Dérivation Automatique pour le calcul des sensibilités appliqué au dimensionnement en génie électrique

Petre Enciu

INSTITUT POLYTECHNIQUE DE GRENOBLE

N° attribué par la bibliothèque

THÈSE pour obtenir le grade de **DOCTEUR DE GrenobleINP**

Spécialité : « Génie Électrique »

préparée au laboratoire de Génie Électrique de Grenoble (G2eLab) dans le cadre de **l'École Doctorale « Électrotechnique Électronique Automatique et Traitement du Signal »**

préparée et soutenue publiquement par

Petre Enciu

le 12 Octobre 2009

Titre :

Dérivation Automatique pour le calcul des sensibilités appliqué au dimensionnement en génie électrique

---

sous la direction du Pr. Laurent Gerbaud

JURY

M. Claude Marchand	Rapporteur
M. Stéphane Brisset	Rapporteur
Mme. Andrea Walther	Président
M. Laurent Krähenbühl	Éxaminateur
M. Vincent Leconte	Éxaminateur
M. Laurent Gerbaud	Directeur de thèse
M. Frédéric Wurtz	Codirecteur

A ma famille...

Table des matières

Table des figures

Remerciements

Pendent cette période de trois années passée au Laboratoire de Génie Electrique de Grenoble (G2ELab), les acteurs principaux qui ont participer à la finalisation de cette thèse ont été mes directeurs. Ils ont été toujours là à coté de moi (à savoir qu'ils travaillent dans la salle commune ensemble avec toutes les doctorants de l'équipe), ils m'ont beaucoup aidé et inspiré. Cette liste de satisfaction est longue, chacun de mes deux directeurs ont contribué petit à petit, chacun avec son brique, pour réussir à écrire des articles scientifiques, le manuscrit et de faire une soutenance qui m'as rendu très très fier. Donc, je leur dédie pas seulement ce paragraphe, mais cette thèse, je suis fier de rappeler leurs noms...je tiens donc à remercier avec fermeté en premier lieu Laurent Gerbaud et Frédéric Wurtz d'avoir proposé ce sujet et d'avoir accepté de m'encadrer en cette thèse. J'encourage tout futur docteur ou stagiaire de ne pas hésiter à dire "oui" à un sujet proposé par ces personnes.

Un autre épisode important de ma thèse était la collaboration avec Benoit Delinchant. Son esprit créateur souvent nous laisse sans réplique, donc préparez-vous bien avant de lui parler. Benoit Delinchant a été et il reste encore un défi pour moi. Une grosse partie du développement logicielle réalisé pendant cette thèse a été faite avec son aide et ses idées. Je lui remercie et je lui souhaite une belle carrière par la suite.

Le troisième épisode de mon travail était la collaboration avec Laurence Estrabaut. A savoir qu'elle a travaillé deux bureaux plus loin de moi et elle m'a montré toujours toute sa attention et son respect. Elle m'a beaucoup aidé à démarrer, à continuer et à finir mon travail (oui, elle a contribué à la correction de mon rapport). Je lui remercie profondément.

En gros, celle-ci était ma famille au cours de ces derniers trois années. Evidement l'équipe MAGE ne s'arrête ici. Je tiens à remercier Hieu, Bill, Lil, Hoa, Phoung, Ghaith, Lounes, Hervé, Didier, Asma, Mansour de L'Or, de m'avoir fait confiance et d'utiliser mes outils, ou d'avoir réussi à créer une ambiance plaisante dans la salle PC.

Je tiens à remercier tout particulièrement Mme. Andrea Walther pour sa disponibilité de présider ma soutenance. Pendent ces trois années je me suis beaucoup inspiré de son travail ou du travail de la belle communauté de la Dérivation Automatique qu'elle fait partie. De plus, ces années j'ai utilisé l'outil de Dérivation Automatique ADOL-C qui est d'une incroyable valeur, outil pour lequel Mme. Andrea Walther a contribué avec son

équipe de recherche. Sa présence à ma soutenance m'as rendu très fier et je lui souhaite une belle carrière de professeur à l'université de Paderborn ou ailleurs.

Je tiens à remercier à mes deux rapporteurs M. Claude Marchand et M. Stéphane Brisset d'avoir accepté de rapporter, de juger et finalement de contribuer à l'amélioration de mon manuscrit.

Je tiens à remercier à M. Laurent Krähenbühl pour son esprit au jour de la soutenance. J'ai été profondément impressionné par ses remarques, qui d'après moi n'ont été pas du tout facile de détecter dans ce rapport. Je lui souhaite une très bonne continuation dans sa carrière de recherche.

Finalement, je tiens à remercier toute la troupe, Octavian, Diana, Ghaith, Lucian, Maria, Anca, Monica, Cristi, Adrian. Merci Anca, Monica, Diana, Cristina, Cristi, Adrian pour votre assistance au moment de ma soutenance. J'espère vous rendre ce service au jour de vos soutenances en vous souhaitant une belle continuation. Je remercie également à tous ceux qui ont répondu à ma invitation de soutenance : Catalin, Anca, Radu, Flo, Ion, PB, Paul, Eric. Je remercie mes parents de m'avoir encourager à faire cette thèse et spécialement à Dia qui a réussi à me supporter ces dernières trois années. Je n'oublierai jamais ton effort.

Introduction générale

Les informations fournies par les dérivées sont indispensables dans un grand nombre de domaines scientifiques ou de l'ingénierie. Actuellement, une partie considérable des progrès réalisés dans de nombreux domaines au cours de la dernière décennie n'a été possible qu'en raison de l'existence des dérivées. Elles interviennent dans de nombreuses applications, comme par exemple dans la solution des systèmes d'équations implicites non linéaires ou la solution des systèmes d'équations différentielles ordinaires, d'équations à dérivées partielles ou d'équations différentielles algébriques. Elles sont également omniprésentes dans les domaines de l'analyse de sensibilités, les problèmes inverses et dans la conception multidisciplinaire basée sur l'optimisation.

Evaluer les dérivées pour une fonction donnée, parfois, se révèle être un défi. La Dérivation Automatique de programmes est une technique puissante d'évaluation des dérivées des fonctions décrites au moyen de programmes informatiques dans des langages de haut niveau comme FORTRAN, C ou C++. Contrairement aux approches traditionnelles, telles que la dérivation à la main, les différences finies ou le calcul formel, la Dérivation Automatique offre les bénéfices substantiels suivants : elle est précise, efficace en terme de coût de calcul, applicable pour une formule, ainsi que pour un programme de 100 000 lignes de code et ce pour un effort minimal de la part de l'utilisateur.

Dans le premier chapitre de ce travail, nous nous sommes attachés à illustrer le cadre de nos travaux : la conception en Génie Electrique. La conception, de manière générale, peut se définir comme étant un processus de création, de dessin, ou de projet, plus spécifiquement dans le cadre de la fabrication de produits. Un premier objectif principal est de trouver un compromis entre toutes les exigences formulées dans un cahier des charges. Cet objectif peut s'atteindre lors de l'exploration d'espaces des solutions possibles. L'étape suivante concerne le choix d'une solution et éventuellement sa vérification qui peut se réaliser par l'intermédiaire de prototypes. Toute la chaîne du processus de conception se finit par la réalisation des plans pour la fabrication effective du produit.

Dans la chaîne des étapes du processus de conception, nous sommes particulièrement intéressés par le dimensionnement qui consiste à définir l'objectif, sa résolution et la vérification. Plusieurs solutions sont envisageables pour la résolution de l'objectif. Parmi ces

solutions nous sommes particulièrement intéressés par les techniques d'optimisation basées sur le calcul des gradients. Notre but principal est d'évaluer ces gradients d'une manière formellement exacte, en employant la technique de Dérivation Automatique.

La Conception Assistée par Ordinateur (CAO) permet de concevoir dans des circonstances dont la complexité dépasse la capacité de l'être humain. En effet, le but de la CAO est d'aider au maximum le concepteur dans le sens d'éliminer tout ce qui est fastidieux, laborieux, répétitif, etc. Nous proposons d'intégrer nos travaux dans un environnement logiciel de CAO. Nous introduisons CADES dans le premier chapitre, qui est un environnement logiciel de dimensionnement, permettant de définir des modèles des dispositifs électriques dans un langage simple à appréhender par l'ingénieur concepteur. Une fois le modèle créé, CADES assiste le concepteur dans la phase d'optimisation, notamment dans la formulation du cahier des charges et d'application d'un algorithme d'optimisation.

Dans le deuxième chapitre, nous introduisons les fondements mathématiques qui se cachent derrière la technique de Dérivation Automatique. Cette technique applique d'une manière automatique la règle de dérivation des fonctions composées, sachant que tout programme informatique, quelle que soit sa complexité, exécute une séquence d'opérations mathématiques élémentaires, comme l'addition, la multiplication, etc. et/ou des appels à des fonctions mathématiques de base, comme `sin, cos, exp, log`, etc. Il existe deux modes principaux de propagation des dérivées : le mode direct et le mode inverse. Le mode direct propage efficacement les différentielles, alors que le mode inverse se relève être plus efficace pour l'évaluation des gradients. En s'appuyant sur ces principes, les outils de Dérivation Automatiques implémentent deux stratégies de base : la technique basée sur l'aspect de surcharge des opérateurs et la technique source-to-source. Ces stratégies sont discutées chacune dans ce chapitre. Ce chapitre se termine par un état de l'art de l'application de cette technique puissante en génie électrique.

Le troisième chapitre spécifie les éléments syntaxiques d'un langage de modélisation pour supporter les besoins de l'ingénieur concepteur dans la formulation des modèles de dimensionnement. Fondamentalement, il existe le besoin de décrire des équations analytiques explicites scalaires. A ceci s'ajoute la nécessité d'utiliser des vecteurs et de modéliser certains phénomènes par des algorithmes contenant des instructions complexes comme les branches conditionnelles `if...then...else`, les boucles répétitives `while...do, for`, etc. Le support de travail de ce chapitre est le langage existant sous CADES. Avant notre intervention, ce langage était construit sur beaucoup des restrictions d'ordre mathématique qui peuvent compliquer souvent la tâche de modélisation. Il donne la possibilité au concepteur de décrire des modèles de dimensionnement en s'appuyant sur des équations analytiques scalaires avec la possibilité d'utiliser des programmes décrits dans un langage externe de programmation complet, où le concepteur est lui même responsable de la mise en œuvre du calcul des gradients de ces derniers. Cette tâche est souvent très difficile et nous proposons à l'effectuer automatiquement en utilisant la Dérivation Automatique. Nous désirons aussi à enrichir le langage de modélisation initial (proposé pour décrire les modèles de dimensionnement) en introduisant la vectorisation.

Le quatrième chapitre propose des stratégies architecturales pour la mise en œuvre des compilateurs se conformant aux spécifications du langage de modélisation. L'idée est d'intégrer et de piloter automatiquement un outil de Dérivation Automatique dans le but de créer des programmes cible capables de calculer les gradients de modèles de dimensionnement, quel que soit leur complexité, afin de les exploiter en optimisation basée sur des algorithmes de type SQP. Le support de travail de ce chapitre est aussi l'environnement CADES.

Le dernière chapitre de nos travaux, propose des aspects mathématiques importants, afin d'exploiter efficacement la Dérivation Automatiques pour des méthodes numériques couramment utilisées dans la modélisation des dispositifs électriques. Ce chapitre propose deux catégories de méthodes numériques. La première catégorie est représentée par les méthodes de type Newton de résolution des systèmes d'équations implicites non linéaires et des méthodes d'intégration des fonctions. Ici on a des références précises pour valider les valeurs des dérivées. Ces références sont obtenues en appliquant une dérivation simple à la main et/ou en exploitant les propriétés mathématiques de ces méthodes numériques. La deuxième catégorie est représentée par les méthodes de résolution des systèmes d'équations différentielles, où la seule référence pour les valeurs des dérivées est la méthode des différences finies, avec une précision obtenue sur mesure. Toutes ces méthodes numériques, ainsi que leur dérivation, sont testées sur des problèmes de dimensionnement de dispositifs électriques.

Chapitre I

Contexte et enjeux

La réalité de l'activité de conception pour l'électrotechnicien : une activité d'Analyse Assistée par Ordinateur plutôt qu'une activité de Conception Assistée par Ordinateur

F. Wurtz - 1996

Sommaire

Résumé

Le contexte de ce travail se situe dans la conception des dispositifs électriques. Le processus de conception se réalise sur plusieurs étapes. Nous nous intéressons plus particulièrement à l'étape de dimensionnement, qui repose sur la détermination d'une

solution d'un cahier des charges. Plusieurs alternatives sont envisageables afin de résoudre un problème de dimensionnement. Parmi celles-ci, nous nous intéressons plus particulièrement à l'optimisation sous contraintes, qui connait aujourd'hui une notoriété croissante dans ce domaine.

En optimisation, les algorithmes déterministes basés sur le calcul des gradients représentent un bon compromis pour trouver une solution dans les limites exprimées par les contraintes. Une difficulté majeure peut apparaître à l'utilisation de ces algorithmes, liée à l'évaluation des gradients au cas où les modèles de dimensionnement sont décrits par des programmes informatiques. Parmi plusieurs techniques de calcul des gradients, une très bonne alternative est la Dérivation Automatique de programmes, qui représente une technique puissante d'évaluation des dérivées des fonctions décrites au moyen des programmes informatiques, écrites dans des langages existants comme FORTRAN et C/C++. Notre but est d'utiliser la Dérivation Automatique pour des dispositifs électriques décrits dans un langage informatique de modélisation beaucoup plus simple à appréhender par l'ingénieur électrotechnicien. Ce langage est intégré dans un environnement logiciel de conception. A ce niveau, il est nécessaire de proposer des solutions architecturales permettant d'intégrer et d'utiliser la Dérivation Automatique d'une façon transparente pour le concepteur de dispositifs du génie électrique.

I.1 Contexte du travail : La conception en génie électrique

Notre contexte de travail se situe dans le cadre de la conception des dispositifs électriques. Le processus de conception en général est représenté par une stratégie de résolution d'un besoin tout en respectant un ensemble d'exigences formulées dans un cahier des charges. Pour satisfaire ce dernier, le concepteur a recours à l'exploration d'espaces de solutions, la simulation et la vérification des solutions trouvées.

I.1.1 Présentation du processus de conception

Le processus de conception débute généralement par une analyse fonctionnelle permettant de définir les fonctionnalités que doit offrir le dispositif, afin de répondre aux besoins exprimés. Un cahier des charges est alors ébauché et puis enrichi au fur et à mesure du processus de conception. A partir de ce cahier des charges, le concepteur définit et choisit une structure, puis évalue certains paramètres suivant les besoins ou les objectifs à atteindre. Ceci correspond à l'étape de dimensionnement qui fait plus particulièrement l'objet de notre travail. Une fois l'étape de dimensionnement achevée, il reste à vérifier et valider le dispositif par simulation, établir les plans, construire et vérifier des prototypes et enfin entreprendre éventuellement la production en série.

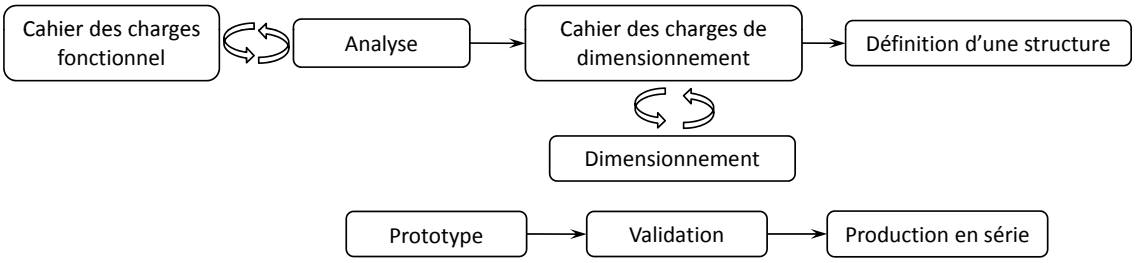


FIGURE I.1 – La conception des dispositifs électriques

I.1.2 Le dimensionnement

L'étape de dimensionnement commence donc dès qu'une structure du dispositif est choisie et elle se termine par la vérification et la validation de tous les paramètres intervenant dans la définition de la solution en itérant si besoin sur différentes phases. Ces phases de dimensionnement, telles que nous les appréhendons, sont illustrées sur la figure I.2.

Comme pour le processus de conception global, chaque étape du dimensionnement peut remettre en cause une ou plusieurs étapes précédemment effectuées. On peut dire que ces processus évoluent d'une manière imprévisible.

