire tant à la construction du Graphe, qu'en cours de calcul.

Le niveau que nous avions appelé "Interface non-graphique" dans la présentation de l'architecture au Chapitre 4 § 2.2.2 se retrouve en fait au niveau de chaque objet lui-même.

II – Présentation du fonctionnement du prototype

Nous allons présenter ici sur un exemple simple le fonctionnement du prototype, en ce qui concerne la création du graphe et des objets qui le composent, et les fonctionnalités dont on dispose. Nous effectuerons ensuite le bilan et la validation du travail effectué par rapport au cahier des charges posé et du travail qui reste à faire

2.1 Initialisation et création du graphe

Pour illustrer le fonctionnement du prototype réalisé, nous allons utiliser l'exemple du modèle de résistance, donné au Chapitre 1 § 2.1.2 et réutilisé au Chapitre 4 et dans la partie précédente. Nous supposons que l'utilisateur est déjà passé par le programme développé en Macsyma, générant les expressions symboliques des équations (cf. § 1.2.2).

2.1.1 Initialisation

Lorsqu'on lance l'application correspondant au prototype réalisé, on ouvre une simple fenêtre, comportant une barre de menu, représentée sur la figure V-5.

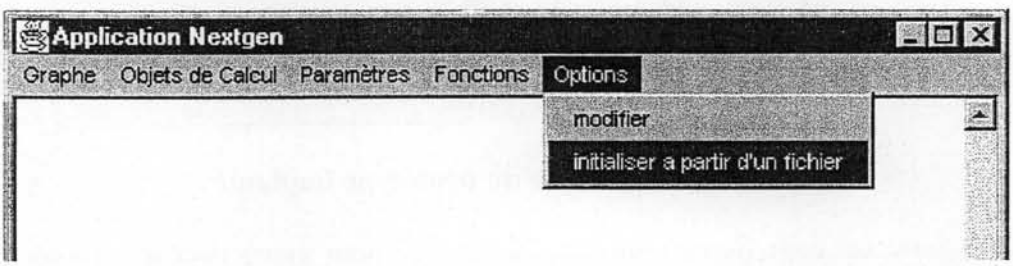


Fig. V-5 : Fenêtre principale du prototype – menu Options

Le passage des informations se fait, comme nous l'avons décrit précédemment, par fichiers. Le fichier issu de macsyma contient les expressions mathématiques des équations et de leur dériées partielles. Il permet de créer les classes Equation contenant ces informations. le fichier de déclaration du graphe écrit par l'utilisateur permet d'en créer des instances et de les relier aux Paramètres correspondant.

Pour stocker les informations sur l'emplacement de ces fichiers, et des répertoires utiles de l'application, nous avons créé (cf. § 1.4) une classe Options. Elle peut être initialisée au moyen d'un autre fichier écrit par l'utilisateur, du type :

```
Repertoire de compilation   z:\\bin\\classes
Fichier du modele     z:\\travail\\testmapse\\resistance.txt
Fichier du Graphe     z:\\travail\\testmapse\\declare.txt
Repertoire de generation      z:\\travail\\testmapse\\user
ClassPath     z:\\bin\\classes;j:\\jdk1.2\\lib\\tools.jar
Precision des calculs  1e-4
```

Lorsqu'on sélectionne le sous-menu "Initialiser à partir d'un fichier" (Fig . V-5) dans le menu "Options", on peut entrer le répertoire et le nom de ce fichier pour effectuer l'initialisation des Options (Fig. V-6)

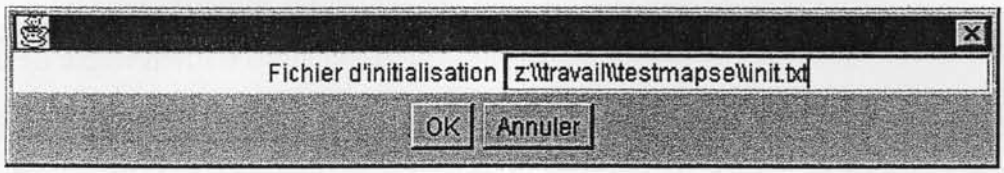


Fig. V-6 : Saisir le fichier d'initialisation

Ces Options peuvent à tout moment, lorsque la fenêtre principale est accessible être modifiées, en sélectionnant le sous-menu "modifier" du menu "Options" (Fig. V-5). On obtient alors une fenêtre permettant de saisir les informations correspondantes (Fig. V-6).

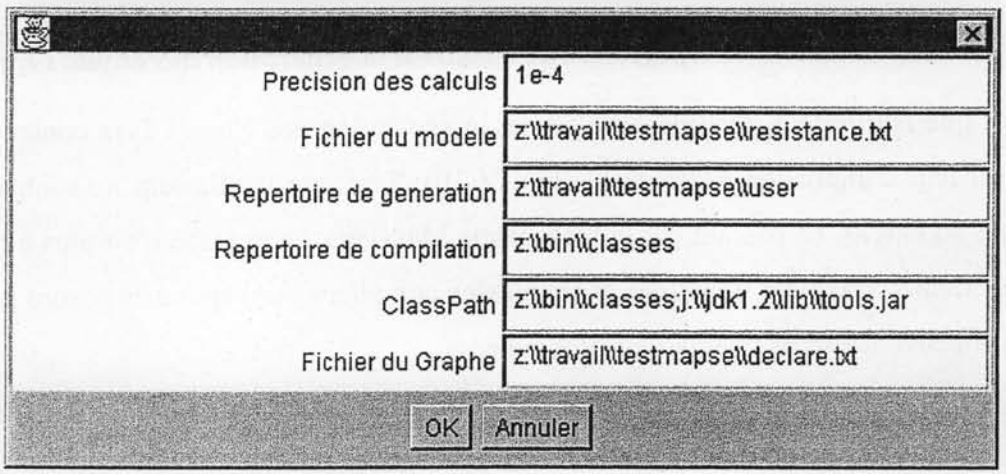


Fig. V-7 : Modifier les Options

2.1.2 Création des classes des équations à partir du fichier issu de Macsyma

Pour créer les classes d'Equations correspondant aux équations du modèle que veut manipuler l'utilisateur, il faut lire le fichier issu de Macsyma contenant ces équations, leurs expressions et les expressions des dérivées partielles (cf. § 1.2.2). Ceci s'effectue en sélectionnant le sous-menu "créer" du menu "Objets de Calcul" de la fenêtre principale (Fig. V-8)

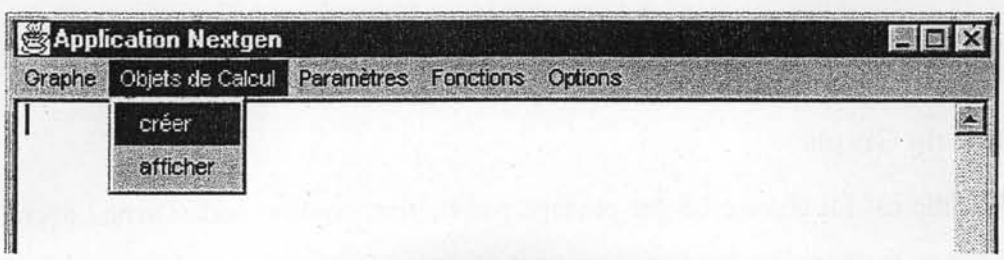


Fig. V-8 : Fenêtre principale du prototype – menu Objets de Calcul

La fenêtre qui s'ouvre permet de saisir les chemins des fichiers nécessaire à la génération des objets de calcul, c'est à dire le fichier issu de Macsyma contenant le modèle, le répertoire où seront généré les fichiers Java correspondant, le répertoire où les classes compilées seront stockées (pour être chargées par la suite) et les librairies nécessaires à cette compilation dynamique. Les chemins stockés dans la classe Options s'affichent par défaut.

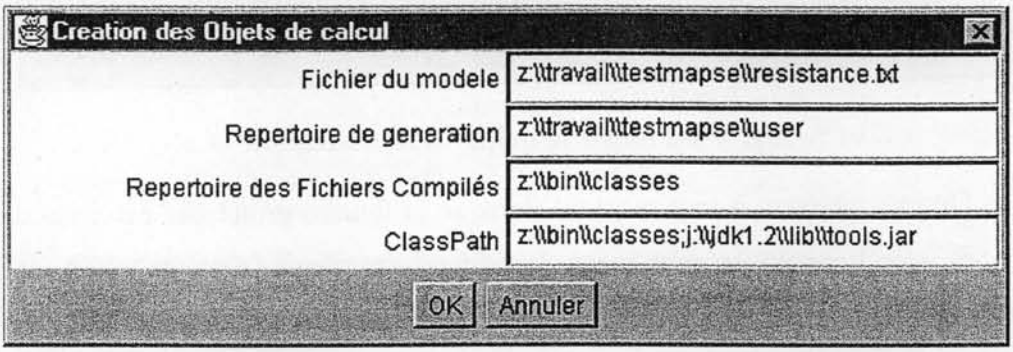


Fig. V-9 : Saisir les fichiers et répertoires nécessaire à la génération des objets Equations

Dès que l'utilisateur a validé cette saisie, la génération des classes Java contenant les équations et leur compilation s'effectuent (Fig. V-10). Tant que l'utilisateur ne souhaite pas modifier les équations, en passant par le mécanisme Macsyma, cette étape n'est plus à refaire, puisque les fichiers Java des équations et les classes compilées correspondantes sont générés et stockés et peuvent être récupérés sans problème.