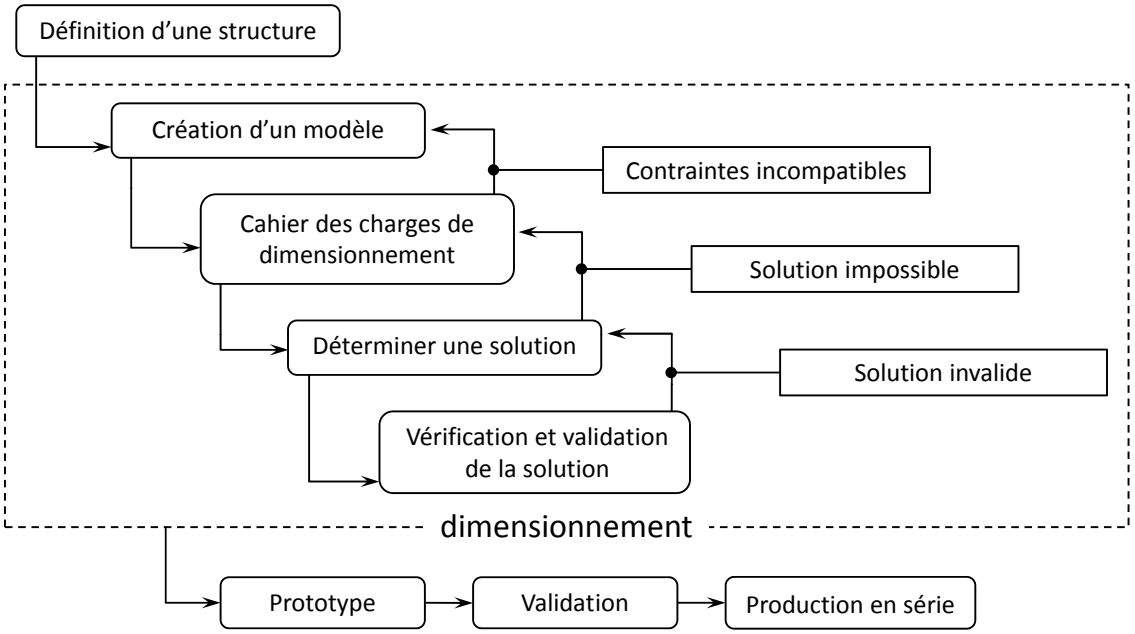


FIGURE I.2 – La conception des dispositifs électriques avec l'étape de dimensionnement détaillée

I.1.2.1 La modélisation

La première phase du dimensionnement des dispositifs électriques est la création d'un modèle, autrement dit la modélisation. Dans cette phase, le concepteur exprime son besoin [1] en fonction des paramètres de dimensionnement [2], en s'appuyant sur les connaissances de certaines lois qui gouvernent les différents phénomènes physiques qui apparaissent dans la structure du dispositif.

I.1.2.2 Le cahier des charges pour le dimensionnement

La deuxième phase du processus de dimensionnement est la création ou l'enrichissement du cahier des charges. Le cahier des charges, comme nous l'appréhendons dans ce travail, représente un ensemble des contraintes formulées pour le besoin (les variables de sortie) et un ensemble de dégrées de liberté formulés pour les paramètres (variables d'entrée). Ces contraintes peuvent être :

1. d'intervalle, ce qui est synonyme du fait qu'une variable d'entrée ou sortie n'est valide que si sa valeur finale se trouve entre deux limites numériques. Ces contraintes peuvent être continues ou imposées à prendre des valeurs discrètes. Nous nous intéressons seulement aux contraintes continues.

1. Le terme *besoin* n'est qu'une abstractisation du terme *variables de sortie* dans le contexte de la modélisation. Cela correspond aux objectifs à atteindre, tout en respectant les contraintes sur les critères de dimensionnement.

2. Le terme de *paramètre de dimensionnement* est utilisé dans ce travail aussi sous le nom de *variables d'entrée*

2. de valeur fixe, ce qui signifie qu'une variable de dimensionnement n'est valide que si elle prend une valeur numérique finale prescrite, avec une précision imposée.

Ceci dit, les contraintes exprimées dans le cahier des charges peuvent être de nature variée, par exemple :

- des contraintes d'encombrement : les dimensions géométriques du dispositif doivent respecter certains gabarits, ou son poids doit être inférieur à une certaine limite.
- des contraintes de compatibilité : le dispositif doit être compatible avec son environnement de fonctionnement (normes, diffusion de chaleur, etc.).
- des contraintes économiques : la production, ou l'utilisation du dispositif doivent être inférieures à un coût critique.
- des contraintes de qualité du résultat : le résultat fourni par le dispositif doit être suffisamment précis, rapide, stable, etc.

I.1.2.3 La résolution du modèle de dimensionnement

La troisième étape du dimensionnement est la détermination d'une solution. Dans la plupart des cas, cette étape implique d'atteindre un objectif tout en respectant les contraintes du cahier des charges. De manière générale, l'objectif est formulé en terme de minimisation d'une fonctionnelle de coût, comme par exemple un rendement, un volume, un temps de réponse, etc. Plusieurs approches, discutées en [93], peuvent être employées dans cette étape. Nous les citons maintenant :

1. l'approche procédurale, qui s'appuie sur de tests et des retours successifs en arrière afin de *"remettre en cause des choix non judicieux (F. Wurtz - [93])"*.
2. l'utilisation d'algorithmes d'optimisation afin de déterminer la solution du problème de dimensionnement [93][95][22][88].
3. les systèmes expert [43][44], qui mettent en œuvre des techniques d'intelligence artificielle.

Parmi ces trois approches, notre travail repose exclusivement sur les techniques d'optimisation, plus précisément sur l'optimisation sous contraintes en s'appuyant sur les modèles continus (pas de valeurs discrètes).

La dernière étape du processus de dimensionnement concerne la vérification et la validation de la solution. Cela permet de s'assurer que la solution trouvée respecte le cahier des charges imposé et que le paramétrage obtenu conduit à une structure valide. Par exemple, dans cette étape, il est nécessaire de vérifier que toutes les dimensions du dispositif sont conformes en réalisant des dessins plus ou moins sophistiqués. On exerce aussi de tolérancement, afin de vérifier la faisabilité industrielle des solutions obtenues.

I.1.3 La conception assistée par ordinateur

Le dimensionnement, tel qu'il est décrit dans les paragraphes précédents, est une étape complexe, qui demande souvent des retours en arrière imprévisibles. Tout ce processus

est indiscutablement réalisable avec une efficacité raisonnable à l'aide de l'ordinateur. Le concepteur crée souvent des programmes informatiques dans des langages existants, ou il fait appel à des logiciels de calcul ou de simulation. Avec une machine de calcul, il ne résout pas forcement les difficultés du dimensionnement. D'après F. Wurtz en [93] et comme nous le citons dans l'épilogue de ce paragraphe, l'utilisation de l'ordinateur est donc un moyen de calcul puissant, support à l'analyse pour la conception. En aucun cas il ne résout certaines difficultés de la conception sans une approche logicielle spéciale.

Dans ce paragraphe, nous introduisons le concept de conception assistée par ordinateur (CAO), qui consiste à employer des logiciels spéciaux permettant de décharger le concepteur, autant que possible, de tâches fastidieuses et répétitives. Au lieu de priver le concepteur d'une partie de son travail, les logiciels de CAO lui permettent de consacrer son temps aux étapes intéressantes du processus de conception, notamment celle de l'innovation [95][94].

Depuis 1990, les logiciels de CAO ont fait l'objet de plusieurs travaux dans notre équipe de recherche, concernant en effet des méthodes d'optimisation et d'aide à la formalisation et traitement des modèles de dimensionnement. Nous citons les travaux de F. Wurtz autour du logiciel *Pascosma* [93], les travaux de L. Gerbaud autour de *Gentiane* [45], les travaux de B. Delinchant sur la composition des outils de CAO [26], les travaux de E. Atienza pour le logiciel *Eden* [3], les travaux de V. Fischer autour de *CoreLab* [40], les travaux de D. Magot pour *CdiOptimizer*[64], les travaux de N. H. Hieu pour l'optimisation discrète [57], le logiciel commercial *Pro@Design* [27], etc.

I.2 L'optimisation sous contraintes

L'optimisation est une technique qui nécessite généralement la définition d'un problème sur lequel une méthode de résolution est employée, afin de déterminer une solution. Cette solution peut être unique ou non, suivant la méthode de résolution employée.

I.2.1 Le problème d'optimisation

Le cahier des charges d'un dispositif peut se formuler en un problème d'optimisation sous contraintes, comme celui de l'équation I.1 :

$$\begin{cases} \min\ J(p) \\ \text{avec} \quad g_j(p) \leq 0 \quad j = 1..l \\ \text{avec} \quad g_j(p) = 0 \quad j = l+1..m \\ p_{i_{min}} \leq p_i \leq p_{i_{max}} \quad i = 1..n \end{cases} \tag{I.1}$$

où J dénote la fonction objectif, l'ensemble de p_i dénote les paramètres de dimensionnement qui peuvent être soumises à n contraintes d'intervalle (n dégrées de liberté) et les g_j dénotent l contraintes d'intervalle et $m - l$ contraintes d'égalité.

I.2.2 Résolution du problème d'optimisation

Les méthodes les plus efficaces de résolution du problème I.1 sont les algorithmes d'optimisation, qui sont des techniques d'exploration automatique d'espaces des solutions du problème.

Afin de satisfaire complètement le problème de l'équation I.1, les algorithmes d'optimisation emploient dans la plupart des cas des techniques itératives ou évolutives. On peut distinguer deux familles importantes d'algorithmes d'optimisation, différentiées par leur façon d'explorer l'espace des solutions. Il s'agit des algorithmes déterministes et stochastiques.

I.2.2.1 Les algorithmes d'optimisation stochastiques

Les algorithmes stochastiques [3] explorent l'espace de recherche de la solution en s'appuyant sur des mécanismes de transition aléatoires. Les algorithmes évolutionnaires (génétiques) représentent une classe d'algorithmes largement utilisée, implémentant des techniques stochastiques. Les avantages principaux des algorithmes d'optimisation stochastiques sont les suivants :

1. ils ont une grande capacité à trouver un optimum global au problème d'optimisation.
2. ils nécessitent seulement le calcul de la fonction objectif et des contraintes, sans recours aux gradients.
3. la valeur optimale trouvée ne dépend pas des valeurs initiales des paramètres.

En contrepartie, leurs principaux inconvénients sont :

1. il est difficile de leur définir un critère d'arrêt.
2. ils sont très coûteux en évaluation du modèle de dimensionnement, donc en terme de temps d'exécution, effectuant généralement un grand nombre d'itérations.
3. ils sont incapables de satisfaire avec une précision élevée l'ensemble de contraintes ou l'optimum global.

Cette famille d'algorithmes ne fait pas l'objet de nos travaux. Elle a été appréhendée au sein du laboratoire, dans les travaux récents de N. H. Hieu [57] pour le dimensionnement des réseaux électriques embarqués. Il existe aussi des travaux en [78][76]

I.2.2.2 Les algorithmes d'optimisation déterministes

Les algorithmes d'optimisation déterministes [4] évaluent la solution du problème de l'équation I.1 par une évolution unique dans l'espace de recherche. Les algorithmes les

3. La *stochastique* est une technique mathématique appliquée à ce qui relève de l'aléatoire, à ce qui est statistique.

4. Le *déterminisme* est une notion philosophique selon laquelle chaque évènement est déterminé par des précédents, suivant une loi de causes à effets. Un algorithme d'optimisation est déterministe, lorsque toute exécution conduit au même résultat, dans les mêmes conditions.

plus discutés, implémentant la stratégie déterministe, sont les algorithmes d'optimisation basés sur le calcul des gradients.

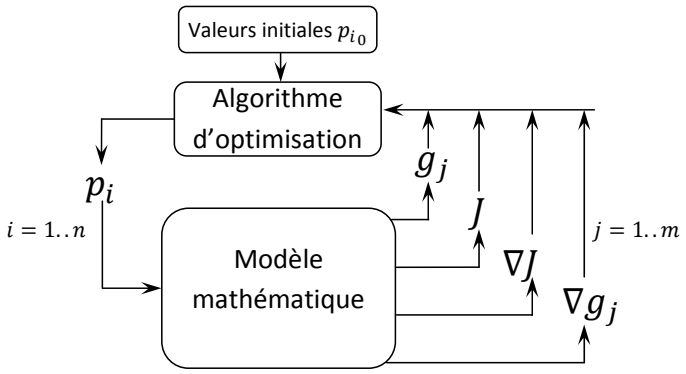


FIGURE I.3 – Le couplage entre un algorithme d'optimisation basé sur le calcul de gradients et le modèle de dimensionnement

Ces algorithmes utilisent les informations fournies par l'ensemble de gradients $\nabla J = \frac{\partial J}{\partial p_i}$ et $\nabla g_j = \frac{\partial g_j}{\partial p_i}$, $\forall i = 1..n$, $\forall j = 1..m$ pour amener de manière précise la solution à partir d'une valeur initiale des paramètres jusqu'à une valeur optimale. Les principaux avantages des algorithmes déterministes sont les suivants :

1. la rapidité de convergence due à l'exploitation des informations fournies par les gradients.
2. la capacité importante à satisfaire un grand nombre de contraintes avec une précision élevée.
3. le critère d'arrêt précis.
4. la capacité de satisfaire un cahier des charges fortement contraint (quelques centaines)

Cependant, ces algorithmes ont aussi les inconvénients suivants :

1. la probabilité élevée d'être piégés dans des optima locaux.
2. la sensibilité élevée à la précision des gradients ; le calcul précis des gradients représente souvent une tache difficile.
3. le besoin d'initialisation des paramètres de dimensionnement.

L'utilisation des algorithmes d'optimisation basés sur le calcul des gradients est le point de départ de la problématique de nos travaux. Pour ceci, nous utilisons particulièrement l'algorithme SQP [49] de minimisation séquentielle quadratique. Sachant que ces algorithmes ont une sensibilité élevée aux valeurs des gradients, notre but est d'évaluer ces derniers d'une manière exacte et efficace. Dans les paragraphes suivants, nous introduisons des techniques existantes permettant d'évaluer ces gradients.

I.3 Description des modèles de dimensionnement

Dans la phase de modélisation, le concepteur est amené à définir les lois mathématiques de variation de l'objectif et des variables contraintes dans le cahier des charges en fonction des paramètres de dimensionnement, afin d'assurer l'évolutivité des algorithmes d'optimisation. Plusieurs approches peuvent être utilisées et elles sont détaillées par la suite.

I.3.1 Nature du modèle

Dans la description des modèles de dimensionnement (la modélisation), une bonne alternative est de faire appel à des formules analytiques explicites. Cependant, il apparaît souvent le besoin d'utiliser des algorithmes implémentant des méthodes numériques utiles pour la description des certains phénomènes ou aspects mathématiques. Nous distinguons ainsi deux types d'éléments utilisés dans la description des modèles de dimensionnement, i.e. les formules analytiques explicites et les algorithmes numériques.

I.3.1.1 La modélisation analytique explicite

La modélisation analytique représente le moyen le plus simple de description des besoins dans un problème de dimensionnement. Dans ce paragraphe, nous introduisons les formules analytiques qui sont décrites avec :

- des opérateurs mathématiques, `+,-,*,/`.
- des fonctions mathématiques de base, comme `exp(), log()`, etc. et des fonctions trigonométriques, hyperboliques et leur fonctions réciproques.

Pour nous, ces expressions sont orientées ; la variable à gauche du signe "=" est affectée par la valeur de la formule à droite. Ceci représente la définition d'une expression analytique *explicite*. La variable du membre gauche de toute égalité exprime un besoin. Par exemple, dans l'expression suivante : $\Gamma = F \cdot \frac{d}{2} \cdot \cos(\alpha)$, Γ est une variable de sortie du modèle, déduite à partir de F, d et α. Cette expression est donc orientée en termes d'évaluation.

I.3.1.2 La modélisation semi-analytique

Dans la modélisation des dispositifs, la formulation analytique des variables de sortie n'est pas toujours possible et le concepteur est alors amené à utiliser des algorithmes numériques décrits dans un langage de programmation procédural. Nous introduisons la modélisation *semi-analytique*, qui combine les formules analytiques explicites avec les algorithmes numériques. Le besoin de tels algorithmes apparaît par exemple pour :

- la résolution des systèmes d'équations implicites.
- le calcul des intégrales.
- la résolution des systèmes d'état.

Ces trois problèmes, où le concepteur fait appel aux algorithmes numériques, sont d'un réel intérêt pour nos travaux et nous les aborderons plus en détail dans le chapitre V.

Pour nous, un algorithme numérique est une routine de calcul procédural qui pourra implémenter les aspects suivants :

- des boucles répétitives ; en général implémentées en faisant appel aux instructions tels que `do..while, for`.
- des branches conditionnelles, en utilisant les instructions `if..then..else, switch..case`.
- des fonctions récursives.
- des fonctions itératives.

La modélisation semi-analytique offre au concepteur plus de flexibilité dans le but de définir des modèles de dimensionnement, en éliminant la limitation liée à l'emploi de la modélisation purement analytique, ce qui n'est applicable que dans certaines situations.

I.3.2 Langage de description des modèles

Dans la description des modèles de dimensionnement, le concepteur a la responsabilité de définir la nature des variables en les regroupant selon le critère d'entrées/sorties. Ceci est imposé par le couplage entre le modèle de dimensionnement et l'algorithme d'optimisation, illustré sur la figure I.3. Afin de faciliter la programmation des modèles, nous proposons d'utiliser un *langage de modélisation*. La seule différence entre un langage de programmation classique et un langage de modélisation est que ce dernier donne la possibilité au compilateur associé, d'identifier automatiquement les paramètres de dimensionnement en entrée, de variables en sortie. Il n'y a pas des déclarations à faire. Ils n'implémentent pas des aspects de gestion de mémoire dynamique, des pointeurs, des références, de typage des données, etc. Ils peuvent s'apparenter au langage de programmation existant sous Matlab©[65]. Ainsi, ces langages sont souvent plus intuitifs et plus simples à appréhender par l'ingénieur concepteur électrotechnicien.

Plusieurs langages de modélisation ont été développées au cours de ces derniers années dans notre dans notre équipe de recherche. Nous rappelons les travaux de F. Wurtz [93], E. Atienza [3], L. Allain [1] et V. Fischer [40]. Dans nos travaux, nous utilisons le langage *sml*[5][38][39][23].

I.3.2.1 Le langage *sml* au début de nos travaux

Le langage de modélisation *sml*, tel qu'il existait au début de nos travaux, reposait sur l'existence des formules analytiques exclusivement scalaires. Les éléments intervenant dans la modélisation analytique sont plus détaillés au chapitre III.

Ce langage permettait de faire de la modélisation semi-analytique. Celle-ci fait l'objet de l'utilisation en *sml* de programmes écrits dans le langage Java [85] sous la forme de ce qu'on appelle *fonctions externes*. Un concepteur expert (de langage) définit un algorithme numérique dans une méthode Java ou tout autre langage en l'encapsulant en un exécutable Java. Ainsi, cet exécutable est utilisable sous la forme d'une *fonction externe du langage*

5. *sml* est l'acronyme de *System Modelling Language*.

sml dans la modélisation. Cette fonction est elle aussi scalaire avec des arguments scalaires. La dérivation du contenu de l'algorithme est à la charge du concepteur, ainsi que la programmation associée. Ces aspects sont expliqués avec plus de détails au chapitre III.

I.3.2.2 Les besoins d'évolution

Un inconvénient majeur du langage de modélisation *sml* est qu'il n'offre pas la possibilité d'utiliser des vecteurs. En ce qui concerne les types de données multiples, nous pouvons estimer avec fermeté, que les vecteurs ou les listes en général sont omniprésents dans la plupart des programmes décrivant des modèles de dimensionnement. Transcrire un vecteur dans le langage *sml* scalaire peut se faire "élément par élément", mais cela peut être à l'origine des lourdeurs et complications majeures qui augmentent vite avec le nombre d'éléments. Nous envisageons dans le chapitre III d'implémenter cette fonctionnalité importante.

Les algorithmes numériques utilisés dans la modélisation sous *sml* sont définis en Java avec la mention que le concepteur doit aussi implémenter le calcul des gradients en vue d'une optimisation SQP ultérieure. Ceci est un frein énorme à la description des algorithmes, d'autant que la dérivation peut être très délicate, s'appuyant dans la mesure du possible sur des propriétés mathématiques des aspects implémentés [20]. Notre but est d'éliminer complètement cette tâche laborieuse grâce à la mise en œuvre de la Dérivation Automatique de programmes.

I.4 Dérivation des modèles de dimensionnement

Dans ce paragraphe, nous sommes intéressés par le calcul des dérivées directionnelles de toute fonction vectorielle décrivant un modèle de dimensionnement :

$$F : \mathbb{R}^n \to \mathbb{R}^m \quad y = F(x) \tag{I.2}$$

donnée au moyen d'un programme informatique. Nous ajoutons la mention que dans nos travaux, n dénote le nombre des variables indépendantes (entrées) et m le nombre des variables dépendantes (sorties), donc pour lesquelles les dérivées partielles sont à calculer. En optimisation, les algorithmes basés sur le calcul des gradients utilisent les informations de certaines dérivées dans une direction, plutôt que la matrice jacobienne complète, F'. La dérivée directionnelle, $\dot{y}$, est donnée par :

$$\dot{y} = F'(x) \cdot \dot{x} = \lim_{h \to 0} \frac{F(x + h \cdot \dot{x}) - F(x)}{h} \tag{I.3}$$

où $\dot{x} \in \mathbb{R}^n$ est la direction de calcul des dérivées.

I.4.1 La méthode de différences finies

Une méthode classique capable d'approximer la limite mathématique de l'équation I.3 est la méthode des différences finies (DF). Il existe plusieurs formulations de cette méthode, dont la plus simple est décrite en I.4 :

$$\dot{y} = F'(x) \cdot \dot{x} \approx \frac{F(x + h \cdot \dot{x}) - F(x)}{h} \tag{I.4}$$

La précision des dérivées en I.4 dépend fortement du pas de dérivation h. Pour un pas trop petit, le principe de calcul de dérivées basé sur la méthode des différences finies peut être à l'origine d'instabilités numériques importantes engendrées par les erreurs de troncature, puisqu'il réalise une division par une quantité petite. Contrairement, pour un pas trop grand, la précision de la dérivée peut être gravement affectée. Trouver un bon compromis du pas de dérivation représente une tâche difficile. En [25], les auteurs spécifient certains critères pour déterminer la longueur optimale du pas pour un problème particulier. Cependant, il n'existe pas une formulation générale d'application de la méthode DF en optimisation reposant sur le calcul des gradients, cette méthode évaluant des dérivées avec une précision risquée.

I.4.2 Le calcul formel

Le calcul formel des dérivées est implémenté dans des packages comme Macsyma [72], Maple [61], Mathematica [92], etc. Il est bien connu que la formule de calcul de dérivées augmente vite en taille avec le nombre des variables indépendantes. Considérons, par exemple la fonction en I.5 :

$$F : \mathbb{R}^n \to \mathbb{R} \quad F(x) = \prod_{i=1}^{n} x_i = x_1 \cdot x_2 \cdot ... \cdot x_n \tag{I.5}$$

Le gradient correspondant se formule symboliquement comme en I.6 :

$$\begin{gathered} \nabla F(x) = \\ = (x_2 \cdot x_3 \cdot ... \cdot x_n, \\ x_1 \cdot x_3 \cdot .... \cdot x_n, \\ ....., \\ x_1 \cdot x_2 \cdot ... \cdot x_{i-1} \cdot x_{i+1} \cdot ... \cdot x_n, \\ ....., \\ x_1 \cdot x_2 \cdot .... \cdot x_{n-1}) \end{gathered} \tag{I.6}$$

Les expressions formelles des dérivées peuvent utiliser une quantité élevée de mémoire ou d'écran. Evidemment, il se peut que la formule de calcul des dérivées contienne des sous-expressions communes, comme $x_1 \cdot x_2 \cdot ... \cdot x_n$, qui sont évaluées pour chaque point particulier du ∇F. Les outils de calcul formel les plus utilisés implémentent en général des facilités élaborées pour simplifier les expressions mathématiques (optimisation de formules). Une

dérivée formelle, bien qu'exacte, peut être instable numériquement, indépendamment de sa taille [46]. En [46], les auteurs proposent un compromis entre le calcul numérique et formel pour un meilleur conditionnement des formules de calcul des dérivées.

I.4.3 La dérivation à la main

La dérivation à la main est une technique appliquée au prix de l'effort humain d'un mathématicien, ou d'un concepteur, qui programme le calcul de la dérivée en optimisant l'implémentation par l'utilisation d'un maximum *d'astuces*, ou de propriétés techniques, scientifiques et mathématiques. Elle reste indiscutablement la technique la plus efficace de calcul des gradients. Par exemple, dans le cas précédent de la fonction f décrite par l'équation I.5, un être humain peut, en connaissant parfaitement le domaine de définition de la fonction, appliquer des règles mathématiques et de programmation pour simplifier et optimiser le calcul des dérivées en les écrivant simplement (dans un programme informatique) comme :

$$\frac{\partial F(x)}{\partial x_i} = \frac{F}{x_i} \quad x \in \mathbb{R}^n, \quad x_i \neq 0 \tag{I.7}$$

où $F = F(x)$ représente une variable temporaire stockée en mémoire avant de réaliser le calcul effectif des dérivées. Cependant, l'effort humain d'implémentation du calcul des dérivées augmente vite avec la complexité de la fonction.

I.4.4 La Dérivation Automatique de programmes

La Dérivation Automatique (DA) de programmes apparaît être une bonne alternative permettant d'évaluer les gradients des fonctions décrites au moyen de programmes informatiques. Cette technique réduit énormément l'effort d'implémentation du calcul des gradients et ne connait pas de limitation en ce qui concerne la complexité du programme à dériver. Elle évalue les gradients avec une précision élevée, sans erreurs de troncature ou annulation. Aujourd'hui elle est implémentée dans un certain nombre d'outils, dont la plupart sont listés dans la base de données actualisée régulièrement en [8].

I.4.5 Bilan des techniques de dérivation

La dérivation à la main, malgré le fait qu'elle soit la technique la plus performante, exige un effort humain d'implémentation considérable. La méthode de différences finies évalue les dérivées avec une précision risquée. De ce fait, nous nous intéressons ici seulement à la Dérivation Automatique de programmes et au calcul formel.

I.4.5.1 Les limites du calcul formel

Une limitation majeure du calcul formel est qu'il fonctionne en général pour des formules mathématiques, plutôt que pour des algorithmes contenant des instructions (`while`, `for`, `if...then...else`). Il n'existe pas aujourd'hui de formalisme précis pour dériver

formellement une fonction décrite au moyen d'un programme informatique. Une des principales intentions de la Dérivation Automatique de programmes est de suppléer à cette lacune du calcul formel des dérivées. En général, cette technique n'est pas limitée par la complexité de la fonction à dériver. Elle supporte toute instruction du langage informatique dans lequel la fonction est formulée.

Les packages de dérivation formelle implémentent en général des techniques élaborées d'optimisation de calcul, sans lesquelles les formules augmentées des dérivées risquent d'utiliser une quantité énorme de mémoire. Ceci est accompli directement par la Dérivation Automatique, sans générer une expression extrêmement compliquée et puis de la simplifier. Cependant, le calcul formel produit des dérivées avec un coût de mémoire de calcul acceptable si le nombre de degrés de libertés est plutôt petit. Le fait de limiter la taille du modèle se fait souvent au détriment de la fidélité de la modélisation par rapport à la physique à modéliser. Ceci implique aussi la réduction de l'espace des contraintes, donc de l'espace de recherche d'une solution optimale. Les chances de trouver des solutions pour une structure qui s'écarte du dimensionnement initial souhaité sont ainsi considérablement diminuées. La dérivation formelle vectorielle est aussi lourde à mettre en œuvre, notamment lorsque l'on dérive par rapport à des composants de vecteurs.

I.4.5.2 Bilan des performances des techniques de dérivation

Il se peut que la Dérivation Automatique perde en efficacité de temps d'exécution de la fonction de calcul des dérivées devant le calcul formel, puisqu'en général, les moteurs de dérivation formelle réalisent des optimisations dans les formules (voir le tableau I.1).

Table I.1 – Classification des méthodes de dérivation

	Calcul formel	DF	DA	Dérivation à la main
Effort humain (utilisation)	☹	☺☺	☺	-
Effort humain (implémentation)	☹	☺☺	☹	☹
Efficacité (temps d'exécution)	☺	☺	☺	☺☺
Efficacité (mémoire)	☹	☺	☺	☺☺
Précision	☺☺	☹	☺☺	☺☺
Applicabilité^{I} aux programmes	☹	☺☺	☺☺	☺☺
Applicabilité^{II} aux programmes	-	☺☺	-	-

^{I} si le programme source, écrit dans un langage humain de la fonction à dériver, est disponible.
^{II} si le programme source, écrit dans un langage humain de la fonction à dériver, n'est pas disponible.

Cependant, les outils de Dérivation Automatique actuels assument que si un programme donné calcule la fonction à dériver efficacement, ils produisent un programme

de calcul des dérivées aussi performant que la dérivation formelle. De manière générale, cela peut se faire avec un effort humain, ou en utilisant des options d'optimisation des compilateurs. Cependant, nous verrons dans le chapitre II que certains outils de Dérivation Automatique peuvent accomplir directement certaines optimisations de ces calculs.

Sachant que les performances de la technique de Dérivation Automatique de programmes sont prometteuses, nous proposons dans ce travail de l'utiliser comme technique principale de dérivation. Nous explorons en détail cette technique au chapitre II et nous l'appliquons pour des problèmes précis de dimensionnement en génie électrique au chapitre V.

I.5 Architecture logicielle de dimensionnement

De manière générale, pour faire fonctionner un outil de Dérivation Automatique de programmes décrits dans un langage de modélisation, il est nécessaire de suivre les étapes suivantes :

1. se familiariser avec la syntaxe du langage de modélisation.
2. choisir un outil de Dérivation Automatique existant ou créer un nouveau dédié au langage de modélisation ; si l'outil existe déjà il est possible de sauter l'étape no. 3.
3. maîtriser des techniques d'implémentation d'outil de Dérivation Automatique.
4. connaître les principes mathématiques de la Dérivation Automatique.
5. savoir utiliser l'outil de Dérivation Automatique.
6. intégrer l'outil de Dérivation Automatique dans l'architecture du compilateur du langage de modélisation ou au niveau du programme cible.

I.5.1 La problématique

A partir de modèles mathématiques de dimensionnement décrits dans un langage source de modélisation, notre but est de définir un compilateur, afin de créer des exécutables (dans le langage cible) exploitables par les algorithmes d'optimisation. Il faut être donc capable de pouvoir dériver dans un premier temps des modèles analytiques scalaires. Ensuite, nous enrichissons la syntaxe du langage dans le but de vectorisation. Au niveau de la modélisation semi-analytique, notre deuxième but est de créer un compilateur unique pour la création des fonctions externes dérivées avec un outil de Dérivation Automatique de programmes. De cette façon, nous prendrons complètement en charge la tâche de dérivation des algorithmes utilisés en modélisation. Nous adoptons comme support de travail l'environnement logiciel de dimensionnement CADES.

I.5.2 CADES, un environnement logiciel dédié à la conception

Nous introduisons dans ce chapitre, l'environnement logiciel CADES [6][23] qui comprend plusieurs modules logiciels avec des fonctions précises pour le dimensionnement. Cet environnement a pour but d'assister le concepteur dans l'optimisation. Tout d'abord, un compilateur spécialisé pour le langage *sml*, génère un programme cible, encapsulant, conformément à une norme, le modèle mathématique. Le programme cible, se présentant sous forme d'un *composant de calcul*, est exécutable seulement par une machine virtuelle Java. Il s'agit donc d'un programme Java. Le composant de calcul implémente une série des fonctionnalités qu'on appelle *facettes* suivant les spécifications de sa norme. Une facette rend possible la connexion du composant de calcul avec ce qu'on appelle un *service*. Un service répond à un besoin particulier tout en utilisant une facette. Par exemple, le concepteur peut générer avec le compilateur, un composant de calcul en demandant une facette de calcul de gradients de son modèle mathématique. Ainsi, il peut bénéficier du service d'optimisation avec des algorithmes basés sur le calcul de gradients.

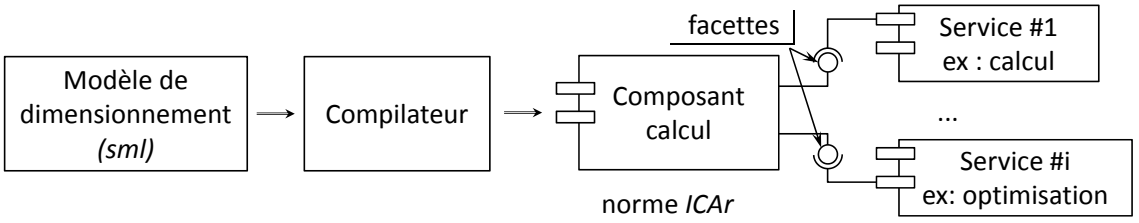


FIGURE I.4 – Paradigme *composant logiciel* implémenté sous CADES

Le compilateur du langage *sml*, dans l'état initial de nos travaux emploie automatiquement le calcul formel des dérivées et fonctionne donc seulement pour des modèles analytiques scalaires explicites. Ce compilateur supporte la modélisation semi-analytique par des fonctions externes décrites dans un langage externe (Java) avec la mention que le concepteur est responsable avec la programmation de leurs gradients.