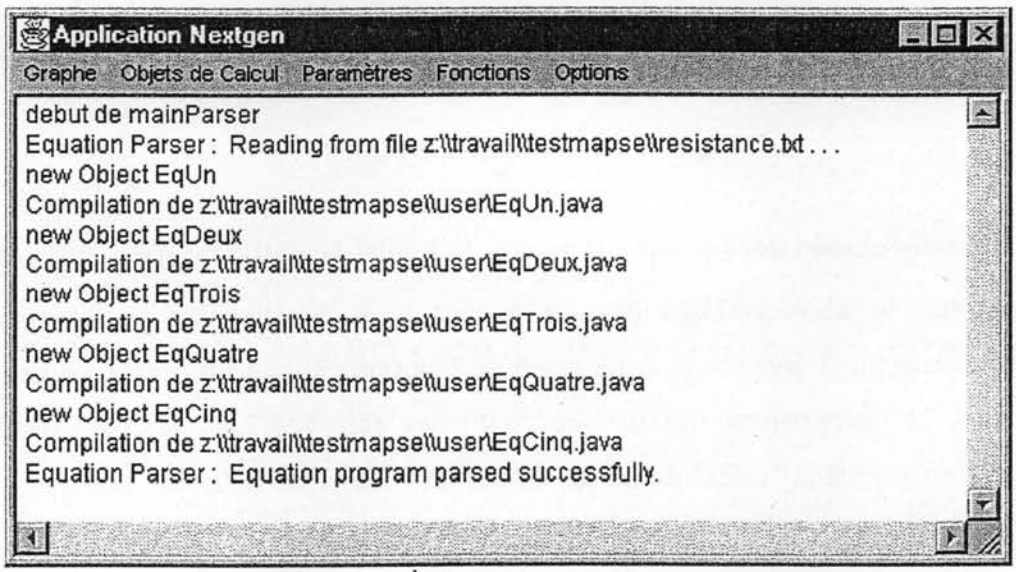


Fig. V-10 : Session de génération du modèle en classes Java

2.1.3 Création du Graphe

Le Graphe est lui-aussi créé par passage par fichier, comme nous l'avons décrit au § 1.2.2., de la même manière que les équations sont générées

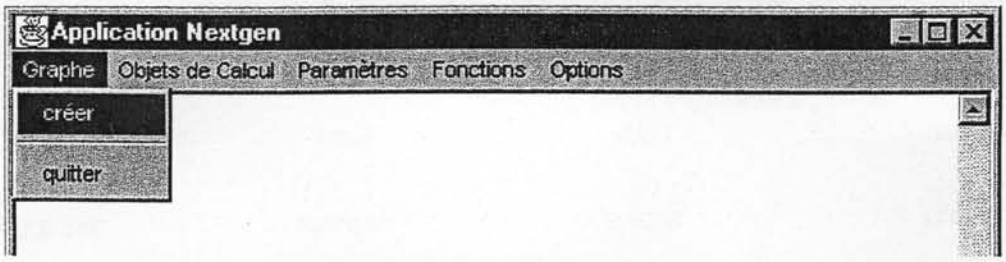


Fig. V-11 : Fenêtre principale du prototype – menu Graphe

En sélectionnant le sous-menu "créer" du menu "Graphe" dans la fenêtre principale (Fig. V-11), on ouvre une fenêtre de saisie du nom donné au modèle ainsi que des fichiers et répertoires utiles pour la création du graphe, c'est à dire le fichier contenant la déclaration des Equations à instancier et des Paramètres liés (cf. § 1.2.2) et le répertoire où se trouve les classes compilées de ces Equations (Fig. V-12). Les chemins stockés dans la classe Options s'affichent par défaut.

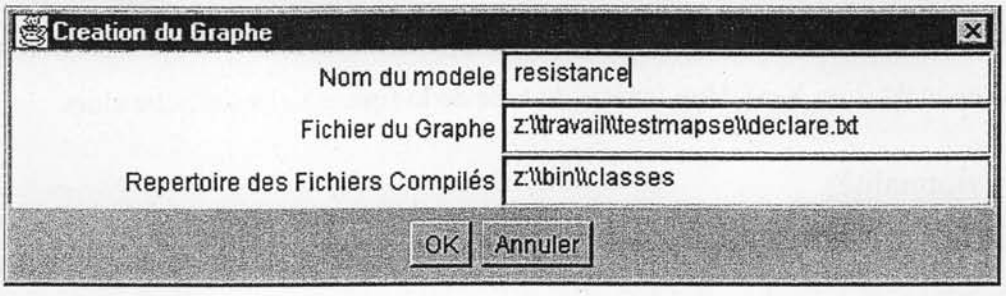


Fig. V-12 : Saisir les fichiers et répertoires nécessaire à la création du Graphe

Dès que l'utilisateur a validé cette saisie, les instances des Equations et des Paramètres du Graphe sont créées et reliées entre elles (Fig. V-13).

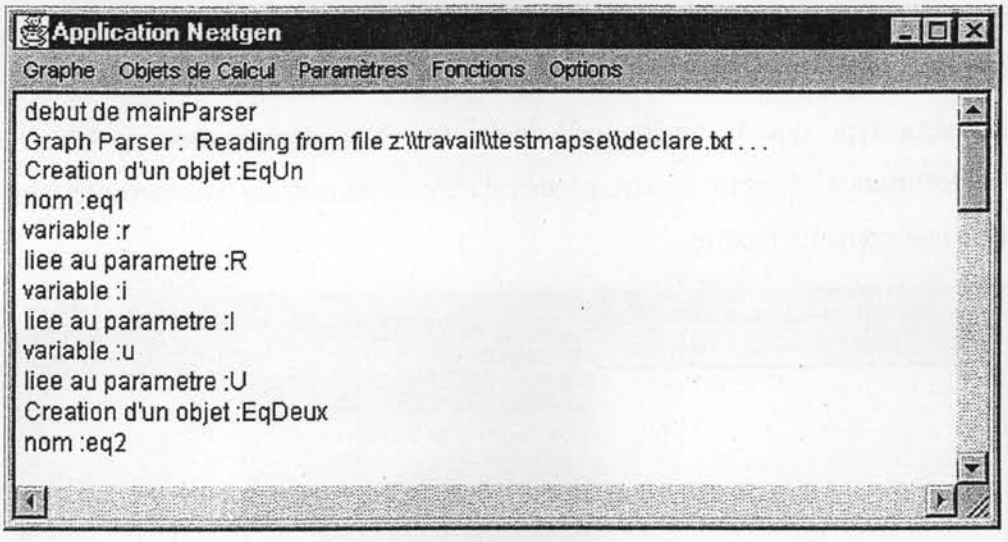


Fig. V-13 : Session de création du Graphe

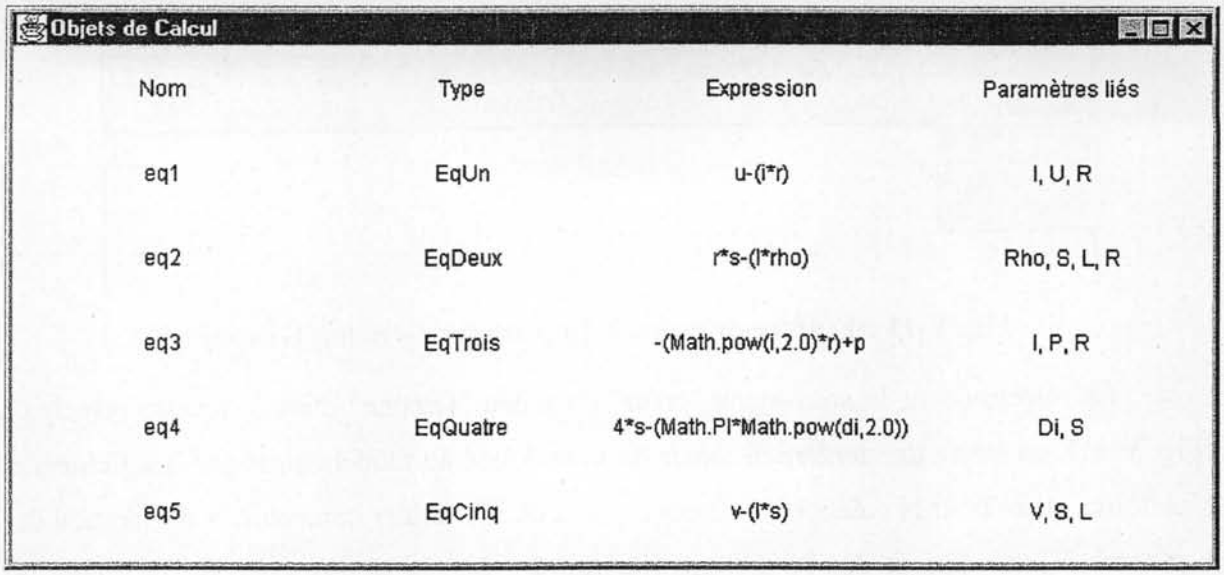


Fig. V-14 : Visualisation des Equations

On peut à tout moment ensuite, lorsque la fenêtre principale est accessible, visualiser le modèle créé, en sélectionnant le sous-menu "afficher" du menu "Objets de Calcul" de la fenêtre principale (Fig. V-8). Une fenêtre du type de la figure V-14 s'affiche alors.

2.2 Fonctionnalités

Après que le modèle a été généré, l'utilisateur souhaite pouvoir effectuer la propagation de contraintes sur celui-ci, et le calculer.

2.2.1 Orientation et propagation de contraintes

Les sous-menu "Orientation" et "Propagation de contraintes" du menu "Fonctions" de la fenêtre principale permettent d'ouvrir les fenêtres correspondant à ces deux fonctionnalités (Fig. V-15). On obtient alors pour les deux fonctions une fenêtre avec un menu identique (Fig. V-16), de même type que la fenêtre principale, qui, elle, devient inaccessible. Le menu "Matrice d'Occurrence" de cette fenêtre permet d'afficher la matrice d'occurrence générale du Graphe dans une nouvelle fenêtre.

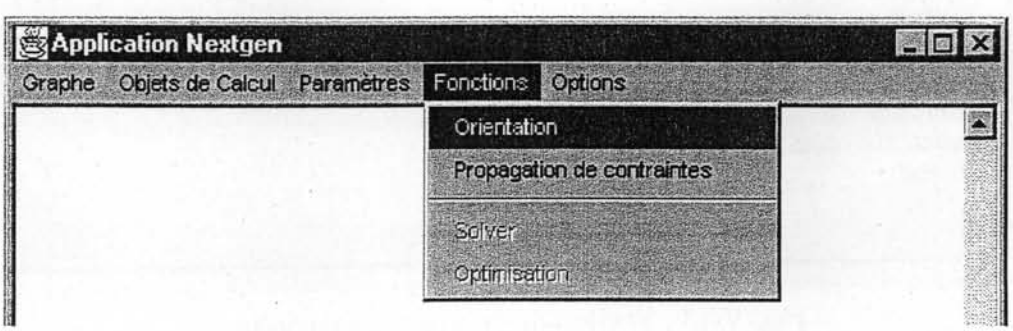


Fig. V-15 : Fenêtre principale du prototype – menu Fonctions

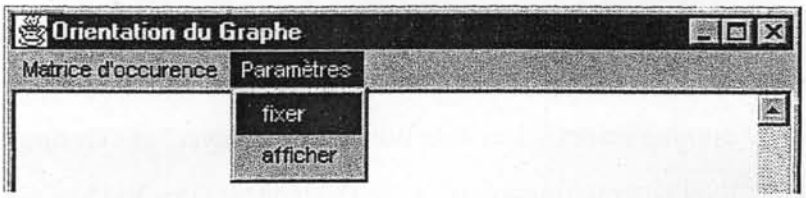


Fig. V-16 : Fenêtre de la fonction Orientation – menu Paramètres

Le sous-menu "fixer" du menu "Paramètres" permet à l'utilisateur de fixer les Paramètres d'entrée désirés. Selon que l'on effectue une orientation du Graphe ou une Propagation de valeurs sur les paramètres, la fenêtre qui s'ouvrira permettra ou non de rentrer une valeur pour le Paramètre fixé (Fig. V-17 et V-18).

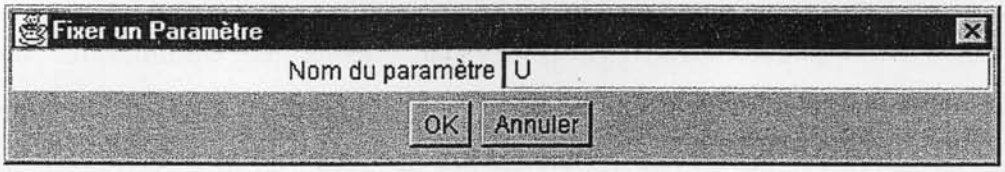


Fig. V-17 : Saisie d'un Paramètre d'entrée pour l'Orientation

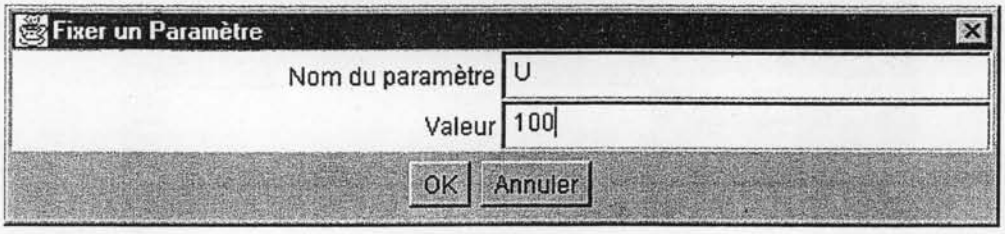


Fig. V-18 : Saisie d'un Paramètre d'entrée pour la Propagation

A chaque saisie d'un paramètre d'entrée, la classe "OccurenceSolver" est appelée et les équations qui peuvent être résolues sont trouvées (Fig V-19).

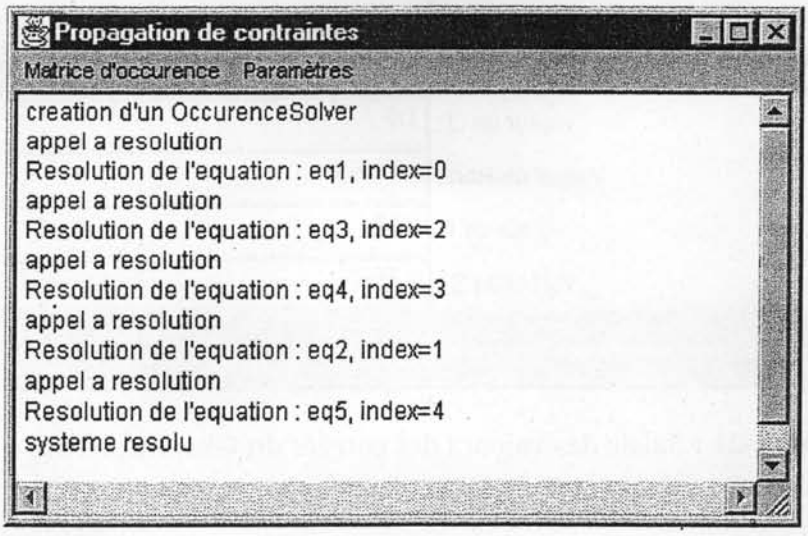


Fig. V-19 : Session de résolution du modèle par Propagation de valeurs

Le calcul n'est effectué que si l'on est dans la fonction Propagation. Lorsque tout se déroule bien, la mention "système résolu" est affichée, l'orientation est enregistrée et le Graphe est considéré comme orienté. Les fonctionnalités "Solver" et "Optimisation" du menu "Fonctions" qui étaient auparavant inaccessibles le deviennent (Fig.V-15).

En cas d'erreur dans les processus d'Orientation ou de Propagation de contraintes, un message est affiché. L'utilisateur ne peut plus rien faire dans la fenêtre, il doit retourner à la fenêtre principale pour éventuellement réinitialiser le Paramètres ou en modifier certains (sous-menus du menu "Paramètres" de la fenêtre principale Fig. V-15).

2.2.2 Solver et calcul des dérivées

Lorsque le Graphe est orienté, les sous-menus "Solver" et "Optimisation" du menu "Fonctions" de la fenêtre principale sont accessibles. L'Optimisation n'est pas implantée pour l'instant. La sélection de "Solver" ouvre une fenêtre de Calcul du modèle (Fig. V-20) et la fenêtre principale devient inactive.

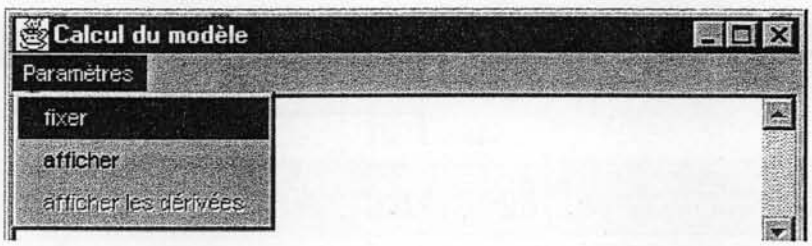


Fig. V-20 : Fenêtre de la fonction Solver – menu Paramètres

Le sous-menu "fixer" du menu "Paramètres" permet de fixer les valeurs des entrées du Graphe orientées. Les valeurs par défaut sont affichées (Fig. V-21), si elles existent (par exemple lorsqu'on a effectué une propagation de contraintes).

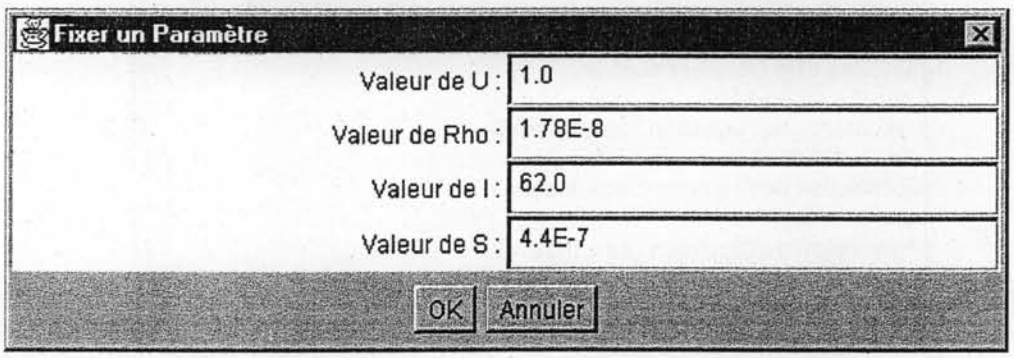
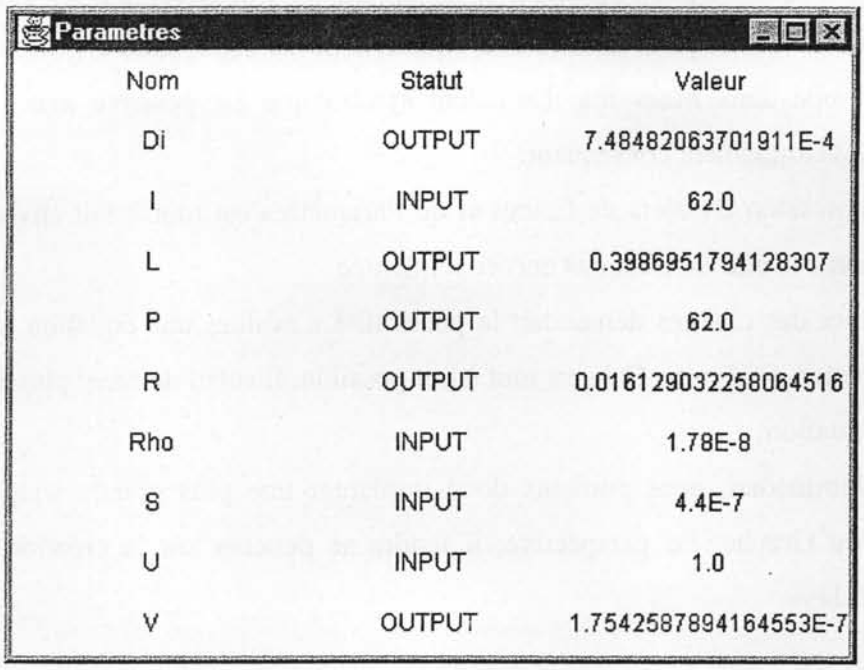


Fig. V-21 : Saisie des valeurs des entrées du Graphe orienté

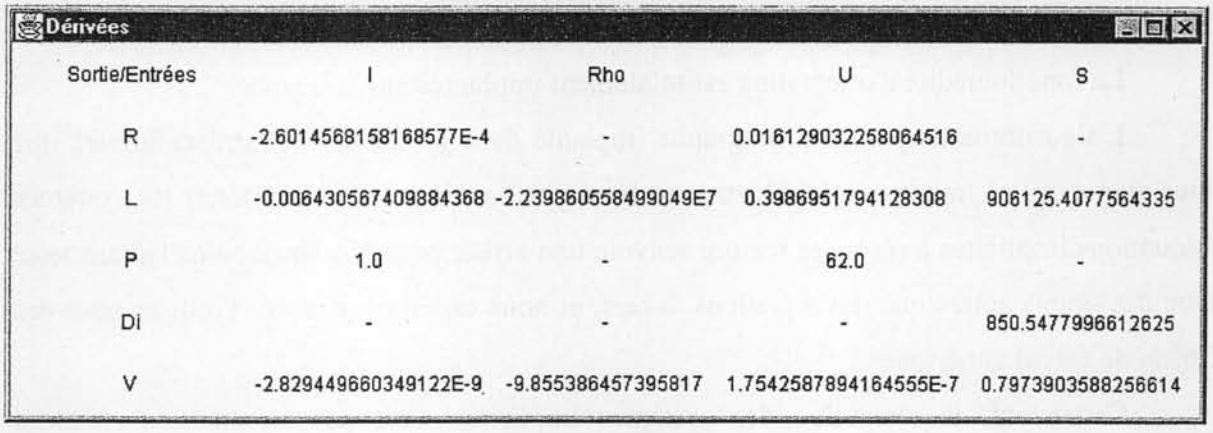
La saisie effectuée, le calcul des sorties et de leurs dérivées par rapport aux paramètres d'entrée s'effectue. La mention "Calcul terminé" apparaît dans la fenêtre. L'utilisateur peut alors afficher les valeurs des Paramètres (Fig. V-22) et des dérivées (Fig. V-23).



Parametres

Nom	Statut	Valeur
Di	OUTPUT	7.48482063701911E-4
I	INPUT	62.0
L	OUTPUT	0.3986951794128307
P	OUTPUT	62.0
R	OUTPUT	0.016129032258064516
Rho	INPUT	1.78E-8
S	INPUT	4.4E-7
U	INPUT	1.0
V	OUTPUT	1.7542587894164553E-7

Fig. V-22 : Visualisation des Paramètres



Dérivées

Sortie/Entrées	I	Rho	U	S
R	-2.6014568158168577E-4	-	0.016129032258064516	-
L	-0.006430567409884368	-2.239860558499049E7	0.3986951794128308	906125.4077564335
P	1.0	-	62.0	-
Di	-	-	-	850.5477996612625
V	-2.829449660349122E-9	-9.855386457395817	1.7542587894164555E-7	0.7973903588256614

Fig. V-23 : Visualisation du tableau des dérivées

Le tiret (-) dans la fenêtre de visualisation des dérivées (Fig. V-23) signifie que cette dérivée est nulle.