Afin d'intégrer la Dérivation Automatique au niveau de cette architecture, nous sommes amenées à faire évoluer plus particulièrement l'architecture du compilateur et celle du composant de calcul. Ces aspects sont développés au chapitre IV.

I.6 Application des méthodes numériques à la modélisation des dispositifs électriques

Les méthodes numériques que l'on désire traiter grâce à nos travaux sont les algorithmes de résolution des systèmes d'équations implicites non linéaires de type Newton-Raphson, les méthodes d'intégration des fonctions, et des résolutions de systèmes d'équations différentielles ordinaires. Des problèmes de modélisation des dispositifs résolus par ces méthodes numériques sont présentés sur le tableau I.2 :

6. Le terme CADES est l'acronyme de *Component Architecture for Design of Engineering Systems*.

TABLE I.2 – Méthodes numérique couramment utilisées dans la modélisation des dispositifs électriques

Nature du dispositif	Modélisation	Méthode numérique
Electromagnétique statique	Réseaux des réluctances	Newton-Raphson
Micro-actionneurs	Loi de Biot-Savard, loi de Laplace	Intégration des fonctions
Actionneurs électromagnétiques	Equations différentielles ordinaires	Runge-Kutta, Développement en series de Taylor

Nous désirons résoudre les circuits réluctants, utilisés dans la modélisation des dispositifs électromagnétiques statiques, par les méthodes de type Newton-Raphson, sachant que ces circuits sont modélisés par des équations implicites non linéaires. Il existe aussi un besoin fort pour le calcul du champ magnétique par la loi de Biot-Savard, qui fait l'objet d'une intégrale à plusieurs dimensions dans tout point de l'espace. Cette application concerne notamment les micro-actionneurs électromagnétiques. Un autre besoin très important est constitué par la résolution des systèmes d'équations différentielles ordinaires, afin de modéliser les phénomènes dynamique intervenant dans les actionneurs ou dans les circuits électriques.

En vue d'optimisation SQP, les gradients de toutes ces méthodes numériques doivent être évalués. Dans le chapitre V nous proposons d'évaluer ces gradients en utilisant la technique de Dérivation Automatique de programmes dans une manière efficace, en exploitant si possible les diverses propriétés de ces méthodes.

I.7 Conclusion

Ce chapitre illustre le contexte de nos travaux, qui est la conception des dispositifs électriques. Au niveau du dimensionnement, nous sommes particulièrement intéressées par le calcul des gradients des modèles de dimensionnement en vue de l'optimisation en utilisant des algorithmes SQP. Nous citons plusieurs techniques de calcul différentiel, et nous nous focalisons sur la Dérivation Automatique de programmes qui sera utilisée au cours de ces travaux.

Nous proposons de nous appuyer sur CADES, un environnement logiciel dédié au dimensionnement et qui permet au concepteur, en s'appuyant sur un langage simple, de décrire des modèles de dimensionnement. La syntaxe de ce langage est très proche de celle existante sous Matlab ou Python, qui représentent des environnements de programmation de plus en plus utilisés par les ingénieurs électrotechniciens. Ce langage propose une modélisation par formules analytique scalaires et une alternative pour définir et utiliser des routines numériques introduites sous la forme de fonctions externes. Le concepteur bénéficie ainsi de plus de flexibilité dans la description des modèles de dimensionnement. L'inconvénient majeur du langage *initial sml* est lié à l'impossibilité d'utiliser des vecteurs. De plus, au niveau de la modélisation semi-analytique, le fait d'utiliser des fonctions ex-

ternes, impose au concepteur de définir aussi la fonction de calcul du gradient, qui pourra représenter une tache difficile dans la plupart des cas. Notre but ici est d'éliminer complètement ces deux inconvénients en s'appuyant sur la technique de Dérivation Automatique des programmes. Afin de réaliser cela, l'architecture initiale d'un outil de dimensionnement tel que CADES devra évoluer.

Chapitre II

Dérivation Automatique de programmes : Principes, Techniques et Applications

Automatic Differentiation operates by systematic application of the chain rule, ... applied not to symbolic expressions, but to actual numerical (floating point) values.

A. Griewank - 2000

Sommaire

Résumé

Ce chapitre étudie une technique qui s'appelle Dérivation Automatique, Dérivation Algorithmique, ou simplement Dérivation de Code qui permet d'obtenir les dérivées des fonctions décrites par des programmes informatiques dans des langages comme C/C++, FORTRAN. Les dérivées sont calculées sans erreurs de troncature ou approximation. Elles sont évaluées avec la précision de l'ordinateur effectuant les calculs.

Le principe qui gouverne cette technique est la règle de dérivation des fonctions composées. Ainsi, un programme informatique, quel que soit sa complexité, exécute une séquence d'opérations arithmétiques élémentaires comme l'addition ou des fonctions

élémentaires comme `sin()`, `exp()`, `log()`, *etc. En appliquant la règle des fonctions composées sur ces opérations, le calcul des dérivées d'un ordre arbitraire peut se propager automatiquement. De point de vue mathématique, il existe deux modes de propagation des dérivées, i.e. le mode inverse et le mode direct, illustrés dans la première partie de ce chapitre.*

La Dérivation Automatique de programmes est implémentée aujourd'hui dans un certain nombre d'outils. Ces outils sont distingués par deux techniques principales d'implémentation de la Dérivation Automatique, i.e. la technique basée sur l'aspect des surcharges d'opérateurs existant dans certains langages de programmation comme FORTRAN90 ou C++ et la technique source-to-source.

La dernière partie de ce chapitre est dédiée à un état de l'art de l'application de la Dérivation Automatique dans le domaine du Génie Electrique. Dans un nombre restreint de travaux, notamment dans l'optimisation, la simulation et l'analyse de sensibilités, les ingénieurs électriciens préfèrent remplacer les méthodes de dérivation traditionnelles par la Dérivation Automatique de programmes.

II.1 Introduction : L'intérêt de la Dérivation Automatique de programmes pour l'optimisation

Un nombre croissant d'applications informatiques scientifique ou de l'ingénierie exige le calcul des dérivées. Ce besoin survient au cas où certains paramètres d'un modèle mathématique non linéaire doivent être ajustés pour adapter certaines données, ou dans le cas d'une optimisation des performances en modifiant des variables de dimensionnement ou de contrôle. Quand les valeurs exactes des dérivées sont nécessaires, le calcul formel, la dérivation à la main et la Dérivation Automatique [1] restent les meilleurs choix.

Avec la même précision, la Dérivation Automatique demande beaucoup moins d'effort humain d'implémentation du calcul des dérivées que la dérivation à la main et s'applique pour des fonctions plus complexes que le calcul formel. Cependant, la seule limitation de la Dérivation Automatique est que le code source du programme informatique à dériver doit être disponible (voir tableau I.1).

Il faut être conscient que la Dérivation Automatique ne résout cependant pas les limites mathématiques de dérivation bien connues en certains points ou hors du domaine de définition des certains fonctions mathématiques. L'expérience montre que cette limite peut être éliminée à la formulation du cahier des charges d'optimisation, en posant des contraintes adéquates afin de se positionner dans le domaine de définition des ces fonctions. Un autre aspect important concerne les discontinuités créées par les instructions conditionnelles telles que `if...else...`. Dans ces cas, soit statistiquement les chances de tomber sur ces points sont faibles, soit elles influent peu l'optimisation et en conséquence on les ignore. Si leur influence est importante, elles exigent un traitement particulier. Par exemple pour le cas de la discontinuité montrée dans la figure II.1, il est plus convenable de considérer $\frac{\partial x}{\partial P} = 0$.

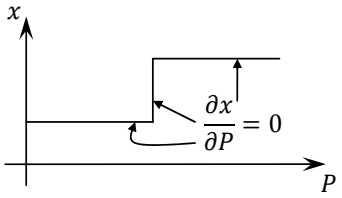


FIGURE II.1 – Fonction dérivable par morceaux

II.2 Les techniques de Dérivation Automatique

II.2.1 Le principe de la Dérivation Automatique

Le principe qui gouverne la Dérivation Automatique est l'utilisation de la règle mathématique des fonctions composées, appliquée au calcul des dérivées. Un outil de Dérivation

1. Dans nos travaux, nous utilisons le terme de *Dérivation Automatique* pour indiquer la *Dérivation Automatique de programmes*.

Automatique prend en entrée un programme informatique P, qui calcule pour un ensemble d'entrées données $x \in \mathbb{R}^n$, une fonction vectorielle $y = F(x) \in \mathbb{R}^m$:

$$P \equiv \{I_1; I_2; \cdots I_p;\} \tag{II.1}$$

Ainsi, le programme P peut être vu comme une séquence ordonnée d'instructions, $I_i,\ i = 1..p$. Chaque instruction du programme de calcul de la fonction à dériver correspond à une opération mathématique élémentaire (`+, -, *, /`) et/ou un appel à une fonction mathématique de base (`pow, exp, tan, sin`, etc.). En conséquence, la fonction F représente une composition de fonctions :

$$P = F = f_p \circ f_{p-1} \circ \cdots \circ f_1 \tag{II.2}$$

où chaque f_k représente la fonction mathématique implémentée par l'instruction I_k. La plupart des opérations mathématiques élémentaires sont dérivables sur tout leur domaine de définition (en général $\mathbb{R}$). Cependant, il y a des exceptions, par exemple la fonction racine carrée (`sqrt`) est définie en zéro, mais elle n'y est pas dérivable ; il en est de même pour la fonction valeur absolue. En [56], les auteurs précisent qu'il est peu probable de tomber sur ce genre des points de discontinuités dans une itération d'optimisation basée sur le calcul de gradients.

La Dérivation Automatique exploite la règle des fonctions composées pour la fonction F décrite par l'équation II.2, afin d'évaluer la matrice jacobienne en un point $x \in \mathbb{R}^n$.

$$F'(x) = f'_p(x_{p-1}) \cdot f'_{p-1}(x_{p-2}) \cdot \quad \cdots \quad \cdot f'_1(x_0) \tag{II.3}$$

où $x_0 = x$ et $x_k = f_k(x_{k-1}),\ \forall k = 1..p-1$ représente le vecteur de toutes les variables du programme après l'exécution de l'instruction I_k.

Pour faciliter la compréhension de la règle des fonctions composées, plus exactement la décomposition d'un programme de calcul d'une fonction en opérations mathématiques élémentaires, nous discutons cet aspect autour d'un exemple précis [19][39]. Prenons la fonction $F : \mathbb{R}^2 \to \mathbb{R}^2$ ([19]), décrite par l'algorithme de la figure II.2. Ici, x représente les variables indépendantes [2], y les variables dépendantes [3], a une variable intermédiaire et i une variable qui n'intervient pas dans le calcul, donc dans la dérivation. La fonction n'est pas définie en $x_2 = 0$ et n'est pas dérivable en $x_1 = 2$:

2. Nous associons le terme de *variable indépendante* à toute variable par rapport à laquelle nous désirons évaluer les dérivées partielles.

3. Nous associons le terme de *variable dépendante* à toute variable pour laquelle nous désirons évaluer les dérivées partielles.

```
function f(double x[])
return double y[]
_____________________________
begin function f:
double a, y[2];
int i;
_____________________________
if(x[1]>2)
        a = x[1] + x[2];
else
        a = x[1]*x[2];
for i = 1 : 2
        a = a * x[i];
y[1] = a/x[2];
y[2] = sin(a);
_____________________________
end function f;
```

FIGURE II.2 – Fonction de test pour la Dérivation Automatique

L'exécution de la fonction de la figure II.2 a du sens uniquement pour un ensemble de valeurs d'entrée x. Cet aspect est très important puisqu'on a souvent tendance à associer la Dérivation Automatique à la différentiation numérique, or il ne faut surtout pas la confondre avec la dérivation numérique qui repose sur la méthode DF exposée en I.4. En prologue de [52], les auteurs ont une discussion intéressante pour mieux distinguer les notions de *symbolique* et *numérique* dans le contexte de dérivation en général. Il y ressort que le terme *numérique* est souvent confondu avec *approximation*. Nous rappelons que la Dérivation Automatique *n'est pas symbolique*, mais une technique *numérique précise*. Elle calcule les *valeurs* des dérivées partielles de façon très précise d'une fonction en un point spécifié par l'utilisateur.

Supposons que la fonction décrite auparavant doive se calculer au point $x = (x_1 = 3.0, x_2 = 1.5)$. En conséquence, le programme compilé donné en II.2 exécutera dans l'ordre de v_{-1} à y_2, la séquence des opérations listées dans le tableau II.1. Nous adoptons la notation $v_{i\leq 0}$ pour les variables indépendantes et $v_{i>0}$ pour les variables intermédiaires.

TABLE II.1 – Séquence d'exécution de la fonction test

v_{-1}	=	x_1	=	3.0
v_0	=	x_2	=	1.5
v_1	=	$v_{-1} + v_0$	=	4.5
v_2	=	$v_{-1} \cdot v_0$	=	4.5
v_3	=	$v_1 \cdot v_2$	=	20.25
v_4	=	v_3/v_0	=	13.5
v_5	=	$\sin(v_3)$	=	0.98552
y_1	=	v_4	=	13.5
y_2	=	v_5	=	0.98552

Ce qui donne au final :

$$y_1 = (x_1 + x_2) \cdot x_1 \cdot x_2/x_2$$
$$y_2 = \sin[(x_1 + x_2) \cdot x_1 \cdot x_2]$$

Pour la fonction de test, décomposée en opérations élémentaires dans le tableau II.1, nous n'effectuons pas intentionnellement les simplifications pour y_1, afin d'illustrer que la Dérivation Automatique dérive en théorie le programme tel qu'il est écrit, sans effectuer des optimisations dans les formules.

La fonction de test contient deux branches conditionnelles (`if...then...else`) et une instruction itérative répétitive(`for ...`). Un aspect intéressant, concernant les branches conditionnelles, est le fait que le programme de calcul de la fonction exécute seulement la branche qui satisfait la condition imposée. Dans notre cas, c'est la branche correspondante à `if` puisque $x_1 = 3.0 > 2$. De manière générale, un programme s'exécute *par morceaux*[4], toutes les branches étant déroulées. De même, la Dérivation Automatique de programmes contenant des branches conditionnelles sera effectuée par morceaux.

Plusieurs approches existent pour représenter une séquence d'exécution d'un programme informatique. En [48], les auteurs évoquent une série d'alternatives bien adaptées pour modéliser la règle des fonctions composées utilisée par la Dérivation Automatique. Parmi elle, nous citons l'approche par dualité et celle par substitutions régressives, évoquées aussi en [40]. Cependant, ces deux approches ne sont plus utilisées dans les publications spécialisées les plus récentes. L'approche qui est couramment utilisée aujourd'hui, peut être grâce à sa simplicité, est l'approche par graphe de calcul [7], aussi évoquée en [48] sous le nom d'approche par graphe de Kantorovitch.

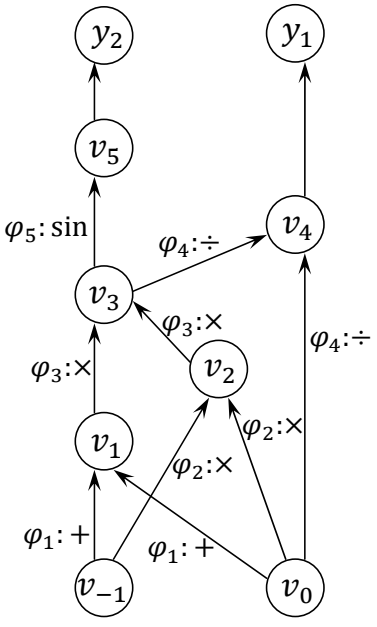


FIGURE II.3 – Approche de représentation par graphe de calcul de la fonction de test

La représentation basée sur le graphe de calcul pour un problème particulier exige la décomposition en opérations et fonctions mathématiques élémentaires. Pour la fonction de test décrite sur la figure II.2, décomposée dans le tableau II.1, le graphe de calcul associé

4. En adoptant une traduction du terme anglais *piecewise*.

pour $x[1] > 2$ se construit comme montre la figure II.3.

Chaque nœud du graphe représente une variable indépendante ou intermédiaire et un arc allant du nœud v_j au nœud v_i représente une fonction mathématique élémentaire qui relie les nœuds en question. Par la suite, nous reprenons la notation adoptée en [52], pour caractériser toute relation nœud-arc :

$$v_i = \varphi_i(v_j)_{j \prec i} \quad i > 0, j < i \tag{II.4}$$

qui signifie que chaque valeur v_i (avec $i > 0$) est obtenue en appliquant une fonction élémentaire φ_i pour un ensemble d'arguments v_j (avec $j < i$). La relation de précédence, $j \prec i$ signifie que v_i dépend directement de v_j.