2.3 Satisfaction du cahier des charges et perspectives

Nous allons effectuer ici le bilan du travail effectué par rapport au cahier des charges, posé dans la première partie du Chapitre 4.

2.3.1 Gestion du modèle

Nous avons implanté la création du modèle, avec une structure objet.

La création de nouveaux éléments du modèle est possible, mais nécessite de repasser par toutes les étapes, ce qui peut être fastidieux pour l'utilisateur. Il faudrait ajouter la

possibilité de ne rajouter qu'une équation au graphe. Ceci n'implique pas de travail important à effectuer. L'utilisateur, pour ajouter une équation devra cependant toujours passer par le module développé dans Macsyma. Le calcul symbolique est possible avec Java, mais il implique un développement conséquent.

La suppression d'Objets de Calcul et de Paramètres est tout à fait envisageable dans l'immédiat, même si elle est n'est pas encore implantée.

Le cahier des charges demandait la possibilité d'évaluer une équation avec plusieurs jeux de paramètres différents. Cela est tout à fait possible. Il suffit de créer plusieurs instances d'une même équation.

Dans l'immédiat, nous pouvons donc implanter une plus grande souplesse dans la modification du Graphe. En perspective, il faudra se pencher sur la création d'un solveur symbolique en Java.

2.3.2 Fonctionnalités

Propagation de contraintes

La fonctionnalité d'orientation est totalement implantée sur le Graphe.

L'algorithme de parcours de graphe implanté dans la classe "OccurrenceSolver" que nous avons utilisé fonctionne très bien avec les équations. Il permet de détecter les systèmes d'équations implicites à résoudre (ce qui renvoie une erreur pour l'instant). Nous l'avons testé avec des objets autres que des équations, à part, et nous espérons, pouvoir l'utiliser pour des Objets de calcul génériques.

Cependant, la résolution des systèmes implicites n'est pas effectuée. Il faudrait implanter un algorithme de résolution des systèmes non linéaires. Le calcul des dérivées étant disponibles, on peut envisager d'utiliser un algorithme similaire à celui présenté dans le Chapitre 3.§ 2.2.

En ce qui concerne la propagation proprement dite, nous n'avons implanté pour l'instant que ce qu'on pourrait appeler de la "propagation de valeurs". En effet, la prise en compte de contraintes sur les paramètres n'a pas été effectuée.

En outre, il est pour l'instant peu facile de revenir en arrière en cas de problème dans la propagation. L'utilisateur est obligé de revenir à la fenêtre principale. Il n'y a pas non plus de mise en mémoire des dernières opérations effectuées par exemple.

Solveur et Optimisation

L'enregistrement de l'orientation du Graphe permet d'effectuer les calculs rapidement, dans un ordre logique, suivant les paramètres d'entrée fixés.

La fonction Solver est implantée. Elle ne permet cependant pas de résoudre les systèmes implicites.

En ce qui concerne l'optimisation, nous avons implanté le calcul des dérivées des paramètres de sortie par rapport aux paramètres d'entrée, ce qui nous permet de valider l'approche choisie.

Il faut maintenant implanter un algorithme d'optimisation ainsi qu'un mécanisme de normalisation, et la possibilité de prendre en compte des contraintes sur les paramètres.

Conclusions du chapitre

Dans ce chapitre, nous avons décrit l'implantation effective d'un prototype, issu des spécifications de cahier des charges posé, et de l'architecture présentée au chapitre précédent. Nous avons décrit en détail les choix techniques qui ont présidés à la réalisation de ce prototype, en présentant les difficultés rencontrées. Enfin nous avons présenté une session de fonctionnement de ce prototype, en présentant finalement, en quoi il satisfait au cahier des charges. Les principales fonctions pour la création d'un modèle et son calcul, ainsi que le calcul des dérivées ont été implantées. Cependant, il reste encore des points du cahier des charges non remplis, qui demandent plus ou moins de travail d'implantation en perspective. Ainsi, le dimensionnement sous-contrainte effectif d'un dispositif électromagnétique, à l'aide de son modèle analytique, qui était un des buts principaux de l'outil n'est pas faisable pour l'instant. Cette présentation nous amène à la conclusion de cette étude, et à la description de ses perspectives.

Conclusions et Perspectives

Conclusions

Dans cette étude, portant sur la conception sous contraintes en génie électrique, nous nous sommes intéressés, après un état de l'art sur le domaine concerné, plus particulièrement au problème de dimensionnement sous contraintes de dispositifs électromagnétiques. Le dimensionnement de structures connues est en effet un des aspects les plus important du travail du concepteur. Dans cette optique, nous nous sommes intéressés à l'utilisation des modèles analytiques, très utilisés en phase de préconception, et très répandus en génie électrique.

Notre problématique s'est alors définie comme d'offrir au concepteur un outil et une méthodologie pour effectuer le dimensionnement sous contraintes à l'aide de modèles analytiques. Après un tour d'horizon des outils existants, nous avons conclus à la nécessité d'un approche unificatrice, permettant au concepteur, d'effectuer la propagation de contraintes sur le modèle, de l'évaluer (solver) et de l'optimiser sous contraintes, le but étant d'utiliser au mieux toutes les informations contenues dans les équations analytiques du modèle.

Pour spécifier cette nouvelle approche, nous nous sommes au préalable penchés sur un des outils existants, Pascosma, dédié au dimensionnement sous contraintes. Cette étude nous a permis de dégager un certains nombre de limites de cet outil, et les problèmes inhérents à la manipulation des modèles analytiques. Un de ces problèmes est la prise en charge des systèmes implicites. Suite à cette étude, nous avons développé une méthode pour le résoudre.

Enfin, nous avons pu poser le cahier des charges pour la spécification de notre approche. Nous en avons déduit une architecture générale, structurée en objets. La dernière partie de ce travail a été l'implantation effective d'un prototype informatique, avec la justification des choix techniques effectués.

Ce prototype, à l'état actuel, permet la création d'un modèle d'équations analytiques, son orientation et son calcul, ainsi que le calcul des dérivées des paramètres de sortie. La prise en charge des paramètres implicites n'y est pas effectuée, la prise en compte d'un cahier des charges sur le modèle non plus. L'optimisation d'un modèle n'est pas encore possible, mais le mécanisme de calcul des dérivées valide la possibilité d'utiliser des méthodes d'optimisation de type gradients, très efficaces avec les modèles analytiques.

Perspectives

Les perspectives à court terme de notre travail concernent la manipulation complète d'un modèle analytique :

- souplesse dans les mécanismes d'ajout /suppression d'objets et de modification du modèle,
- prise en charge des systèmes d'équations implicites, avec l'implantation d'un algorithme de résolution des systèmes non-linéaires,
- prise en compte de contraintes sur le modèle, et définition d'une fonction objectif,
- mécanismes d'historique des opérations effectuées et de retour en arrière pour la propagation de contraintes sur le modèle,
- implantation d'un algorithme d'optimisation, pour le dimensionnement sous contraintes. Des travaux sont effectués dans ce sens, dans le cadre d'un DEA, pour l'intégration dans Java d'algorithmes existants écrits en FORTRAN, à l'aide des mécanismes du JNI (Java Native Interface)

A plus long terme, nous souhaitons ouvrir l'environnement à d'autres sources d'information que les équations analytiques, utiles pour le concepteur, comme des objets numériques, des fonctions affines par morceaux [SAUV 99], des minimisations par les moindres carrés [SAUV 99b], des systèmes d'équations, etc. Ceci implique la création de classes héritées d'Objet de Calcul, donc implantant les méthodes définies par cette classe générique, mais différentes de la classe Equation. Ces classes doivent être créées sous forme de fichiers Java. Il faut donc envisager des mécanismes d'aide à la création de tels fichiers, ainsi que la possibilité éventuelle de communiquer avec des objets étrangers à l'environnement de travail, comme des objets distribués en utilisant par exemple des techniques CORBA ou RMI.

Nous souhaitons également offrir un environnement souple et modulaire, avec la possibilité de modifier les algorithmes de parcours de graphe (utilisation de threads par exemple), d'optimisation (intégration d'objets dont les dérivées partielles ne sont pas calculables), de résolution des systèmes non-linéaires.

Les perspectives annexes sont les problèmes d'interface et de sauvegarde. L'interface graphique existante est assez rudimentaire et doit être améliorée. En ce qui concerne la sauvegarde, le but est de pouvoir restituer un graphe déjà utilisé sans avoir à le recréer à chaque fois, et de récupérer l'état des différents éléments du graphe. Nous envisageons pour cela d'exploiter le mécanisme de "sérialisation" de Java.

La création d'une bibliothèque de modèles existants serait un plus.

Annexe A. Théorème des fonctions implicites

Théorème des fonctions implicites dans le cas général (p entrées, q paramètres implicites, q équations)

Soient E, F et G trois espaces vectoriels normés de dimensions respectives p,q et q

soit O un ouvert de ExF

soit f de classe C1 de O dans G

$$E \times F \to G$$
$$(x_1, \cdots, x_p, x_{p+1}, \cdots, x_{p+q}) \mapsto (f_1, \cdots, f_q)(x_1, \cdots, x_{p+q})$$

soit c=(a,b) appartenant à O,

On suppose que la différentielle partielle de x en (a,b) est un isomorphisme de F dans G

ceci se traduit en pratique par la non nullité du déterminant des q dernières colonnes du Jacobien $J_f(a,b)$:

$$\begin{vmatrix} \frac{\partial f_1}{\partial x_{p+1}} & \cdots & \frac{\partial f_1}{\partial x_{p+q}} \\ \vdots & \ddots & \vdots \\ \frac{\partial f_q}{\partial x_{p+1}} & \cdots & \frac{\partial f_q}{\partial x_{p+q}} \end{vmatrix} (a,b) \neq 0$$

et on suppose de plus $f(a,b)=0$

Alors, il existe U, un voisinage ouvert de a dans E et V, un voisinage ouvert de b dans F

il existe φ de classe C1 de U dans V telle que

a- UxV est inclus dans O

b- $\forall (x,y) \in U \times V, f(x,y) = 0 \Leftrightarrow y = \varphi(x)$

c- $\forall x \in U, d\varphi_x \in \mathsf{L}(E,F)$ et $d\varphi_x(x) = df_y^{-1}(x,\varphi(x)) \circ df_x(x,\varphi(x))$

(avec $df_x(x,\varphi(x)) \in \mathsf{L}(E,G)$)

En clair, un voisinage de (a,b) dans l'ouvert UxV $\{(x,y) \in U \times V | f(x,y) = O_G\}$ est exactement le graphe d'une application de classe C1 de U dans V. (Et donc quel que soit x appartenant à U, il existe un unique y appartenant à V tel que $f(x,y)=O_E$ et y est fonction de classe C1 de x)

c- nous assure que φ est C1 et permet d'exprimer les dérivées partielles de φ en utilisant :

$$\forall i \text{ tq } 1 \leq i \leq q, \quad f_i(x_1, \cdots, x_p, \varphi_1(x_1, \cdots, x_p), \cdots, \varphi_q(x_1, \cdots, x_p)) = 0$$

donc :

$$\frac{\partial f_i}{\partial x_1}(x,\varphi(x)) + \sum_{j=1}^{q} \frac{\partial f_i}{\partial x_{p+j}}(x,\varphi(x)).\frac{\partial \varphi_j}{\partial x_1}(x) = 0 \text{ dans } U$$

On obtient q équations à q inconnues. C'est un système dont le déterminant est non nul, on peut donc l'inverser et trouver $\frac{\partial \varphi_i}{\partial x_j}$.