La Dérivation Automatique concerne seulement les fonctions dont l'exécution est représentable par la relation II.4 et évaluées pour un argument particulier. Cette règle est applicable aux fonctions itératives ou récursives. Ces aspects se compliquent pour les fonctions implicites, mais il existe des solutions pour gérer ces dernières, comme nous le discutons dans le chapitre V. Toutes les structures de contrôle disponibles dans les langages de programmation sont conformes avec cette règle. Grâce à ce formalisme, il ressort que la Dérivation Automatique connaît peu de limitations en ce qui concerne la complexité du programme décrivant une fonction.

II.2.2 Les modes de Dérivation Automatique

La formule II.4, permet de calculer la valeur en un nœud du graphe de calcul. Cette formule est très simple à dériver puisque toute fonction φ_i représente une fonction mathématique élémentaire. Cependant, la propagation des dérivées, par exemple de v_i vers $v_{k \neq i}$ présente certaines spécificités. Ce paragraphe étudie les approches mathématiques de propagation des dérivées en Dérivation Automatique. Il en existe principalement deux. La première s'appelle *le mode direct*[5] de dérivation applicable efficacement au calcul des différentielles. Elle réalise le calcul des dérivées dans la même direction que le calcul de la fonction, i.e. du bas vers le haut sur le graphe de calcul, sachant qu'on représente toujours les variables indépendantes (les $v_{i \leqslant 0}$) au bas de ce graphe. La deuxième approche est *le mode inverse* applicable efficacement au calcul des gradients. Elle exige une évaluation de la fonction en *mode direct* et une propagation de valeurs des dérivées en mode inverse (du haut vers le bas du graphe de calcul).

II.2.2.1 Le mode direct de Dérivation Automatique

Le mode direct propage les valeurs des dérivées dans la même direction que les valeurs de la fonction. L'équation II.3 sert de base à cette approche. Les dérivées f'_k qui apparaissent dans le membre droite de cette équation sont des matrices de taille d'ordre

5. *Le mode direct* de dérivation existe aussi en littérature sous le nom de *mode tangent*. En anglais il existe plusieurs formulations pour ce terme, comme par exemple *forward mode*, ou *forward accumulation*, *tangent propagation*, ou *forward/tangent sweep*.

$m \times n$. Leur multiplication est très coûteuse en temps CPU et en quantité de mémoire requise. Heureusement, il existe des applications qui exigent seulement des projections de la matrice jacobienne. Ces projections sont les *différentielles* ou les *dérivées directionnelles* de la fonction à dériver. Elles se calculent pour une direction $\dot{x} \in \mathbb{R}^n$ spécifiée dans l'espace des entrées [6]. Dans la plupart des cas, ces directions représentent une base Cartésienne [7] qui donne les dérivées partielles de toutes les sorties par rapport à une seule entrée. De manière générale, les dérivées directionnelles sont données dans l'équation II.5.

$$F'(x) \cdot \dot{x} = f'_p(x_{p-1}) \cdot f'_{p-1}(x_{p-2}) \cdot \cdots \quad \cdot f'_1(x_0) \cdot \dot{x} \tag{II.5}$$

et peuvent se calculer en effectuant des multiplications successives *matrice* $\times$ *vecteur* de la droite vers la gauche. Cet aspect est plus efficace que d'effectuer des multiplications *matrice* $\times$ *matrice*. Les multiplications dans l' équation II.5 commencent donc par $f'_1(x_0)$, qui est la dérivée de la première instruction du programme et se finissent avec $f'_p(x_{p-1})$, la dérivée de la dernière instruction. De cette manière, ces dérivées sont propagées à partir du bas du graphe de calcul vers le haut. C'est le principe du mode direct de dérivation.

Le formalisme qui gouverne cette propagation est donné dans les équations II.6 en s'appuyant sur l'équation générale II.4 de propagation du calcul d'une fonction implémentée au moyen d'un programme informatique.

$$\begin{cases} v_i = \varphi_i(v_j)_{j \prec i} \quad i > 0, j < i \\ \dot{v}_i = \displaystyle\sum_{j \prec i} \frac{\partial}{\partial v_j} \cdot \varphi_i(v_j) \cdot \dot{v}_j \end{cases} \tag{II.6}$$

Il y apparaît que le mode direct de dérivation exige le calcul préalable de toutes les valeurs v_j (intermédiaires ou de sortie de la fonction F). Donc chaque instruction originale du programme principal est suivie par l'instruction de calcul des dérivées. Ainsi on constate que le mode direct calcule les valeurs de la fonction et ses dérivées partielles en parallèle.

Pour faciliter la compréhension de ce mode de dérivation, nous l'appliquons sur la fonction test donnée sur la figure II.3, en évaluant uniquement les dérivées partielles par rapport à la variable x_1. Ainsi $\dot{x} = \left[\frac{\partial x_1}{\partial x_1} = 1, \frac{\partial x_1}{\partial x_2} = 0\right]$. Le tableau II.2 illustre pas-à-pas la séquence des opérations effectuées par ce mode de dérivation.

6. $\dot{x} = \left[\frac{\partial x_i}{\partial x_j}\right],\ i \in \{1..n\},\ \forall j = 1..n.$

7. La $i^{ème}$ base Cartésienne, dans un espace à n dimensions, est représentée par le vecteur $v^n = \{0, 0, \cdots, 1, \cdots, 0\} \in \mathbb{R}^n,\ \ v^n(i) = 1, v^n(k \neq i) = 0, \forall k, i \leqslant n.$

TABLE II.2 – La séquence d'opérations appliquées par le mode direct de dérivation sur la fonction test

v_{-1}	=	x_1	=	3.0
$\dot{v}_{-1}$	=	$\dot{x}_1$	=	1.0
v_0	=	x_2	=	1.5
$\dot{v}_0$	=	$\dot{x}_2$	=	0.0
v_1	=	$v_{-1} + v_0$	=	4.5
$\dot{v}_1$	=	$\dot{v}_{-1} + \dot{v}_0$	=	1.0
v_2	=	$v_{-1} \cdot v_0$	=	4.5
$\dot{v}_2$	=	$\dot{v}_{-1} \cdot v_0 + \dot{v}_0 \cdot v_{-1}$	=	1.5
v_3	=	$v_1 \cdot v_2$	=	20.25
$\dot{v}_3$	=	$\dot{v}_1 \cdot v_2 + \dot{v}_2 \cdot v_1$	=	11.25
v_4	=	v_3/v_0	=	13.5
$\dot{v}_4$	=	$(\dot{v}_3 - \dot{v}_0 \cdot v_4)/v_0$	=	7.5
v_5	=	$\sin(v_3)$	=	0.98552
$\dot{v}_5$	=	$\cos(v_3) \cdot \dot{v}_3$	=	1.9072
y_1	=	v_4	=	13.5
$\dot{y}_1$	=	$\dot{v}_4$	=	7.5
y_2	=	v_5	=	0.98552
$\dot{y}_2$	=	$\dot{v}_5$	=	1.9072

Différentes études concernant la complexité des modes de dérivation sont discutées en [52]. Il en ressort, sans entrer dans les détails, que le temps d'évaluation déployé par le mode direct est un multiple $\omega_{dir} \in [2, 5/2]$ du temps d'évaluation de la fonction elle-même :

$$TIME\{F(x), F'(x) \cdot \dot{x}\} = \omega_{dir} \cdot TIME\{F(x)\} \tag{II.7}$$

où $TIME\{F(x), F'(x) \cdot \dot{x}\}$ est le temps d'évaluation de la fonction F et de ses dérivées partielles, correspondant au total du mode direct de dérivation. Le facteur ω_{dir}, est comparable au facteur d'efficacité caractérisant la méthode de différences finis, sachant que cette méthode évalue les valeurs des dérivées à un cout deux fois plus élevé ($\omega_{DF} = 2$) que l'évaluation de la fonction elle-même.

Une étude similaire concernant la mémoire utilisée pour chaque allocation des réels est aussi discutée en [52]. L'équation II.8 montre que le mode direct est deux fois plus cher en quantité de mémoire utilisée que la fonction de base.

$$MEM\{F(x), F'(x) \cdot \dot{x}\} = 2 \cdot MEM\{F(x)\} \tag{II.8}$$

Il existe cependant des cas où l'utilisateur peut demander les dérivées partielles sur plusieurs directions, p, $p \leq n$. Ceci est ce qu'on appelle le *mode direct vectoriel* de dérivation. Cette fois, $\dot{x}$ dans l'équation II.5 n'est pas un vecteur mais une matrice de taille $n \times p$. Ca revient évidemment à calculer successivement des produits $matrice \times matrices$, ce qui est inévitable. A l'extrême, la matrice jacobienne totale, qui demande la quantité des opérations la plus élevée se calcule en adoptant pour $\dot{x}$ la matrice unité d'ordre n ($\dot{x} = \mathbb{I}_n$).

II.2.2.2 Le mode inverse de Dérivation Automatique

Dans la même philosophie, le mode inverse [8] exige un calcul préalable de la fonction (qui peut s'effectuer seulement en mode direct). Contrairement à la propagation directe, le mode inverse réalise la propagation des dérivées dans le sens inverse du calcul de la fonction.

Nous avons montré pour le mode direct que l'équation générale II.3 de propagation des dérivées est inefficace à s'appliquer sous cette forme. Pour l'optimisation mono-objectif ou les problèmes inverses, le i^{eme} gradient de la fonction F est souvent demandé plutôt que la matrice jacobienne. Ce gradient peut se calculer pour une direction $\bar{y} \in \mathbb{R}^m$ spécifié dans l'espace des sorties [9]. Cette direction est la i^{eme} base Cartésienne transposée. Le i^{eme} gradient se calcule selon l'équation [10] II.9.

$$\nabla F_i(x) = F'^{\top}(x) \cdot \bar{y} = f_1'^{\top}(x_0) \cdot \quad \cdots \quad \cdot f_{p-1}'^{\top}(x_{p-2}) \cdot f_p'^{\top}(x_{p-1}) \cdot \bar{y} \tag{II.9}$$

Sous cette forme, on est amenés à effectuer efficacement des produits $matrice \times vecteur$ de la droite vers la gauche. Ces opérations commencent par la prise en compte de $f_p'^{\top}(x_{p-1})$ qui est la matrice jacobienne transposée de la dernière instruction du programme est se finit par la première instruction. Donc ces dérivées sont propagées à partir du haut du graphe de calcul vers le bas. C'est le principe du mode inverse de dérivation.

La direction $\bar{y}$ dans l'espace de sortie est appelée *variable adjointe*. Pour ceci, ce mode de dérivation réalise en effet la propagation des variables adjointes. Le résultat final donne les dérivées partielles d'une sortie par rapport à toutes les entrées (un gradient). Le formalisme qui gouverne la propagation inverse est donné dans l'équation [11] II.10, en s'appuyant sur l'équation générale II.4 de propagation du calcul d'une fonction implémentée au moyen d'un programme informatique.

$$\bar{v}_j = \sum_{i \succ j} \bar{v}_i \cdot \frac{\partial}{\partial v_j} \cdot \varphi_i(v_j) \tag{II.10}$$

Le mode inverse de dérivation exige le calcul préalable de toutes les variables v_j (intermédiaires ou de sortie de la fonction F). L'exécution du programme principal est suivie de celle de la procédure de calcul des dérivées. Ainsi, le mode inverse calcule toutes les valeurs de la fonction, et puis ses gradients.

Pour illustrer ceci, nous appliquons le mode inverse de dérivation sur la fonction test donnée dans la figure II.3, afin d'évaluer les dérivées partielles seulement de la sortie y_2

8. Pour *le mode inverse* de dérivation il existe plusieurs formulations en anglais comme *reverse mode backward accumulation*, *reverse propagation*, *reverse differentiation*, ou *adjoint differentiation mode*. A ne pas confondre avec *backward differentiation* qui représente un terme utilisé pour la discrétisation des équations différentielles ordinaires.

9. $\bar{y} = \left[\frac{\partial y_i}{\partial y_j}\right],\ i \in \{1..m\},\ \forall j = 1..m.$

10. $X^{\top}$ représente la matrice transpose de X.

11. La relation $i \succ j$ dénote que i est le successeur direct de j dans le graphe du calcul de la fonction de base.

par rapport aux variables x_1 et x_2. Ainsi, on pose $\bar{y} = \left[\frac{\partial y_2}{\partial y_1} = 0, \frac{\partial y_2}{\partial y_2} = 1\right]$. Le tableau II.3 illustre pas-à-pas la séquence d'opérations effectuée par ce mode de dérivation.

TABLE II.3 – La séquence d'opérations appliquée par le mode inverse de dérivation sur la fonction de test

Fonction de base			Mode inverse				
v_{-1}	=	x_1=3.0	$\bar{v}_5$	=	$\bar{y}_2$	=	1.0
v_0	=	x_2=1.5	$\bar{v}_4$	=	$\bar{y}_1$	=	0.0
v_1	=	$v_{-1} + v_0 = 4.5$	$\bar{v}_3$	=	$\bar{v}_5 \cos(v_3) + \bar{v}_4 / v_0$	=	0.16952
v_2	=	$v_{-1} \cdot v_0 = 4.5$	$\bar{v}_2$	=	$\bar{v}_3 \cdot v_1$	=	0.76288
v_3	=	$v_1 \cdot v_2 = 20.25$	$\bar{v}_1$	=	$\bar{v}_3 \cdot v_2$	=	0.76288
v_4	=	$v_3 / v_0 = 13.5$	$\bar{v}_0$	=	$\bar{v}_1 + \bar{v}_2 \cdot v_{-1} - \bar{v}_4 \cdot v_3 / v_0^2$	=	3.05152
v_5	=	$\sin(v_3) = 0.98552$	$\bar{v}_{-1}$	=	$\bar{v}_1 + \bar{v}_2 \cdot v_0$	=	1.9072
y_1	=	$v_4 = 13.5$	$\bar{x}_2$	=	$\bar{v}_0$	=	3.05152
y_2	=	$v_5 = 0.98552$	$\bar{x}_1$	=	$\bar{v}_{-1}$	=	1.9072

De même que pour le mode direct, l'utilisateur peut demander le calcul en mode inverse d'un nombre q, $q \leq m$ gradients. Cette fois $\bar{y}$ n'est plus un vecteur des variables ajointes, mais plutôt une matrice de taille $q \times m$. Ceci est ce qu'on appelle *le mode inverse vectoriel* de dérivation.

II.2.2.3 L'implémentation du mode inverse de Dérivation Automatique - le checkpointing

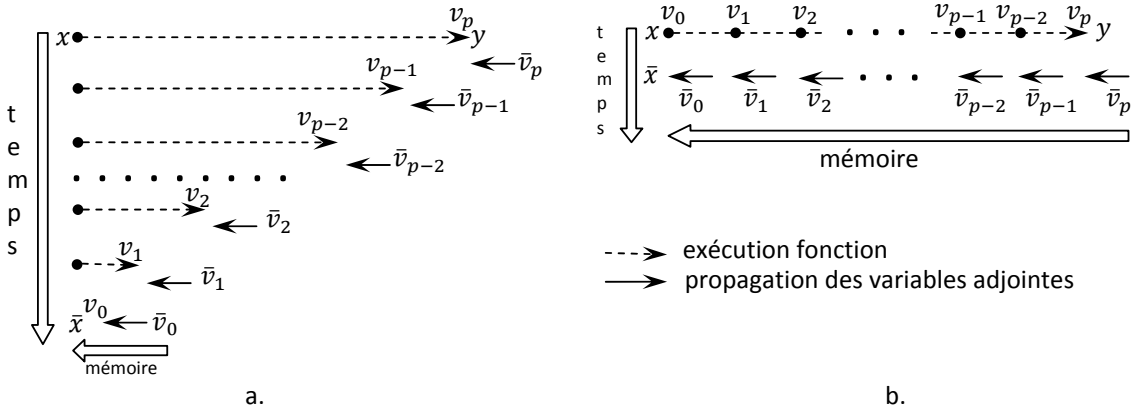


FIGURE II.4 – L'implémentation du mode inverse de dérivation ; a. la stratégie *Recompute All* ; b. la stratégie *Store All*

Nous avons montré que le mode inverse de dérivation exige le calcul préalable de toutes les variables intermédiaires $v_{i>0}$ intervenant dans le calcul des dérivées des sorties. Ces variables sont obtenues en exécutant en préalable la fonction à dériver. Le problème qui se pose est le stockage de toutes ces variables en mémoire avant de réaliser la dérivation effective en mode inverse. Par exemple, pour une fonction itérative, exécutant des milliers d'itérations, il faut stocker autant de valeurs. La quantité de mémoire utilisée risque de

surcharger assez vite les ressources disponibles de la machine de calcul. D'ailleurs, il est fortement recommandé d'éviter ce mode de dérivation pour des fonctions itératives.