Dans [1], on retrouve explicité ce théorème.

Il exprime le jacobien J de φ par : $J=-Q^{-1}.P$

avec :

$$P = \left[\frac{\partial f_i}{\partial x_j}(x_1,\cdots,x_p,\varphi_1(x_1,\cdots,x_p),\cdots,\varphi_q(x_1,\cdots,x_p))\right],(i,j)\in \mathbf{N}_q \times \mathbf{N}_p$$

$$Q = \left[\frac{\partial f_i}{\partial y_k}(x_1,\cdots,x_p,\varphi_1(x_1,\cdots,x_p),\cdots,\varphi_q(x_1,\cdots,x_p))\right],(i,k)\in \mathbf{N}_q \times \mathbf{N}_q$$

Ce qui correspond à la formule littérale donnée plus haut.

Ces équations nous donnent un système linéaire de qxp équations à qxp inconnues (les $\frac{\partial \varphi_i}{\partial x_j}$, $1 \leq i \leq q$, $1 \leq j \leq p$) qui est résoluble par des algorithmes tels que le pivot de Gauss par exemple. On peut donc calculer de façon « juste » les dérivées partielles, au sens que l'on ne fait pas d'approximation, aux erreurs numériques de calcul près.

On aura donc besoin aussi d'une routine de résolution de systèmes linéaires à n inconnues.

Références

[1] E. Ramis, **Cours de Mathématiques Spéciales**, 3rd ed., vol. 3. Ed.Masson, Paris, 1991

Annexe B. Module développé dans Macsyma, pour la prise en charge des paramètres implicites

Module développé dans Macsyma, pour l'implantation de la deuxième méthode de prise en charge des paramètres implicites dans un modèle analytique

Cahier des charges

On suppose que les équations entrées par l'utilisateur sont de la forme :

$$\begin{cases} f_1(x_1,\cdots,x_n,e_1,\cdots,e_q)=0 \\ \vdots \\ f_n(x_1,\cdots,x_n,e_1,\cdots,e_q)=0 \end{cases}$$

où n est l e nombre de paramètr es implicites $(x_1,\cdots,x_n)$,
donc aussi le nombre d' équations $(f_1,\cdots,f_n)$,
et q le nombre de paramètres d ' entrée $(e_1,\cdots,e_q)$

L'outil fournira les expressions des paramètres implicites, fonctions des paramètres d'entrée, des valeurs initiales des paramètres implicites et éventuellement des paramètres de réglage des algorithmes utilisés pour la résolution, qui seront directement utilisables dans les fichiers de déclaration de PASCOSMA, ainsi que la déclaration des dérivées partielles de ces fonctions et évidemment les fichiers fortran correspondants (Fig. B-1).

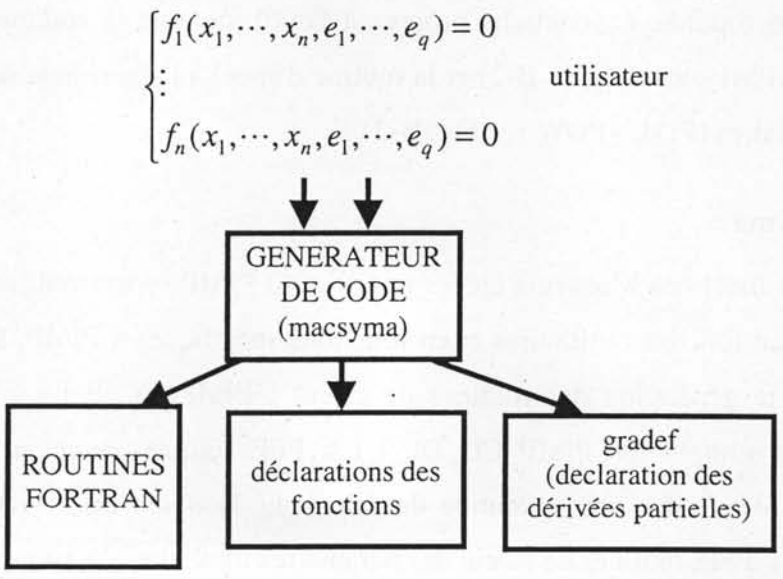


Fig. B-1 : Fonctions à réaliser par l'outil de prise en charge des paramètres implicites

Calcul des paramètres implicites et de leurs dérivées partielles

Le calcul des paramètres implicites s'effectuera grâce à une routine Harwell de résolution de systèmes non linéaires à n inconnues (routine NS02AD) [1]. Cette routine utilise notamment le calcul des équations implicites et de leurs dérivées partielles. Les fonctions $(f_1,\cdots,f_n)$ entrées par l'utilisateur et leurs dérivées devront donc être codées. Elle nécessite

aussi la résolution d'un système linéaire d'équations, pour le calcul de certains vecteur d'entrée de la routine.

En appliquant le théorème des fonctions implicites [2], on peut calculer les dérivées partielles des paramètres implicites de façon juste, ce qui est d'un grand intérêt pour l'utilisation dans PASCOSMA.

Structure de l'outil

L'outil se décompose en deux sous parties qui correspondent aux différentes étapes de son utilisation. La première partie correspond au code source Macsyma où sont définies les fonctions propres à cet outils qui servent à la génération de code en Fortran. La deuxième partie correspond aux routines Fortran « fixes » qui sont soit la routine de résolution de systèmes non linéaires issue de la bibliothèque Harwell, soit des routines propres à l'outil et qui seront à mettre dans le répertoire de génération de l'application PASCOSMA directement.

Routines Fortran fixes

Les routines Fortran fixes sont divisées en routines et fonctions issues de la bibliothèque HARWELL et en routines et fonctions propres à l'outil, comme la routine d'inversion de matrice (PIMP_APMI sur la Figure B-2) et la routine d'appel à l'algorithme de résolution des systèmes non-linéaires (PIMP_POW sur Fig. B-2).

Fonctions Macsyma

Toutes les fonctions Macsyma créées pour l'outil PIMP se trouvent un même fichier. Elles se divisent en fonctions utilitaires et en fonctions spécifiques à PIMP. Le passage dans Macsyma sert à la génération des routines de calcul : PIMP_CC_F_i_j_PFE, routines de calcul des fonctions implicites, PIMP_CC_DF_i_j_k_PFE, routines de calcul des dérivées de ces fonctions, PIMP_CALL_PQ_i, routine de calcul du Jacobien des fonctions implicites, PIMP_CC_F1_i_j_PFE, routines de calcul des paramètres implicites du système implicite, qui utilisent l'algorithme de résolution des systèmes non-linéaires, et PIMP_CC_DF1_i_j_k_PFE, routines de calcul de leurs dérivées par rapport aux paramètres d'entrée du système implicite (Fig. B-2).

La fonction principale Macsyma à utiliser pour la génération concenant un système implicite est la suivante. C'est la seule fonction que manipule directement l'utilisateur.

PIMP_CC_GENERE (NUMI,STR_PIMP,LISTE_EQ_I,LISTE_PIMP_I,LISTE_PIMPINI_I,PARAM_NS02)

Interface d'entrée

numi: numéro du système,
str_pimp : chaîne de caractères du nom attaché à PIMP : c'est soit « PIMP », soit « MPIMP » selon que l'outil PIMP est utilisé seul ou avec l'outil MPMC
liste_eq_i : liste des équations du système
liste_pimp_i : liste des paramètres implicites du système
liste_pimpini_i : liste des paramètres implicites initiaux
param_ns02 : liste des paramètres de la routine ns02ad

Interface de sortie

---

true si tout se passe bien, false sinon avec un message d'erreur et arrêt.
Il y a aussi des warnings pour certains problèmes rencontrés qui peuvent être passés outre par l'utilisateur.

Le fonctionnement complet de PASCOSMA avec la prise en charge des paramètres implicites est montré sur la Figure B-2.

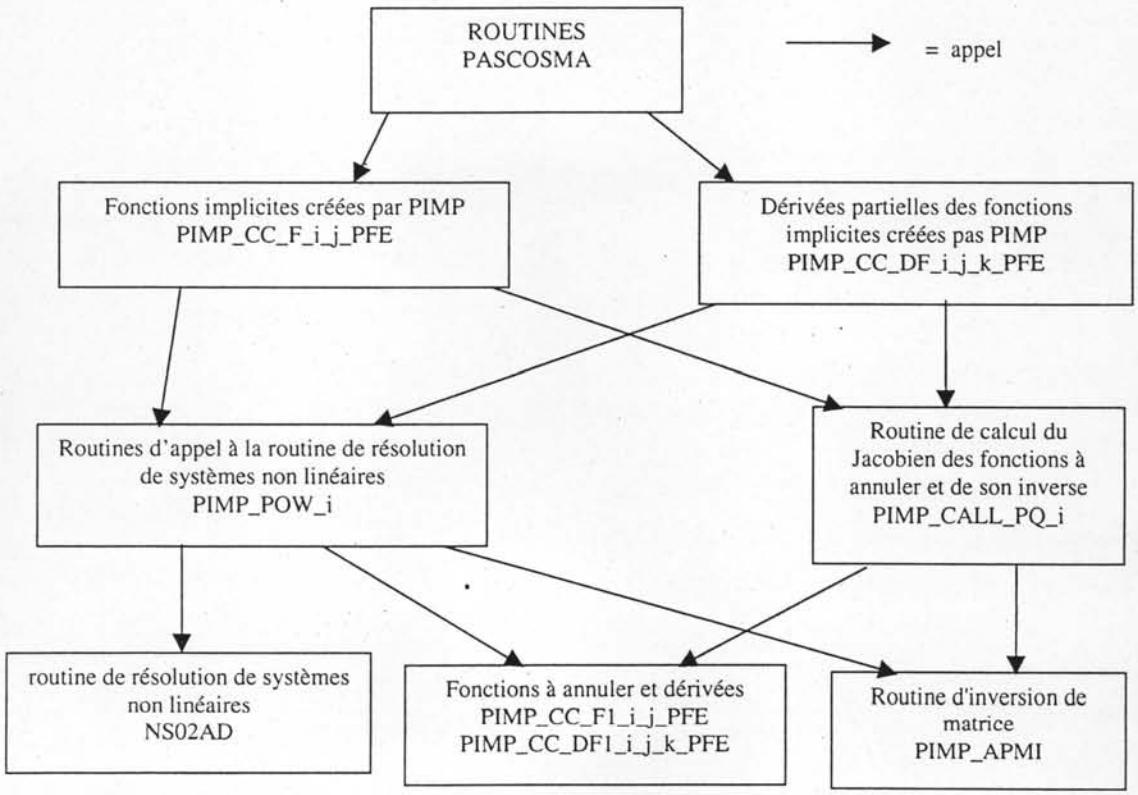


Fig. B-2 : Fonctionnement de l'outil de prise en charge des paramètres implicites