Deux stratégies extrêmes peuvent exister pour l'implémentation de ce mode de dérivation. La stratégie *Recompute All* (voir figure II.4a) repart autant de fois du début du programme, afin de recalculer chaque variable intermédiaire. Le coût de cette stratégie est maximal en temps d'exécution. La stratégie *Store All* (voir figure II.4b) mémorise toutes les variables intermédiaires. Cette stratégie utilise une quantité de mémoire maximale.

En réalité aucun de ces deux mécanismes n'est implémenté par le mode inverse de dérivation [56]. Le *checkpointing* est une stratégie permettant de trouver un bon compromis entre la mémoire utilisée et le temps d'exécution. Cette stratégie consiste à exécuter le programme à dériver en deux phases, i.e. une *passe avant* et une *passe arrière*. Un checkpoint est un fragment C du programme pour lequel la passe avant n'effectue aucun stockage. Lorsque la passe arrière atteint le checkpoint on relance une exécution de C avec stockage et ainsi la passe arrière peut reprendre. La figue II.5 illustre ces aspects dans l'ordre chronologique de déroulement des événements.

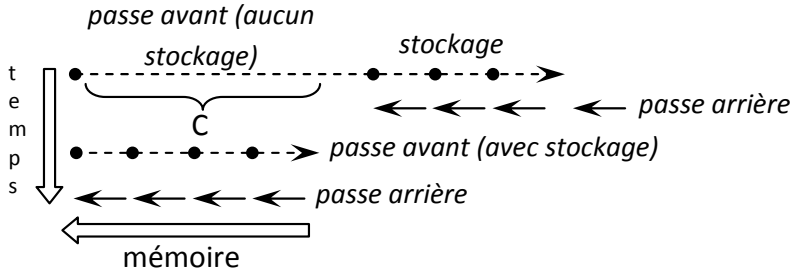


FIGURE II.5 – Les étapes du checkpointing

Au prix d'une double exécution de C et de la mémorisation de l'état au début de C, on a réduit la quantité maximale de stockage utilisée à l'issue de la passe avant.

II.2.2.4 Bilan des modes de propagation des dérivées

Dans ce paragraphe, nous montrons ce qu'il faut retenir pour les modes de propagation des dérivées. Ainsi, le mode direct est souhaitable pour :

- le calcul des différentielles, c'est-à-dire pour les dérivées partielles de toutes les sorties de la fonction par rapport à une seul entrée ; ceci est le *mode direct scalaire*.
- le calcul de la matrice jacobienne d'une fonction si le nombre de variables dépendantes est supérieur au nombre de variables indépendantes ($m > n$) ; ceci est le *mode direct vectoriel*.

En contrepartie, le mode inverse est recommandé pour :

- le calcul d'un gradient, c'est-à-dire pour les dérivées partielles d'une sortie de la fonction par rapport à toutes les entrées ; ceci est le *mode inverse scalaire*.
- le calcul de la matrice jacobienne d'une fonction si le nombre de variables dépendantes est inférieur au nombre de variables indépendantes ($m < n$) ; ceci est le *mode*

inverse vectoriel.

Nous ajoutons la mention que le mode inverse de Dérivation Automatique n'est pas souhaitable pour la dérivation des fonctions effectuant un grand nombre d'itérations, puisque toutes les variables intermédiaires doivent être stockées. Cela est valable même si le mode inverse implémente une stratégie de checkpointing performante.

II.2.3 Les techniques d'implémentation de la Dérivation Automatique

La Dérivation Automatique est implémentée depuis une trentaine d'années dans des outils dont la plupart sont listés dans [8]. La plupart de ces outils sont gratuits, open-source, ou utilisables directement dans une interface Web (par exemple TAPENADE [55]). Deux techniques principales sont à la base de la création de ces outils et elles utilisent le même principe mathématique pour l'évaluation des dérivées, la règle de fonctions composées, présentée dans le paragraphe II.2.1. Nous présentons ces deux techniques dans les paragraphes suivantes.

II.2.3.1 La technique de transformation source-to-source

Comme son nom l'indique, un outil implémentant la technique *source-to-source* prend en entrée un programme initial, décrit dans un langage source, et génère un deuxième programme augmenté, appelé programme cible, capable d'évaluer la fonction et ses dérivées. Dans la plupart des cas les langage source et cible sont les mêmes. Grâce à cette démarche, l'outil de dérivation joue le même rôle qu'un compilateur de langage informatique. En conséquence, il porte souvent le nom de *compilateur pour la Dérivation Automatique*. Le principe qui caractérise cette technique est illustré sur la figure II.6.

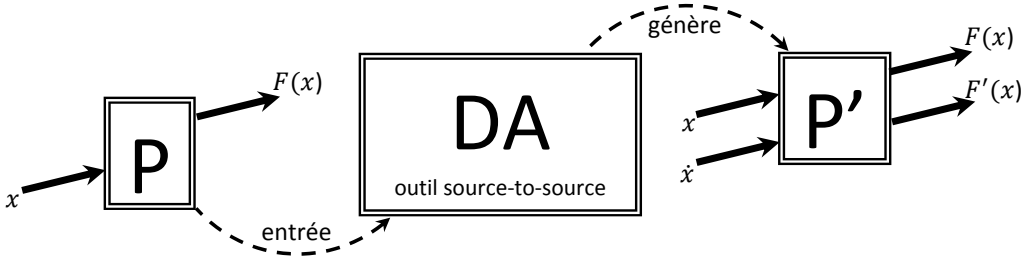


FIGURE II.6 – Le principe de la technique source-to-source

Dans l'Annexe A.1, nous utilisons l'outil TAPENADE [56] pour générer le programme P' qui évalue les dérivées de la fonction de test donnée sur la figure II.2.

En ce qui concerne l'implémentation d'un outil basé sur la transformation du code source, celle-ci représente une tâche très élaborée et elle peut devenir très couteuse en termes d'effort de réalisation. Ceci n'est pas dû forcément aux aspects de dérivation, mais au fait que certains niveaux d'abstraction du langage en question doivent être spécialisés. Cet effort peut être réduit considérablement si certaines fonctionnalités des compilateurs

sont réutilisées. La grande majorité des outils existants, implémentant la technique source-to source définissent leur propre compilateur de langage. Exception de cette affirmation fait par exemple l'outil OpenAD [87] qui utilise un compilateur FORTRAN90 déjà existant appelé Open64 [77]. La figure II.7 donne une représentation générale de l'architecture d'un outil de Dérivation Automatique basé sur la transformation source-to-source.

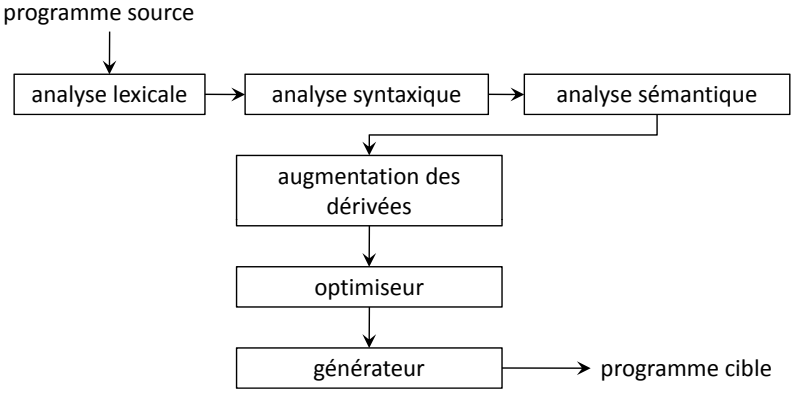


FIGURE II.7 – L'architecture d'un compilateur source-to-source

La plupart des compilateurs de la Dérivation Automatique supportent le langage FORTRAN, qui est plus restrictif et qui offre aux programmateurs une structure d'instructions simple à utiliser. Pour les langages comme C/C++, l'implémentation d'un outil de Dérivation Automatique, basé sur la transformation source-to-source, devient plus complexe, puisque il faut alors gérer les pointeurs, la mémoire dynamique ou le polymorphisme des objets (pour C++ qui est un langage orienté objet). Toutefois, il existe des travaux en cours autour de l'outil ADIC [12]. Notons aussi que l'outil TAPENADE, a été enrichi et peut depuis peu dériver des programmes écrits dans le langage source C (voir [69]).

Les outils implémentant la transformation source-to-source ont l'avantage d'être capables d'analyser et d'optimiser au maximum le programme de calcul de dérivées. Ces optimisations portent notamment sur l'analyse d'activité des variables, le *checkpointing*, ou la gestion des matrices creuses. L'analyse d'activité identifie les variables intermédiaires qui n'interviennent pas dans le calcul des sorties. La dérivation de ces variables est donc complètement ignorée, donnant ainsi plus d'efficacité au calcul des gradients. Le checkpointing est très efficace dans les outils implémentant la transformation source-to-source. D'ailleurs, un atout important de cette technique est qu'elle rend possible l'implémentation très efficace du mode inverse.

La technique source-to-source de Dérivation Automatique de programmes est implémentée dans les outils (cf. à la source [8]) TAPENADE [55][56] et TAF pour la dérivation des programmes FORTRAN95, TAMC [47] et ADIFOR [13] pour des programmes FORTRAN77, ADiMat [11] pour des programmes Matlab, OpenAD [87] pour des programmes C/C++, FORTRAN77, FORTRAN95 etc.

II.2.3.2 La technique de surcharge d'opérateurs

Certains outils de Dérivation Automatique s'appuient sur la surcharge d'opérateurs disponible dans certains langages de programmation d'haut niveau intégrant les concepts de programmation orienté-objet, e.g. C++, Ada, FORTRAN90, Python, etc. Un outil de Dérivation Automatique basé sur la surcharge d'opérateurs prend dans la plupart des cas la forme d'une librairie, essentiellement écrite dans le même langage que le langage de dérivation supporté par l'outil lui-même. La taille de cette librairie est considérablement plus petite que la taille d'un outil source-to-source. De plus, l'effort d'implémentation d'un tel outil est aussi largement réduit. Pour dériver une fonction décrite au moyen d'un programme informatique avec un outil basé sur la surcharge des opérateurs, l'utilisateur est amené à effectuer certaines modifications mineures dans le programme source et à le recompiler ensuite. Ceci est contraire au cas d'utilisation d'un outil source-to-source où le programme initial reste inchangé. Nous appelons cette phase *l'instrumentation* avec un outil de Dérivation Automatique. La figure II.8 illustre le principe d'utilisation d'un outil de Dérivation Automatique reposant sur la surcharge d'opérateurs.

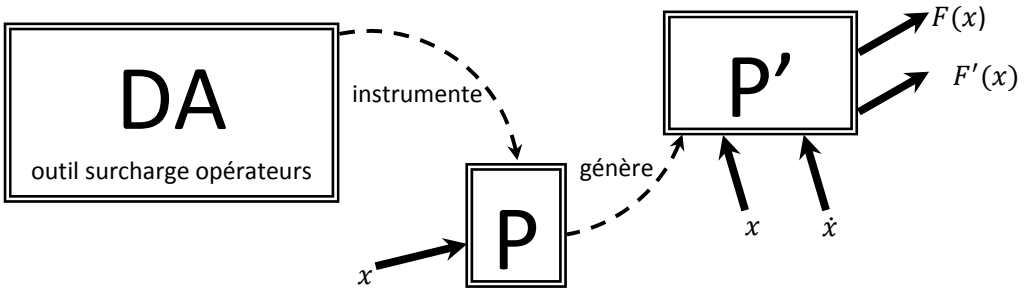


FIGURE II.8 – Le principe de la technique de Dérivation Automatique basée sur la surcharge d'opérateurs

La surcharge d'opérateurs permet de définir un nouveau sens pour les opérateurs mathématiques de base (`+, -, *, /`, etc.) pour de nouveaux types de données. Contrairement à la technique source-to-source qui insère le code de calcul des dérivées dans le programme cible augmenté (voir le programme généré par TAPENADE en Annexe A.1), la technique de surcharge des opérateurs cache cette information dans les objets pour lesquels les opérateurs sont redéfinis. En effet, les outils de Dérivation Automatique s'appuyant sur la surcharge d'opérateurs redéfinissent notamment toutes les fonctions mathématiques de base φ_i de la relation II.4 en supposant que les opérandes v_j et v_i sont d'un type de données autre que celles réelles primitives (`double, double precision, real ou float`). Ces opérandes sont des objets spécialement définis par l'outil de Dérivation Automatique, instances d'une classe qui peut se formuler comme sur la figure II.9.

En utilisant cette technique, les informations concernant les opérations du programme de calcul sont accumulées dynamiquement à l'exécution du programme (dans les variables membres de la classe `ADType` en figure II.9) et non pas statiquement comme le réalise un outil source-to-source. Ainsi, les valeurs des dérivées se propagent simultanément avec

les valeurs de la fonction. Pour cela, la technique de surcharge d'opérateurs implémente efficacement le mode direct de dérivation étudié dans le paragraphe II.2.2.1.

```
class ADType{
   protected:
   double value;
   double dvalue;
   ...
   public:
   ...
   ADType operator * (ADType &arg){
       ADType result;
       result.value  = this->value  * arg.value;
       result.dvalue = this->dvalue * arg.value +
                       this->value  * arg.dvalue;
       return result;
   }
   ... //autres opérations
};
```

FIGURE II.9 – Classe surchargeant l'opérateur * pour la Dérivation Automatique

Du fait de la simplicité de mise en œuvre de la Dérivation Automatique basée sur la surcharge des opérateurs, les outils concernés peuvent implémenter des fonctionnalités mathématiques additionnelles autres que celle de dérivation de premier ordre. Les plus importantes sont notamment les dérivées d'ordre supérieur et le calcul des coefficients de Taylor. Toutefois, ce type d'outils perde beaucoup en performances en termes d'optimisation du programme de calcul des dérivées par rapport à son concurrent implémentant la technique source-to-source. L'analyse d'activité des variables, par exemple, n'est pas possible pour un outil basé sur la surcharge des opérateurs, puisque les opérateurs eux-mêmes ne peuvent pas offrir d'informations additionnelles concernant les dépendances entre certaines variables.

Nous citons les outils de Dérivation Automatique de programmes s'appuyant sur la technique de surcharge d'opérateurs suivants : ADOL-C [90][51], FAD [5] pour des programmes C/C++, ADOL-F [79] pour des programmes FORTRAN95 [79], ADMAT [2] pour des programmes Matlab, etc.

II.2.3.3 Dérivation Automatique en utilisant une trace de calcul

En utilisant la logique de surcharge d'opérateurs, les outils ne sont pas très efficaces à l'implémentation du mode inverse de dérivation. Dans le paragraphe II.2.2.2, nous avons montré que ce mode de dérivation doit utiliser toutes les valeurs des variables intermédiaires participant au calcul des sorties en exécutant au préalable le calcul de la fonction. Les outils de dérivation implémentant la technique de surcharge d'opérateurs perdent normalement cette information après le retour de la fonction. Afin de sauvegarder ces informations en mémoire, les outils basés sur la technique de surcharge d'opérateurs utilisent une *trace de*

calcul[12]. Ces informations sont écrites de diverses façons selon l'outil l'implémentant. En s'appuyant sur la figure II.8, la trace de calcul correspond au programme P' et elle est générée par le programme initial instrumenté. Dans la plupart des cas, dans une trace de calcul sont stockées les identificateurs des opérations (par exemple `MULT` sur la figure II.10) et les opérandes associés (`ID`), plutôt que les valeurs intermédiaires. La figure II.10 illustre la structure d'un objet de la Dérivation Automatique surchargeant les opérateurs, étendue pour la gestion d'une trace de calcul.

```
class ADType{
   protected:
   double value;
   double ID;
   ...
   public:
   ...
   ADType operator * (ADType &arg){
       ADType result;
       tapeOperation(MULT, result, this->ID, arg.ID);
       result.value  = this->value  * arg.value;
       return result;
   }
   ... //autres opérations
};
```

FIGURE II.10 – Classe surchargeant les opérateurs dans une trace de calcul pour la Dérivation Automatique

L'avantage principal de l'utilisation d'une trace de calcul est que toute l'information d'un programme informatique est réduite au niveau des opérations arithmétiques élémentaires. Ceci-dit, l'information complexe concernant le polymorphisme des objets, les pointeurs, la mémoire dynamique est complètement ignorée, sachant que pour un outil basé sur la transformation source-to-source ceci représente une difficulté énorme. Dans l'optimisation sous contraintes, les informations stockées dans la trace de calcul, peuvent être réutilisées à chaque itération pour des nouveaux ensembles de paramètres de dimensionnement sans reéxécuter la fonction originale, ce qui implique une économie de temps d'exécution qui peut devenir très significative avec le nombre des opérations.