Références

[1] E. Ramis, **Cours de Mathématiques Spéciales**, 3rd ed., vol. 3. Ed.Masson, Paris, 1991

[2] Harwell Subroutine Library, release 1984, routines VF13AD, NS02AD, disponibles chez R.S.L., Z.I.R.S.T., Chemin du Pré Carré, 38240 Meylan

Annexe C. Description des classes principales de l'architecture développée

Description des classes principales de l'architecture développée

Sur la figure de l'architecture implantée donnée au Chapitre 5 § 1.4, et représentée Fig. A-1, ont été représentées les différentes classes développées, regroupées dans leur package. Nous avons créé en effet 5 packages :

- *design* (Graphe sur la figure A-1), qui contient la classe Graphe et les classes des objets contenus dans le Graphe,
- *algorithm* (Algorithmique sur la figure A-1), qui est destiné à contenir les algorithmes d'optimisation et de résolution de systèmes non-linéaires, et qui ne contient que l'algorithme de parcours de Graphe pour l'instant,
- *service* (Services sur la figure A-1), qui contient tous les services utiles, comme la classe Matrice d'Occurrence, les Parser, la classe des Options d'environnement.
- *exception* (Gestion des erreurs sur la figure A-1), qui contient la hiérarchie des erreurs possibles sur l'application (Erreurs de calcul, nom inconnu, valeur erronée, erreur dans le Graphe, etc)
- *gui* (Interface Graphique sur la figure A-1), qui contient les fenêtres de l'application et a gestion des événements correspondants.

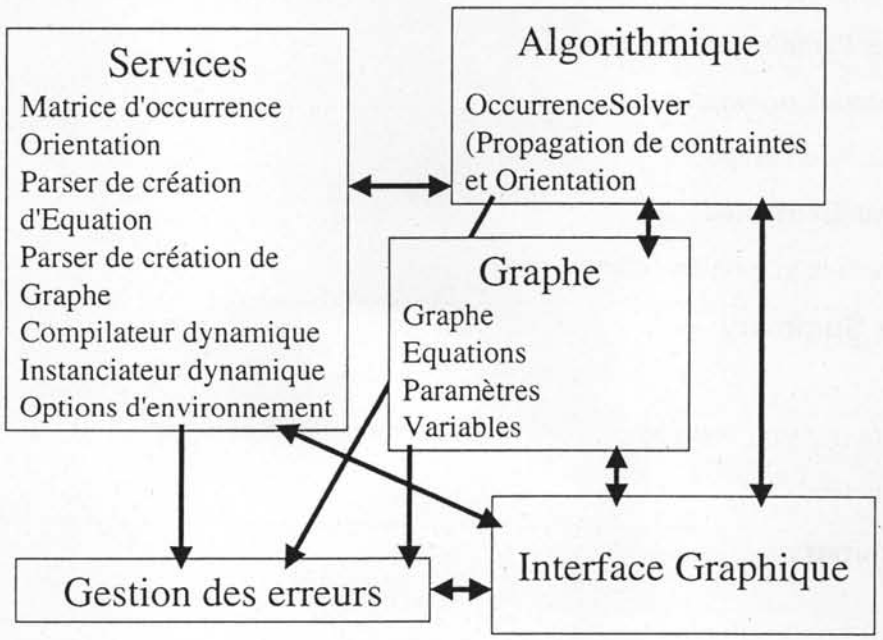


Fig. A-1 : Architecture du prototype implanté

Nous allons donner ici le détail des classes du package *design*, qui sont le cœur de notre approche, puisqu'elles contiennent l'information sur le Graphe.

package design

Class Graph

java.lang.Object

|

+--**design.Graph**

---

public class **Graph**

extends java.lang.Object

classe du graphe comprenant les méthodes de services sur le graphe utilisées par l'interface et les algorithmes de solveur, d'optimisation, etc

---

Field Summary

private java.lang.String ***name***

nom du graphe

private java.util.Vector ***designObjectList***

liste des objets de calcul dans le graphe

private java.util.Vector ***parameterList***

liste des Parametres dans le graphe

private Orientation ***orientation***

orientation du graphe

private boolean ***isOriented***

indique si le graphe est orienté

Constructor Summary

Graph (*java.lang.String entree*)

constructeur

Method Summary

void **addDesignObject** (*DesignObject* obj)

l'objet de calcul donné en paramètre est rajouté dans le vecteur designObjectList

void **addInputParameter** (*Parameter* param)

ajoute un paramètre d'entrée au Graphe

void **addParameter** (*Parameter* param)

ajoute le Paramètre donné en entrée au vecteur parameterList

boolean **containsDesignObject** (*java.lang.String* nom)

indique si un designObject portant le nom nom existe déjà

boolean **containsParameter** (*java.lang.String* nom)

indique si un Parameter portant le nom nom existe déjà

int **degreesOfFreedom** ()

renvoie le nombre de degrés de liberté

DesignObject **getDesignObject** (*java.lang.String* nom)

retourne l'objet de calcul possédant ce nom

java.util.Vector **getDesignObjectList** ()

renvoie le vecteur designObjectList

java.lang.String **getName** ()

retourne le nom du graphe

Orientation **getOrientation** ()

retourne l'orientation du graphe si elle existe (objet vide sinon)

Parameter **getParameter** (*java.lang.String* nom)

retourne le paramètre possédant ce nom

java.util.Vector **getParameterList** ()

renvoie le vecteur parameterList

java.util.Vector **getSortParameterList** ()

renvoie le vecteur des paramètres dans l'ordre alphabétique

boolean **isOriented** ()

renvoie un booléen indiquant si l'orientation est valable ou non

OccurenceMatrix **occurenceMatrix** ()

renvoie la matrice d'occurrence du graphe - les lignes sont les objets de calcul, les colonnes les variables

boolean **removeDesignObject** (*DesignObject* obj)

l'objet de calcul est supprimé du vecteur designObjectList

boolean **removeParameter** (*Parameter* param)

supprime le Paramètre donne en entrée du vecteur parameterList

void **resetOrientation** ()

remet l'orientation a zéro

void **setOrientation** (*Orientation* orient)

entre une orientation de graphe valide

package design

Class DesignObject

java.lang.Object

|

+--**design.DesignObject**

Direct Known Subclasses:

Equation

---

public abstract class **DesignObject**

extends java.lang.Object

classe mère abstraite décrivant l'objet de calcul générique

Field Summary

protected java.lang.String ***name***

nom de l'objet de calcul

protected int ***numberOfOutput***

nombre de variables de sortie de l'objet de calcul

protected java.util.HashSet ***variableList***

liste des variables de l'objet de calcul

Constructor Summary

DesignObject ()

constructeur sans argument

DesignObject (*java.lang.String* nom)

constructeur qui affecte le nom

Method Summary

abstract void **compute**()

méthode de calcul de l'objet de calcul abstraite

void **compute** (*java.util.HashSet* list)

calcule la liste des variables de sortie donnée en paramètre

abstract void **compute** (*java.util.HashSet* ouputList, *java.util.HashSet* inputList)

calcule la liste des variables de sortie en fonction des variables d'entrée. c'est une classe abstraite

abstract void **computeDerivative** ()

méthode de calcul des dérivées abstraite

void **computeDerivative** (*java.util.HashSet* list)

calcule les dérivées des variables de list en fonction des autres variables (=ensemble des variables - celles de list) par rapport aux autres variables

abstract void **computeDerivative** (*java.util.HashSet* outputList, *java.util.HashSet* deriveList, *java.util.HashSet* inputList)

calcule les dérivées des output par rapport aux derive en fonction des input.

abstract boolean **consistencyTest** ()

méthode de test de cohérence abstraite

java.lang.String **getName**()

renvoie le nom de l'objet de calcul

int **getNumberOfOutput** ()

renvoie le nombre de sorties de l'objet de calcul

java.util.HashSet **getParameterList** ()

renvoie la liste des paramètres liés aux variables de l'équation. Le HashSet est vide s'il n'y en a pas

Variable **getVariable** (*java.lang.String* nom)

renvoie la variable correspondant au nom donne en entrée

Variable **getVariableBoundToParameter** (*Parameter* param)

méthode qui retourne la variable liée au paramètre donné en entrée si elle existe, sinon la méthode revoie une exception

java.util.HashSet **getVariableList** ()

renvoie la liste des variables

boolean **isComputable** (*java.util.HashSet* list)

dit si la liste des variables donnée en entrée est calculable .

abstract boolean **isComputable** (*java.util.HashSet* outputList, *java.util.HashSet* inputList)

dit si la liste des sorties est calculable en fonction de la liste des entrées.

boolean **isDerivable** (*java.util.HashSet* outputList)

dit si la liste des variables donnée en entrée est dérivable par rapport aux variables d'entrée (qui sont considérés comme étant les variables de l'objet de calcul non demandées en sortie).

abstract boolean **isDerivable** (*java.util.HashSet* outputList, *java.util.HashSet* deriveList, *java.util.HashSet* inputList)

dit si la liste des sorties est dérivable en fonction de la liste des entrées par rapport aux variables de la deuxième liste.

boolean **isVariable** (*java.lang.String* nom)

renvoie un booléen disant si une varaible correspondant au nom donne en entrée existe dans l'objet

void **setName** (*java.lang.String* nom)

modifie le nom de l'objet de calcul

protected **void testDeriveList** (*java.util.HashSet* list)

teste si la liste de variables par rapport auxquels on dérive donnée en entrée est bonne : tous ses éléments appartiennent a l'objet de calcul, elle est de la bonne taille. Renvoie une exception dans le cas contraire

protected void **testInputList** (*java.util.HashSet* list)

teste si la liste d'entrées donnée en entrée est bonne : tous les éléments appartiennent a l'objet de calcul, elle est de la bonne taille. Renvoie une exception dans le cas contraire.

protected void **testOutputList** (*java.util.HashSet* list)

teste si la liste de sorties donnée en entrée est bonne : tous ses éléments appartiennent à l'objet de calcul, elle est de la bonne taille. Renvoie une exception dans le cas contraire

---

package design

Class Equation

java.lang.Object

|

+--design.DesignObject

|

+--**design.Equation**

---

public abstract class **Equation**

extends DesignObject

classe abstraite, patron des objets de calcul équation. C'est une sous classe de la classe mère DesignObjet décrivant l'objet de calcul générique

Field Summary

protected java.lang.String ***expression***

contient l'expression implicite de l'équation sous forme mathématique (u-r*i pour u=r*i)

protected double ***expressionValue***

valeur de l'expression - permet de tester la cohérence d'une équation

Fields inherited from class design.DesignObject :

name, numberOfOutput, variableList

Constructor Summary

Equation ()

constructeur sans argument

Equation (java.lang.String nom)

constructeur qui affecte le nom (name)

Method Summary

void **compute** ()

méthode qui calcule, en fonction des variables qui ont été fixées, la variable de sortie

void **compute** (*java.util.HashSet* outputList, *java.util.HashSet* inputList)

calcule la liste des variables de sortie donnée en entrée, en fonction de la liste des variables d'entrée.

abstract void **compute** (*Variable* var)

methode abstraite qui calcule la variable

void **computeDerivative** ()

méthode qui calcule les dérivées, en fonction des variables qui ont été fixées

void **computeDerivative** (*java.util.HashSet* outputList, *java.util.HashSet* deriveList, *java.util.HashSet* inputList)

dérive la première liste des variables donnée en entrée par rapport a la seconde en fonction de la troisième.

abstract void **computeDerivative** (*Variable* aderiver, *Variable* derparrap)

méthode abstraite qui calcule la dérivée de aderiver par rapport à derparrap

abstract double **computeExpression** ()

méthode qui calcule l'expression de l'équation

boolean **consistencyTest** ()

méthode de test de cohérence

java.lang.String **getExpression** ()

renvoie l'expression de l'expression implicite de l'équation

boolean **isComputable** (*java.util.HashSet* outputList, *java.util.HashSet* inputList)

dit si la liste des variables de sortie donnée en entrée est calculable en fonction de la liste des entrées. implantation de la méthode abstraite issue de DesignObject.

abstract boolean **isComputable** (*Variable* var)

méthode abstraite qui dit si la variable donnée en entrée est calculable

boolean **isDerivable** (*java.util.HashSet* outputList, *java.util.HashSet* deriveList, *java.util.HashSet* inputList)

dit si la liste des variables de sortie donnée en entrée est dérivable en fonction de la liste des entrées. implantation de la méthode abstraite issue de DesignObject.

abstract boolean **isDerivable** (*Variable* var)

méthode abstraite qui dit si la variable donnée en entrée est derivable

protected void **setDerivativeValue** (*Variable* var, *Variable* der, *double* valeur)

méthode pour affecter la valeur de sa dérivée à une variable (appelle une méthode protected de Variable)

protected void **setValueOut** (*Variable* var, *double* valeur)

méthode pour affecter sa valeur à une variable (appelle une méthode protected de Variable)

---

package design

Class Parameter

java.lang.Object

|

+--**design.Parameter**

---

public class **Parameter**

extends java.lang.Object

classe qui décrit les paramètres du Graphe ; liés aux Objets de calcul

Field Summary

private java.lang.String ***name***

nom du paramètre

private java.util.HashSet ***variableList***

liste des variables auxquelles est relié le paramètre

private double ***value***

valeur du paramètre

private java.util.Hashtable ***derivative***

valeurs des dérivées si le graphe est orienté

private int ***status***

statut : entrée, sortie, implicite ou inconnu

static int ***INPUT***

attribut statique du statut entrée

static int ***OUTPUT***

attribut statique du statut sortie

static int ***IMPLICIT***

attribut statique du statut implicite

static int ***UNKNOWN***

attribut statique du statut inconnu

private static int ***MAX***

attribut statique final prive qui donne le maximum de la valeur de statut

private static int ***MIN***

attribut statique prive qui donne le minimum de la valeur de statut

Constructor Summary

Parameter (*java.lang.String* nom)

constructeur affectant nom. le statut par défaut est UNKNOWN

Parameter (*java.lang.String* nom, *int* etat)

constructeur a deux paramètres : nom et statut

Method Summary

void **addDerivative** (*Parameter* par, *java.lang.Double* val)

méthode qui propage les valeurs des dérivées trouvées dans la variable liée au paramètre

void **bindVariable** (*Variable* var)

méthode pour lier un paramètre a une variable

void **findValue** (*double* valeur, *Variable* var)

méthode qui propage une valeur trouvée dans la variable liée au paramètre

java.util.Hashtable **getDerivative** ()

renvoie le tableau des dérivées par rapport aux paramètres d'entrée du Graphe

java.lang.String **getName**()

donne le nom du Parametre

java.lang.String **getStatus**()

méthode qui renvoie le statut sous forme de chaîne de caractères (String)

double **getValue** ()

méthode qui renvoie la valeur du paramètre

java.util.HashSet **getVariableList** ()

retourne la liste des variables auxquelles est lié le paramètre

boolean **isImplicit** ()

méthode pour savoir si le paramètre a le statut Implicit

boolean **isInput** ()

méthode pour savoir si le paramètre a le statut Input

boolean **isOutput** ()

méthode pour savoir si le paramètre a le statut Output

void **reInit**()

méthode pour réinitialiser le paramètre (sans toucher aux variables)

void **setStatus** (*int* etat)

méthode pour affecter le statut

void **setValue** (*double* valeur)

méthode pour fixer la valeur du paramètre

private boolean **testStatus** (*int* val)

méthode privée de test sur le statut

void **unbindVariable** (*Variable* var)

méthode pour délier un paramètre d'une variable

package design

Class Variable

java.lang.Object

|

+--design.Variable

---

public class **Variable**

extends java.lang.Object

classe générique des variables liées aux Objets de calcul, permet de stocker les valeurs locales sur l'Objet de calcul. Une Variable n'existe pas sans Objet de calcul

---

Field Summary

private java.lang.String ***name***

nom de la variable

private DesignObject ***desobj***

objet de calcul auquel est liée la variable (il n'y en a qu'un)

private Parameter ***param***

Paramètre auquel est liée la variable (il n'y en a qu'un)

private boolean ***isInput***

indique si c'est une entrée de l'Objet de Calcul

private double ***valueIn***

valeur d'entrée (fixée par l'utilisateur). valueIn est double pour l'instant.

private double ***valueOut***

valeur de sortie (calculée). valueOut est double pour l'instant.

private java.util.Hashtable ***derivative***

tableau de valeurs des dérivées partielles sous la forme d'une Hashtable (table de hachage). la clé est le paramètre par rapport auquel on dérive, l'objet la valeur de la dérivée (attention a la gestion de la cohérence par rapport aux entrées choisies)

Constructor Summary

Variable (*java.lang.String* nom, *DesignObject* obj)

constructeur : prend en paramètre le nom et l'Objet de calcul auquel est liée la variable.

Method Summary

void **bindParameter** (*Parameter* paramet)

méthode qui permet de lier la variable a un paramètre

java.util.Hashtable **getDerivative** ()

méthode qui renvoie toutes les dérivées calculées

double **getDerivativeValue** (*Variable* param)

méthode qui renvoie la valeur de la dérivée de la Variable par rapport à la variable donnée en entrée

java.lang.String **getName** ()

méthode qui renvoie le nom de la Variable

Parameter **getParam** ()

méthode qui renvoie le paramètre auquel est liée la variable

double **getValueIn** ()

méthode qui renvoie la valeur d'entrée de la Variable

double **getValueOut** ()

méthode qui renvoie la valeur de sortie (utile seulement si il n'y a pas de paramètre lié)

boolean **isInput** ()

méthode qui dit si la variables est une entrée pour l'Objet de calcul auquel elle est liée

void **reInit** ()

méthode qui réinitialise lorsque le paramètre lié a été réinitialisé

void **setDerivativeValue** (*Variable* var, *java.lang.Double* valeur)

méthode qui fixe la valeur d'une dérivée

void **setValueIn** (*double* valeur)

méthode qui fixe la valeur d'entrée

void **setValueOut** (*double* valeur)

méthode qui fixe la valeur de sortie

void **unbindParameter** ()

méthode qui permet de délier la variable d'un paramètre

Bibliographie

Bibliographie

[AKOU 84] "3D analytical calculation of the forces exerted between two cuboidal magnets", G. Akoun & J.-P. Yonnet, IEEE Transactions on Magnetics, vol. MAG-20, n°5, Sept 1984, pp1962-1964

[ALGE 70] "Induction Machines", P. L. Alger, ed. Gordon and Breach Science Publishers, 150 Fifth Avenue, New-York, USA, 518 p, 1970

[ALOT 96] "A multiquadrics based algorithm for the acceleration of simulated annealing optimisation procedure", P. Alotto, A. Cuiti & al., IEEE Transaction on Magnetics, vol. 32, n°3, May 1996, pp 1198-1201

[BIAN 95] "Optimal design technics applied to transverse flux induction heating systems", N. Bianchi, F. Dughiero, IEEE Transactions on Magnetics, vol. 31, n°3, May 1995, pp 1992-1995

[BARA 83] "Analytical methods for the design calculation of permanent-magnet synchronous couplings", W. Baran, Topics in Technical Physics, Acta Polytechnica Scandinavica Applied Physics, Series n°138, Helsinki, 1983

[BELOT] Calcul des machines électriques, cours de l'ESE

[BIGE 94] "Processus de conception de produit et dynamique des relations dans l'entreprise et entre entreprises (une approche multi-acteurs)", J. Bigeon et J.-Ch. Monateri, Rapport du projet de recherches IPI Juin 1994, Diffusion interne.

[BOUL 90] "Design of Permanent Magnet DC Motors", IEEE Transactions on Industry, vol. 26, n°4, July/august 1990, pp 786-792

[BRISS 95] "Outil et méthodologie pour la conception des moteurs à réluctance variable à double saillance", S. Brissset, Thèse de Doctorat en Génie Electrique de l'USTL, Lille, 1995

[COCH 89] "Polyphase Induction Motors – Analysis, Design and Application", P. L. Cochran, ed. Marcel Dekker, 675 p., 1989, ISBN 0-8247-4

[COUT 99] "Design of permanent magnet coupling using the PASCOSMA methodology", C. Coutel, P. Pandele, J-P. Yonnet, F. Wurtz & J. Bigeon, Proc. of IEEE IEMDC'99, May 1999, Seattle, USA.

[COUT 99b] "Constrained optimisation of a linear actuator : comparison of two methods to deal with implicit parameters in the analytical model", C. Coutel, F. Wurtz, J. Bigeon & C. Chillet, Proc. of IEEE IEMDC'99, May 1999, Seattle, USA.