Les inconvénients qui peuvent apparaître lors de l'utilisation d'une trace sont dus au fait que les informations concernant les branches conditionnelles qui ne s'exécutent pas au moment de la création de la trace, ne peuvent être utilisées ultérieurement. Nous proposons de nous appuyer sur la fonction de test, afin d'atteindre un niveau de compréhension raisonnable de cet aspect. En s'appuyant sur la forme des opérations mathématiques élémentaires, donnée dans le tableau II.1, supposons qu'on génère une trace de calcul contenant ces informations, pour le même ensemble de valeurs d'entrée que celui du tableau

12. La *trace de calcul* est le terme adopté pour l'emplacement de mémoire tampon utilisé pour sauvegarder des données. Le terme correspondant en anglais est *tape* et ce terme est dérivé de *bandes magnétiques* qui permet, au sens strict, l'accès en lecture/écriture.

II.1. Pour ces valeurs d'entrée, on réutilise la trace pour calculer la dérivée partielle de $\frac{\partial y_2}{\partial x_1}$ en mode direct. La valeur obtenue est juste, et elle est donnée sur le tableau II.2. Supposons qu'on désire la valeur de la même dérivée partielle pour un autre ensemble de données, $x_1 = 1, x_2 = 1.5$ en s'appuyant sur la même trace de calcul. Grâce à un calcul simple, en introduisant dans le tableau II.2 le nouvel ensemble de valeurs, on constate que la valeur correspondante n'est plus correcte. Ceci est dû au fait que pour $x_1 = 1$ c'est la branche `else` qui doit s'exécuter au lieu de `if`. La trace de calcul ne peut donc pas détecter ceci automatiquement puisque toutes les branches conditionnelles sont déroulées lors de sa création. C'est ce qu'on appelle *changement de branche* et une solution très facile pour l'éviter, est de régénérer la trace de calcul pour chaque nouvel ensemble d'entrées. La plupart des outils de Dérivation Automatique implémentent aujourd'hui des mécanismes pour détecter ces changements de branches.

Dans l'Annexe A.2 nous présentons en détail la procédure de création d'une trace de calcul en utilisant l'outil ADOL-C [51][90] de Dérivation Automatique implémentant la technique de surcharge d'opérateurs dans une trace de calcul. Nous y présentons aussi des procédures d'utilisation de la trace en mode direct et inverse en vue de dérivation.

II.3 L'utilisation des fonctions externes implémentée sous ADOL-C

L'origine de cette approche se trouve quelque part dans le passé de l'outil ADOL-C. Elle donnait la possibilité aux utilisateurs de définir leur propre librairie de fonctions externes. Dans la documentation de l'outil [51], cet aspect peut se trouver sous le terme de *quadratures*. L'idée qui se cache derrière une fonction externe est une entité qu'un outil de Dérivation Automatique ne dérive pas suivant ses propres règles. C'est la responsabilité et en même temps la nécessité de l'utilisateur de formuler et de fournir le calcul de ses dérivées. A l'époque, l'utilisateur avait la possibilité de définir ce genre des fonctions avec des restrictions majeures. Cette approche fonctionnait seulement pour des fonctions scalaires à un seul argument ($f_{ext} : \mathbb{R} \to \mathbb{R}$). De plus, le principe permettait seulement l'utilisation des opérations et fonctions mathématiques de base. Ces limitations sont désormais éliminées grâce à un concept plus robuste, implémenté en ADOL-C depuis 2007, permettant d'appeler des fonctions externes $f_{ext} : \mathbb{R}^n \to \mathbb{R}^m$ dans un programme source pour la Dérivation Automatique.

Supposons que la fonction $f_{ext} : \mathbb{R}^n \to \mathbb{R}^m$ est appelée dans le programme II.1 au niveau de l'instruction i. Les gradients de cette fonction sont définis par l'utilisateur dans la fonction f'_{ext}. La façon d'appliquer la règle des fonctions composées est illustrée par l'équation II.11 appliquée au programme P :

$$\begin{cases} P = F = f_p \circ f_{p-1} \circ \cdots \circ f_{ext}(i) \cdots \circ f_1 \\ F'(x) = f'_p(x_p) \cdot f'_{p-1}(x_{p-1}) \cdot \cdots \cdot f'_{ext}(x_i) \cdot \cdots \cdot f'_1(x_1) \end{cases} \tag{II.11}$$

L'équation II.11 montre qu'au niveau de l'appel de la fonction externe, l'outil de Dérivation Automatique transfère le contrôle des dérivées du programme à la fonction externe. Ainsi, la fonction f_{ext} n'est pas dérivée par l'outil, mais par une routine externe que nous appelons f'_{ext}.

La propagation de dérivées est différente selon le mode de dérivation, pour une trace appelant la fonction externe f_{ext}. Ainsi, pour le mode direct, la loi de propagation est donnée sur II.12

$$\begin{cases} v_i = f_{ext}(v_j)_{j \prec i} \quad i > 0, j < i \\ \dot{v_i} = \sum_{j \prec i} f'_{ext}(u_i)_{v_j} \cdot \dot{v_j} \end{cases} \tag{II.12}$$

et pour le mode inverse sur II.13

$$\bar{v_j} = \sum_{i \succ j} \bar{v_i} \cdot f'_{ext}(u_i)_{v_j} \tag{II.13}$$

II.3.1 L'intérêt des fonctions dérivées en externe

L'intérêt le plus évident d'utilisation d'une fonction dérivée en externe est au cas où le code de la fonction en question n'est pas disponible ou il est impossible d'être explicité. Un intérêt plus important est le cas d'utilisation d'une fonction sophistiquée, ou coûteuse en temps d'exécution, pour laquelle le code de calcul des dérivées a besoin d'être optimisé au maximum. Pour cela, il est recommandé de combiner le calcul formel avec la dérivation à la main (où c'est le cas) ou d'utiliser un outil de dérivation automatique (autre qu'ADOL-C) plus performant, spécialisé pour un problème particulier. De plus, certaines parties sensibles du code peuvent éventuellement être écrites dans un langage bas-niveau comme l'assembleur.

Ce concept a été trouvé très utile dans les développements effectués au cours de ces travaux de thèse, notamment pour les aspects concernant la Dérivation Automatique des modèles de dimensionnement semi-analytiques (voir paragraphe IV.5.4), ainsi que pour la plupart des méthodes numériques proposées au chapitre V. Plus de détails sur l'implémentation de ce concept sous ADOL-C, ainsi qu'un exemple, peuvent être trouvées dans l'Annexe A.3

II.3.2 Dérivation avec ADOL-C des programmes appelant des fonctions dérivées en externe

Au cours de nos travaux, ADOL-C implémente seulement la dérivation de premier ordre d'un programme appelant une fonction dérivée en externe. Les deux modes de dérivation, inverse et direct sont applicables dans ce cas. Les pilotes suivants existants en ADOL-C, permettent de réutiliser une trace de calcul des dérivées appelant une fonction dérivée en externe :

TABLE II.4 – Les pilotes ADOL-C permettant la réutilisation d'une trace de calcul appelant une fonction dérivée en externe

Pilot	Mode de dérivation
`zos_forward`	Dérivation d'ordre 0
`fos_forward`	Mode direct scalaire de premier ordre
`fov_forward`	Mode direct vectoriel de premier ordre
`fos_reverse`	Mode inverse scalaire de premier ordre
`fov_reverse`	Mode inverse vectoriel de premier ordre

Pour la dérivation d'ordre supérieur à venir, on s'attend à une stratégie imposant de fournir les dérivées partielles d'ordre supérieur de la fonction, ou les coefficients de Taylor.

II.4 Applications de la Dérivation Automatique en Génie Electrique

En général, les secteurs d'application de la Dérivation Automatique peuvent se classifier en quatre parties. La plus dominante et aussi la plus générique est le domaine des méthodes numériques. Un grand nombre de chercheurs ont montré leur contribution à l'application de la Dérivation Automatique pour certains problèmes de calcul numérique. Les autres secteurs sont représentés par l'analyse de sensibilités, l'optimisation et dernièrement la simulation et les problèmes inverses.

Nous discutons dans cette partie, une série d'applications de la Dérivation Automatique dans le domaine du Génie Électrique. Nous insistons sur la classification de ces applications selon les critères spécifiés auparavant.

Les domaines du Génie Electrique, où la Dérivation Automatique a été utilisée, sont eux aussi diversifiés. Nous constatons qu'il existe un grand nombre de travaux effectués pour la simulation, l'optimisation et le calcul des systèmes et réseaux électriques. L'intérêt principal dans ce domaine est lié aux performances de la Dérivation Automatique dans la gestion des matrices jacobienne creuses, sachant que pour les systèmes ou les réseaux électriques de grande taille, la modélisation est basée sur le calcul matriciel dont la taille est donnée par le nombre de nœuds où de lignes électriques. Dans la plupart des cas, les matrices jacobiennes sont obtenues, soit en utilisant des outils de dérivation basés sur le calcul formel, soit à la main. Certains publications, rappelées dans les paragraphes suivants, montrent que la Dérivation Automatique représente un bon compromis entre l'efficacité en rapidité pour le calcul de gradients et la gestion de matrices creuses afin d'être plus efficaces face aux méthodes traditionnelles de dérivation.

La Dérivation Automatique a été aussi appliquée, ces derniers années, dans les domaines de l'électromagnétisme, l'électromécanique, l'électronique de puissance (simulation des circuits électriques).

II.4.1 Applications de la Dérivation Automatique des programmes pour des méthodes numériques

En [67], les auteurs proposent l'utilisation de la Dérivation Automatique pour le calcul du régime permanent et du *load flow* appliqué aux réseaux électriques. L'outil ADIFOR[13] est employé en mode direct pour dériver des programmes FORTRAN décrivant le système d'équations implicites modélisant le réseau. L'objectif ici est de calculer la matrice jacobienne nécessaire pour la résolution implicite avec la méthode de Newton-Raphson. Cette approche est intégrée dans une structure logicielle capable de modéliser et résoudre les réseaux électriques.

Les auteurs remarquent la simplicité d'utilisation d'un outil de Dérivation Automatique basé sur la transformation source-to-source. Grâce à celui-ci, un utilisateur a possibilité d'introduire des nouveaux modèles d'éléments de circuit pour un problème particulier sans être concerné par la dérivation de ces derniers. Il est mentionné également que l'efficacité de l'outil de Dérivation Automatique peut augmenter le temps de résolution du *load flow* considérablement si la matrice jacobienne creuse, générée par ADIFOR est utilisée.

II.4.2 Applications pour la simulation

Le papier [59], présente une application de la Différentiation Automatique pour la simulation de la dynamique transitoire et à long terme des systèmes électriques. En présence des régulateurs de vitesse et de tension, le système électrique considéré se modélise par un ensemble d'équations différentielles algébriques où les variables d'état sont notamment les courants des machines et les tensions nodales. ADIFOR[13] est utilisé en tant qu'outil de Dérivation Automatique fournissant les dérivées partielles nécessaires pour une version particulière de l'algorithme Newton-Raphson appliqué au système algébrique implicite résolu en parallèle avec les équations différentielles. Les auteurs estiment un gain en efficacité de temps d'exécution jusqu'à 80% par rapport à la dérivation basée sur la méthode des différences finies. Ceci est dû à la capacité de l'outil de dérivation de gérer des matrices creuses.

Dans l'article [60], les auteurs proposent d'utiliser la Dérivation Automatique dans la simulation des circuits pour des applications industrielles. Ils sont intéressés par la dérivation des modèles de transistors MOS décrits dans le langage moderne de modélisation des circuits, VHDL-AMS, qui est de plus en plus utilisé par les communautés d'électroniciens de puissance. Le code FORTRAN généré par cet outil est dérivé par trois outils de Dérivation Automatique source-to-source. Les résultats sont comparés avec le calcul formel d'une version particulière de Maple V [61]. Enfin, toutes ces approches sont comparées avec la dérivation à la main, que nous considérons la plus performante. Il ressort que la Dérivation Automatique est entre 5% et 80 % plus lente que la dérivation à la main. Le calcul formel l'est entre 120% et 180% et il rencontre des problèmes majeures de mémoire pour des modèles de grande taille (à plus de 350 formules).

La Dérivation Automatique intégrée avec le compilateur VHDL-AMS réalise une éco-

nomie de temps de conception des modèles de transistors puisque les utilisateurs électroniciens ne sont plus concernés par la dérivation de ces derniers. Nous reportons une étude similaire dans l'article [36] où un outil de la Dérivation Automatique est utilisé pour modéliser des circuits via Modelica.

II.4.3 Applications pour l'analyse de sensibilités

L'article [58] utilise ADIFOR [13] en tant qu'outil de Dérivation Automatique pour faire une analyse de sensibilités dans les réseaux électriques. Le but est d'estimer l'effet des paramètres sur certains éléments du réseau comme les nœuds, les générateurs ou des lignes. De cette façon les auteurs proposent une solution permettant de trouver les nœuds faibles pour les limites de tensions admissibles, les générateurs faibles pour les limites de puissance réactive générée et les lignes faibles pour la stabilité de tension. Les auteurs illustrent, basées sur des exemples, que les performances en temps d'exécution de la Dérivation Automatique deviennent supérieures au calcul formel pour des problèmes de grande taille.

Un projet de grande taille est porté à l'étude dans les articles [14][15]. Une analyse de sensibilités est effectuée sur le potentiel électrostatique calculé dans un domaine reparti sur plusieurs régions à perméabilités diélectriques différentes. Le potentiel est obtenu en résolvant les équations aux dérivées partielles de Laplace. Dans cette étude, les auteurs utilisent l'outil de Dérivation Automatique, ADIFOR, pour dériver le code FORTRAN implémentant la méthode des éléments finis dans un logiciel de simulation appelé SEPRAN. L'ampleur de ce projet réside dans le fait qu'un programme ayant une complexité très élevée (approximatif 400 000 lignes de code) est dérivé sans problèmes majeures, selon les auteurs.

Pour avoir une idée des résultats obtenus, les auteurs montrent que le potentiel électrostatique est plus sensible aux perméabilités diélectriques dans les zones de séparation des différentes régions. Le programme généré par ADIFOR est jusqu'à un ratio de 2.36 plus lent que le programme initial, et est de 2.88 plus coûteux en mémoire. Ceci est acceptable puisque le programme dérivé ne calcule pas seulement la fonction originale, mais aussi certaines dérivées partielles.

En électromagnétisme, une analyse de sensibilités du champ électrique produit par un conducteur au milieu d'un domaine 2D par rapport aux paramètres géométriques est porté à l'étude dans l'article [16]. Un programme écrit sous Matlab intégrant les équations de Maxwell par la méthode des différences finies dans le domaine du temps (FDTD) est dérivé automatiquement avec l'outil source-to-source ADiMat [11]. Une comparaison avec le calcul de mêmes dérivées avec la méthode de différences finies est évoquée.

Dans l'article [36], les auteurs utilisent la Dérivation Automatique pour évaluer les sensibilités des variables d'état, décrites par systèmes d'équations différentielles algébriques, par rapport à certains paramètres de contrôle. Le contexte d'application est lié à la modélisation des circuits électriques implémentée sous Modelica. L'outil de Dérivation Automatique s'appelle ADModelica [35] et il est basé sur la technique source-to-source pour

dériver des modèles ou des programmes décrits dans le langage Modelica. Il est peut être le premier outil de Dérivation Automatique qui dérive des programmes décrits dans des langages spécialisés pour la modélisation d'une physique particulière et non pas un langage de programmation traditionnel comme C/C++ ou Fortran.

II.4.4 Applications pour l'optimisation

Dans l'article [68], les auteurs présentent un problème d'optimisation pour identifier les positionnements optimaux d'un dispositif FACTS-série dans un réseau, en régime permanent. Le but est d'augmenter au maximum la capacité de transfert de puissance active entre le réseau considéré et le réseau d'interconnexion. Le problème d'optimisation est formulé par l'équation II.14 :

$$\begin{cases} \min F(x) \\ g(x) = 0 \\ l \leq x \leq u \end{cases} \tag{II.14}$$

où les paramètres du modèle sont x, F représente la fonction objectif, g est une fonction des contraintes donnée sous une forme implicite. Elle sert à calculer les équations implicites du régime permanent et de composants du réseau.

Les auteurs utilisent l'outil source-to-source ADIFOR [13] pour générer le programme FORTRAN77 de calcul des dérivées de la fonction objectif et de toutes les contraintes par rapport aux paramètres. Ces dérivées sont utilisées par un programme d'optimisation. En complément, une analyse de sensibilités est réalisée pour la fonction objectif sachant que sur chaque ligne de transport il est installé un dispositif FACTS inactif. Les auteurs estiment que l'emplacement optimal est représenté par la ligne électrique donnant la sensibilité la plus importante de la fonction objectif par rapport à chaque réactance ($\left|\frac{\partial P_{neoud_{bilan}}}{\partial X_{FACTS_{Lignei}}}\right|$).