[COUT 99c] "A comparative study of two methods for constrained optimisation with analytical models dealing with implicit parameters", C. Coutel, F. Wurtz & J. Bigeon, IEEE Trans. on Magn., vol. 35, n°3, May 1999, pp 1738-1741

[DEVA 94] "Optimisation de forme des structures électromagnétiques", J. A. De Vasconcelos, Thèse de Doctorat, Ecole Doctorale de Lyon des Sciences pour l'Ingénieur : Electronique, Electrotechnique, Automatique, 1994

[ESCA 96] "Modélisation objet du processus de conception dans le domaine du génie électrique: application au cas de la machine asynchrone", E. Escande, Thèse de Doctorat, Institut National Polytechnique de Grenoble (INPG), 1996

[ESPA 99] "Modélisation et conception optimale de moteurs sans balais à structure inversée. Application au moteur roue", C. Espanet, Thèse de Doctorat, Université de Franche-Comté, 1999

[FELL 79] "Permanent magnet couplings", C. J. Fellows, CME, June 79, pp 79-84

[FIAC 68] "Non-linear progrmming : sequential unconstrainde minimization techniques", A. V. Fiacco, G. P. McCormick, Wiley, 1968

[FLUX2D] FLUX2D, version 7.30, Notice d'utilisation générale, Juin 1998, CEDRAT, 10 Chemin de Pré Carré, Zirst, 28246 Meylan Cedex

[FRAN 94] "Contribution de la modélisation floue à la conception en génie électrique", F. François, Thèse de Doctorat, INPG, 1994

[GEIB 98] "Corba : des concepts à la pratique", J.M. Geib, C. Grandsart & P. Melle, Informatiques – Inter-éditions, ISBN 2225-83046-0

[GERB 93] "Aide à la conception des ensembles machine-convertisseur-commande. Apport d'une démarche système expert", L. Gerbaud, Thèse de Doctorat, INPG, 1993

[GENT 91] "COCASE, un système d'aide à la conception d'appareillages électriques", A. Gentilhomme, Thèse de Doctorat, INPG, 1994

[GIEL 90] "Analog circuit design optimisation based on symbolic simulation and simulated annealing", G. Gielen, H. Walscharts & W. Sansen, Journal of Solid State Circuits, vol. 25, n°3, Juin 1990, pp 707-713

[GOSL 93] "Algébraic constraints", J. Gosling , Ph. D. Disserattion, Department of Computer Science, Carnegie-Mellon University, May 93

[HABI 98] "Connaître l'activité de conception", B.Bel Habib, rapport interne de l'équipe Conception et Diagnostics Intégrés, Laboratoire d'Electrotechnique de Grenoble (INPG/UJF – CNRS, UMTR 5529), janvier 98

[HARA 97] "Une Approche Multi-Modèles pour la capitalisation des connaissances dans le domaine de la conception.", Y. Harani, Thèse de Doctorat, INPG, 1997

[HARWE] Harwell Subroutine Library, release 1984, routines VF13AD, NS02AD, disponibles chez R.S.L., Z.I.R.S.T., Chemin du Pré Carré, 38240 Meylan.

[HORS 99] "Au coeur de Java 2. Volume I – Notions fondamentales", C. S. Horstmann & G. Cornell, CampusPress, France, ISBN 2-7440-0620-3 (Titre original : "Core Java 2.Volume I : Fundamentals", ISBN 0-13-081933-6)

[ILOG] AIDA, v. 1.76, SMECI, v. 1.76, Manuel de Référence, ILOG S.A., 2 av Galliéni, BP 85, 94253 Gentilly Cedex, 1994

[INFOR] "L'entreprise java : le point sur l'offensive en Europe", Informatiques, Hors serie n°2, Avril-mai 1999.

[KADD 93] "Optimisation de formes de machines électriques, à l'aide d'un logiciel éléments finis et de la méthode des pénalités intérieures étendues", K. Kadded, Thèse de Doctorat, INPG, 1993

[KONE 93] "Contribution à la conception des actionneurs électriques par formulation en termes d'optimisation", A. D. Kone, Thèse de Doctorat, Institut National Polytechnique de Toulouse , 1993

[KONE 93b] "Le dimensionnement des actionneurs électriques : un problème de programmation non linéaire", A. D. Kone, B. Nogarede, M. Lajoie-Mazenc, Journal de Physique III, Février 1993, pp 285-301

[KONO 84] "Expert Systems for personal computers : the TK!Solver approach", M. konopasek, S. Jayaraman, Byte, May 1984, pp 137-156

[LECH 98] "Analyse fonctionnelle des convertisseurs statiques en vue de la conception", C. Lechevalier, Thèse de Doctorat, INPG, 1998

[LUND] http://www.javathings.com/

[MACSY] "Macsyma User's Guide", Second Edition, Maccsyma Inc., 1996

[MAPLE] "The Maple Handbook – Maple V Release 4", D. Redfen, Waterloo Maple Inc., 450 Phillip street, Waterloo, On, Canada, ISBN 0-387-94622-5

[MOHA 94] "An intelligent system for design optimisation of electromagnetic devices", O. A. Mohammed, R. Merchant & F. G. Üler, IEEE Transactions on Magnetics, vol. 30, n°5, Sept 1994, pp 3633-3636

[PIET 98] "Optimisation og Interconnection Inductances with a coupled PEEC-Quasi Newton Method", N. Piette, Y. Maréchal & E. Clavel, Proc. of IEEE CEFC'98, June 1998, Tucson, USA, p 453

[POWE 69] "A method for nonlinear constraints in minimization problems", M. J. D. Powell, R. Fletcher (ed.), Optimization Academic Press, NY, 1969

[POWE 70] "Numerical methods for non-linear algebraic equations", M. J. D. Powell, ed. P. Rabinowitz, Gordon and Breach, 1970, pp 87-161

[RAMA 73] "A comparative study of minimization techniques for optimization of induction motor design", R. Ramarathnan, B. G. Desai & V. Subba Rao, IEEE Transactions on PAS, vol PAS-22, n°5, Sept/Oct 1973, pp1448-1454

[RIBE 94] "Genetic-Algorithm Programming Environments", J. L. Ribeiro Filho, P. C. Treleaven & C. Alippi, IEEE Computer, June 1994, pp 28-43

[SALD 87] "Device Modelling in an electromagnetic design system", C. M. Saldanha & D. A. Lowther, IEEE Transactions on Magnetics, vol. 23, n°5, Sept 1987, pp 2644-2646

[SALD 88] "An Algebraic Constraint System for Computer-Aided Design in Magnetics", C. M. Saldanha, Thesis for Master of Engineering, Computational Analysis and Design Laboratory, Department of Electrical Engineering, McGill University, Montreal, Canada, 1988

[SALD 88b] "Optimisation en Electromagnétisme par application conjointe des Méthodes de Programmation Non Linéaire et de la Méthode des Eléments Finis", R. R. Saldanha, Thèse de Doctorat, INPG, 1988

[SAUV 99] "A Methodology for using Morsels Continuous Linear Functions in Gradient Optimisation of Electrical Devices", C. Sauvey, F. Wurtz, J. Bigeon & F. Binet, Proc. of IEEE IEMDC'99, May 1999, Seattle, USA, pp 195-197

[SAUV 99b] "A Methodology for using Least Squares Minimisation in Gradient Optimisation of Electrical Devices", C. Sauvey, F. Wurtz, J. Bigeon & F. Binet, Proc. of IEEE IEMDC'99, May 1999, Seattle, USA.

[SHOOD 95] "Modélisation objet pour la CFAO : modèle de conception", M. Tollenaere, Laboratoire d'Electrotechnique de Grenoble, Laboratoire de l'INRIA Rhône-Alpes, Laboratoire Sols, Solides et Structures de Grenoble, Lméca d'Annecy, Rapport de fin de contrat, Août 1995

[SING 92] "Practical considerations in the optimisation of induction motor design", C. Singh & D. Sankar, IEE Proceeding-B, vol. 139, n°4, July 1992, pp 365-373

[SLEM 94] "On the design of Hign-Performance Surface-Mounted PM Motors", G. R. Slemon, IEEE Transactions on Industry Applications, vol. 30, n°1, January/February 1994, pp 305-310

[SRID 93] "Active occurrence-matrix based approach to design decomposition", N. Sridhar, R. Agrawal & G. L. Kinzel, Computer-Aided Design, Vol. 25, n°8, August 1993, pp 500-512

[SUN] http://java.sun.com/

[SWIN 91] "DONALD : a workbench for interactive design space exploration and sizing of analog circuits", K. Swings & W. Sansen, Proceedings of European Conference on design Automation, Amsterdam 1991, pp 475-479

[TRIC 91] "Modélisation du processus de conception des machines électriques : le système expert DAMOCLES", F. Trichon, Thèse de Doctorat, INPG, 1991

[ÜLER 95] "Design Optimization of electrical machines using genetic algorihtms", G. F. Üler, O. A. Mohammed & C. S. Koh, IEEE Transactions on Magnetics, vol. 31, n°3, May 1995, pp 2008-2011

[VEIN 60] "Synthesis of induction Motor Designs on a Digital Computer", AIEE Transactions, Power Apparatus and Systems, vol. 79, pp 12-18, 1960

[WAMB 96] "A family of matroid intersection algorithms for the computation of approximated symbolic network function", P. Wambacq, F.V. Fernandez & al., Proceedings of the International Symposium on Circuits & Systems, vol. 4, Atlanta May 1996, pp 806-809

[WATCOM] Watcom FORTRAN 77, guide d'utilisation, Watcom International Corporation, Waterloo, Ontario, Canada, 1993

[WURT 96] "Une nouvelle approche pour la conception sous contraintes de machines électriques", F. Wurtz, Thèse de Doctorat, INPG, 1996

[YONN 92] "Calculation of permanent magnet couplings", J.-P. Yonnet, 12th International Workshop on RE Magnets, Canberra, 1992, pp608-617

[YONN 98] "Automatic design of permanent magnet coupling", J.-P. Yonnet, P. Pandele, C. Coutel & F. Wurtz, 15th InternantionalWorkshop on RE Magnets, Dresdes, 1998, pp799-806

RESUME

CONTRIBUTION METHODOLOGIQUE A LA CONCEPTION SOUS CONTRAINTES DE DISPOSITIFS ELECTROMAGNETIQUES

Ce travail s'intéresse à la conception sous contraintes de dispositifs électromagnétiques à l'aide de modèles analytiques. Après avoir présenté le contexte de conception en génie électrique, et les problèmes inhérents au dimensionnement sous contraintes, notamment celui des systèmes d'équations implicites, l'étude présente une nouvelle architecture orientée objet pour le dimensionnement à l'aide de modèles analytiques. L'objectif est de créer un environnement souple et modulaire pour manipuler les équations analytiques de façon à utiliser toute l'information qu'elles contiennent. On veut par exemple ré-orienter le modèle étudié ou calculer les dérivées partielles symboliques des paramètres de sortie, ceci afin d'effectuer de la propagation de contraintes, du solver ou de l'optimisation sous contraintes. Un prototype informatique implanté est présenté.

Conception sous contraintes	Modèles analytiques	Propagation de contraintes
Optimisation	Equations implicites	Solver
Environnement orienté objet	Calcul symbolique	Dispositifs électromagnétiques

SUMMARY

METHODOLOGICAL CONTRIBUTION TO THE CONSTRAINED DESIGN OF ELECTRICAL DEVICES

This work deals with constrained design of electrical devices, using analytical models. After a presentation of the context of design in electrical engineering, and the description of the problems risen by constrained optimisation and design, especially the problem of implicit systems of equations, this study presents a new object-oriented architecture for the design with analytical models. The aim of the work is to create a modular an open environment that handles analytical equations and can extract and use all the information that they contain. For example, the environment should be able to modify the orientation of the model and to compute the symbolic derivative of the output parameters, in order to realise constraints propagation, solver and optimisation on the model. A prototype has been implanted and is presented.

Constrained design	Analytical models	Constraints propagation
Optimisation	Implicit equations	Solver
Oriented object environment	Symbolic calculus	Electrical devices