Les résultats d'optimisation obtenus pour ce problème sont trouvés satisfaisants. Les auteurs qualifient, entre autre, la combinaison d'un outil de Dérivation Automatique avec un outil particulier d'optimisation, comme une approche efficace et flexible pour la formulation des problèmes d'optimisation dans le domaine des réseaux et systèmes électriques. D'autres détails, intéressants pour nos travaux, par exemple les performances de l'outil de Dérivation Automatique utilisé n'y sont pas abordées.

Une étude plus récente, qui date de 2008, est publiée dans l'article [96]. Elle concerne également l'optimisation dans les réseaux et les systèmes électriques. Cette fois-ci, il s'agit d'un projet industriel au sein de la compagnie française RTE, responsable avec les services d'opération, maintenance et développement du réseau électrique de transport d'électricité. Le but est d'obtenir un état sécurisé optimal du réseau, formulé comme une somme quadratique correspondante à un planning de génération ou à un profil de tensions. En même temps, il existe des contraintes concernant les limites de transfert de puissance sur les lignes électriques (congestions) et les niveaux de tensions nodales admissibles. Le problème d'optimisation est résolu en deux phases. Dans la première phase, un simulateur utilisé au

milieu industriel, linéarise le problème d'optimisation pour faciliter la convergence globale. Un aspect intéressant est lié à la linéarisation, qui implique le calcul des dérivées de premier et de deuxième ordre. Les deux modes de Dérivation Automatique implémentés sous ADOL-C [51], sont combinés afin d'évaluer ces dérivées. Cette fois-ci aussi, les matrices jacobienne et hessienne creuses sont exploitées pour éliminer une grande partie des opérations inutiles et donc augmenter les performances de l'exécution. Les auteurs quantifient le temps d'exécution par 77% dédié au calcul des matrices hessiennes, 14% dédié à la gestion des matrices creuses, 6% dédié au calcul des dérivées de premier ordre et seulement 3% pour le calcul effectif des fonctions. Ces performances sont obtenues pour un réseau d'une centaine de milliers de nœuds.

Un des premiers dimensionnements de dispositifs du Génie Electrique par optimisation basé sur le calcul des gradients en appliquant la Dérivation Automatique a été réalisé par Vincent Fisher en [42], lors de ses travaux de thèse au Laboratoire de Génie Electrique de Grenoble. Il utilise l'outil ADOL-C pour évaluer les gradients des modèles constitués par des équations explicites. Il a appliqué ceci à des modèles électromagnétiques analytiques décrits sous la forme de réseaux de réluctances. L'approche utilisée est d'une importance majeure pour nos travaux puisque nous intervenons dans la continuité des développements de cette thèse en allant plus loin que les équations analytiques simples pour décrire des modèles de dimensionnement. Nous visons notamment les aspects vectoriels et ceux numériques (algorithmiques).

II.5 Conclusion

Nous avons présenté dans la première partie de ce chapitre, les fondements mathématiques et les stratégies d'implémentation de la Dérivation Automatique des programmes. Différentes stratégies d'implémentation sont possibles : outils source-to-source ou outils basés sur la surcharge d'opérateurs. Nous notons que les outils implémentant la technique source-to-source, malgré le fait qu'elles soient plus complexes à se mettre en œuvre, sont plus simples à s'utiliser. Les outils implémentant la technique source-to-source ont l'avantage de générer les programmes de calcul des dérivées optimisés au maximum. L'avantage des outils basés sur la technique de surcharge des opérateurs, est d'utiliser les options d'optimisation implémentées dans les compilateurs. Ces derniers deviennent par ailleurs réellement efficaces lorsqu'ils implémentent la stratégie dite trace de calcul.

De manière générale, il est bien de savoir que le mode inverse est plus efficace pour dériver une fonction avec un nombre des variables de sortie supérieur au nombre d'entrées et le mode direct au cas contraire [52]. De plus, la dérivation en mode inverse des algorithmes itératifs engendre beaucoup de pertes en efficacité, puisque l'outil de dérivation est conduit à stoker un grand nombre d'informations concernant le graphe itératif de la fonction.

Dans la deuxième partie de ce chapitre nous faisons l'état de l'art de l'utilisation de la Dérivation Automatique dans le domaine du Génie Electrique. Cette technique connait cependant un manque important d'utilisation et de notoriété dans un domaine où le calcul

des gradients est omniprésent. Nous signalons un nombre important de travaux dans le domaine des réseaux et systèmes électriques ; mais là, encore dans la plupart des publications, la Dérivation Automatique fait l'objet de comparaisons des performances en grande partie avec la dérivation à la main et n'apparaît donc pas indispensable. Peu d'applications, notifiées dans ce chapitre, font appel à la Dérivation Automatique pour simplifier la tâche de dérivation [16][60][36][14][15].

En ce qui concerne le dimensionnement des dispositifs électromagnétiques, qui fait l'objet de nos travaux de thèse, nous constatons une absence totale de la Dérivation Automatique, sauf dans les travaux de Vincent Fischer que nous poursuivons. D'autres alternatives sont souvent préférées au dimensionnement par optimisation avec des algorithmes basés sur le calcul de gradients. Ceci est peut être dû au fait que le calcul de champ se fait avec une précision élevée par des méthodes numériques complexes, comme celle des éléments finis intégrant les équations à dérivées partielles de Maxwell. Cette méthode implique l'existence d'une fonction objectif bruitée, ayant une variation en fonction de paramètres de conception à beaucoup des minimums locaux. Par ailleurs, d'autres approches d'optimisation ont été approfondies dans la communauté du calcul de champ magnétique, étant donné que les algorithmes d'optimisation basés sur les gradients ont une convergence risquée vers des extremums locaux. Il s'agit notamment des algorithmes stochastiques ou des méthodes d'approximation analytique comme les surfaces de réponses ou le space-mapping.

Finalement, nous rappelons au lecteur que l'utilisation de la Dérivation Automatique dans notre travail est un besoin fort pour le dimensionnement de dispositifs comme nous allons l'illustrer dans les paragraphes qui suivent.

Chapitre III

L'apport de la Dérivation Automatique à l'évolution d'un langage de modélisation pour l'optimisation

Many of the difficulties of translation from modeler's form to algorithm's form can be circumvented by the use of a computer modeling language *for mathematical programming. A modeling language is designed to express the modeler's form in a way that can serve as direct input to a computer system.*

R. Fourer - 1990

Sommaire

Résumé

Dans ce chapitre, nous illustrons les spécificités qu'un langage de modélisation doive implémenter afin de permettre au concepteur de décrire d'une façon logique, simple et intuitive ses modèles de dimensionnement pour l'optimisation. Parmi ces aspects on retrouve fondamentalement le besoin des formules analytiques explicites scalaires. Par ailleurs, nous évoquons aussi le besoin de variables vectorielles et d'algorithmes numériques complexes contenant de boucles répétitives, instructions conditionnelles, etc.

Le support de travail de ce chapitre est représenté par le langage de modélisation sml*, qui repose exclusivement sur des formules analytiques explicites scalaires, dérivées par la technique de calcul formel. Dans sa première partie, ce chapitre illustre l'état initial du langage de modélisation implémenté sous CADES.*

Deux enjeux majeurs sont à l'étude dans ce chapitre. Premièrement, nous souhaitons faire évoluer le langage de modélisation, notamment en introduisant la notion de vectorisation. Deuxièmement, nous souhaitons offrir la possibilité de pouvoir définir des modèles de dimensionnement contenant des instructions complexes et d'évaluer les gradients correspondants de ces derniers avec un effort minimal de la part du concepteur. Pour ces deux aspects, nous avons l'intention d'utiliser la Dérivation Automatique pour différentier les modèles.

III.1 Introduction : la nécessité d'un langage plus élaboré pour décrire des modèles pour l'optimisation

La description des modèles est une des tâches la plus consommatrice de temps de dimensionnement. Cette description se réalise le plus naturellement possible en utilisant des langages de modélisation, qui sont en effet très proches de langages de programmation classiques. L'expérience montre qu'un langage de description de modèles de dimensionnement doit avoir certaines propriétés fondamentales. D'une part, un langage de modélisation doit offrir la possibilité de décrire des modèles analytiques explicites scalaires. D'autre part, il est aussi nécessaire de pouvoir utiliser des vecteurs et de décrire des algorithmes numériques, pour l'implémentation des aspects mathématiques plus complexes comme les implicites, les intégrales et l'intégration des systèmes d'équations différentielles. Ces algorithmes ne sont pas seulement constitués par des formules, mais plutôt par des boucles répétitives, branches conditionnelles ou sauts.

Nous proposons dans ce chapitre de réaliser ces spécifications en s'appuyant sur un langage existant au début de nos travaux, le langage *sml* existant sous CADES, introduit au chapitre I. Ce langage implémentait dès le début de nos travaux le besoin fondamental, c'est-à-dire la modélisation par de formules explicites analytiques scalaires.

III.2 Caractéristiques du langage de modélisation préexistant à nos travaux

Ce paragraphe porte sur la description du langage source de modélisation *sml* qui permet au concepteur, grâce à une syntaxe simple, d'exprimer un modèle mathématique de dimensionnement. La syntaxe de ce langage est basée en grande partie sur la description des modèles analytiques combinée avec des programmes numériques externes. La dérivation est réalisée avec des techniques de calcul formel.

III.2.1 La modélisation analytique

Au début de nos travaux de thèse, la modélisation analytique se réalise dans le langage *sml* en utilisant des équations analytiques explicites et scalaires. Il n'y existait donc pas la possibilité de définir et d'utiliser des variables tableaux ou des équations données sous une forme implicite. La syntaxe d'une équation analytique est donnée dans la figure [1] III.1 :

1. Pour spécifier d'une manière convenable la syntaxe des différentes structure du langage *sml*, nous nous avons inspiré dans nos travaux de la notation utilisée pour décrire les synoptiques des commandes des systèmes d'exploitation. Ainsi, un indicateur (mot ou *token*) non-gras souligné dénote le fait que celui-ci est remplaçable par tout autre indicateur défini par l'utilisateur. Les indicateurs imposées par la syntaxe du langage sont écrits en gras. | dénote l'opérateur `ou` logique. Toute expression suivie par `,...` non-gras dénote une liste ou une répétition.

```
sml:
nom variable = expression mathématique ;
```

FIGURE III.1 – La syntaxe d'une équation analytique écrite dans le langage *sml*

où le membre du gauche est toujours un nom de variable, qui explicite l'équation et le membre du droite est une expression mathématique constituée par :

1. des constantes ou des variables scalaires.
2. des opérateurs mathématiques de base, comme `+, -, *, /`.
3. des fonctions trigonométriques ou mathématiques de base, comme `sin(.), tan(.), exp(.), log(.), pow(.,.)`, etc.

Afin d'améliorer la compréhension, nous proposons l'exemple d'un modèle mathématique purement analytique, qui représente la résistance d'un fil électrique dépendant de la longueur du fil, de sa section et de sa résistivité, ainsi que la loi d'Ohm appliquée pour calculer la chute de tension à ses bornes, supposant le fil parcouru d'un courant électrique.

```
sml:
R0 = rho*L/S;     //résistance du fil électrique;
U = R0*I;         //loi d'Ohm
```

FIGURE III.2 – Modèle analytique écrit dans la syntaxe *sml*, évaluant la résistance d'un fil électrique et la chute de tension à ses bornes en utilisant la loi d'Ohm

Le parseur [2] du compilateur de ce langage vérifie la justesse de ces équations en s'appuyant sur la règle de construction présentée sur la figure III.1. Une fois identifiées, toutes les équations sont analysées par l'analyseur du compilateur dans le but d'identifier toutes les variables, ainsi que leur nature (entrées/sorties). Ainsi, les variables qui apparaissent au moins une fois dans les membres de gauche seront des variables de sortie et autrement, des entrées. Le tableau III.1 illustre l'effet de l'action de l'analyseur sur le modèle de dimensionnement de la résistance du fil électrique :

TABLE III.1 – L'identification des variables d'un modèle mathématique de dimensionnement analytique

Nom variable	Nature
I	Entrée
L	Entrée
rho	Entrée
S	Entrée
R_0	Sortie
U	Sortie

2. Un parseur est une entité d'analyse syntaxique d'un texte, réalisé par une séquence d'indicateurs (des mots), dans le but de déterminer sa structure grammaticale selon un ensemble des règles.

Muni de ces informations, le compilateur est capable de générer le programme cible Java correspondant à ce modèle mathématique. Le programme cible Java généré reste inchangé, puisque ce langage accepte aussi tous les opérateurs et les fonctions mathématiques de base. La seule différence entre les deux langages est le fait que sous *sml*, il n'existe pas de mécanisme permettant de déclarer le type des variables, étant de ce fait plus simple à appréhender. Ainsi, toutes les variables sont par défaut du type `double`. Sous Java toutes les variables présentes dans le modèle de dimensionnement sont déclarées comme des `double`s.

En ce qui concerne la génération du code de calcul des dérivées de modèles de dimensionnement, cela représente une tâche plus élaborée. Ainsi, au début de nos travaux cette tâche était réalisée en utilisant un moteur de calcul formel s'appelant RAMA [3], développé au cours de travaux de thèse de V. Fischer [40] et B. du Peloux [71]. Cet outil transforme toute expression mathématique en un arbre, dont les nœuds peuvent être des opérateurs mathématiques, des fonctions, des variables ou des constantes. Cet arbre est ordonné, les branches dénotant l'appartenance des nœuds enfants aux parents. En appliquant des règles prédéfinies de calcul des dérivées sur cette représentation d'une équation analytique, on peut obtenir les formules symboliques correspondantes. Dans la figure suivante est illustrée l'obtention des formules des dérivées en s'appuyant sur l'équation qui calcule la résistance donnée sur la figure III.2.

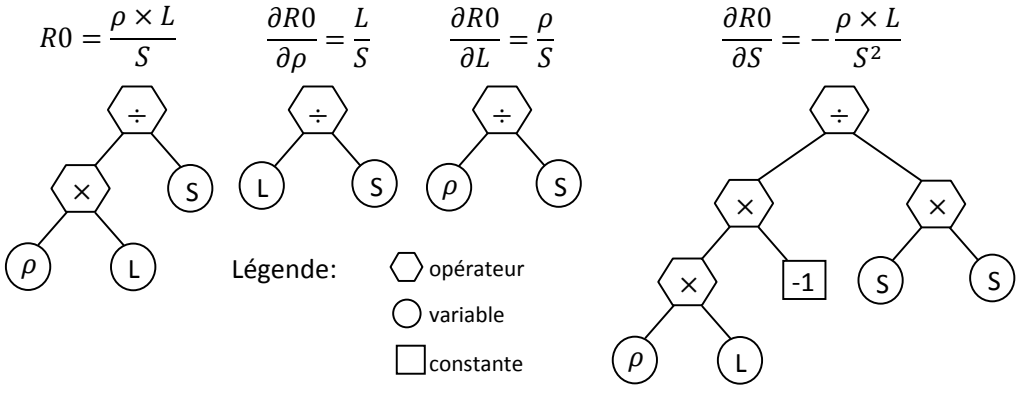


FIGURE III.3 – L'arbre de calcul et du calcul des dérivées généré par l'outil de calcul formel RAMA

III.2.2 Les fonctions internes

La syntaxe du langage *sml* permet aussi au concepteur de définir et d'utiliser des fonctions. Ces fonctions sont appelées *internes* puisqu'elles sont définies à l'intérieur du modèle mathématique de dimensionnement. La syntaxe d'une telle fonction est spécifiée sur la figure III.4.

3. *RAMA* est l'acronyme de *Rule Applicator for Mathematical Analysis* est il représente un outil dédié à l'analyse des expressions mathématiques et au calcul formel des dérivées.

```
sml:
nom ( argument, ...) = expression mathématique ;
```

FIGURE III.4 – La syntaxe d'une fonction interne écrite dans le langage *sml*

Ainsi, les fonctions internes sont constituées d'un prototype contenant un nom, une liste d'arguments et une expression mathématique scalaire. Leur but est d'utiliser à plusieurs reprises une expression mathématique paramétrée dans le modèle de dimensionnement.

Pour une meilleure compréhension, reprenons l'exemple précédent du modèle de dimensionnement dans la figure III.2 et considérons cette fois que la résistance dépend de la température. La loi de cette dépendance peut s'écrire sous la forme d'une fonction interne, donnée sur la figure III.5.

```
sml:
R0 = rho*L/S;                  //résistance du fil à 0°C;
R(x) = R0*(1+A*x+B*pow(x,2)); //résistance du fil à x°C;
U = R(T)*I;                    //loi d'Ohm
```

FIGURE III.5 – Modèle analytique contenant une fonction interne évaluant la variation de la résistance électrique avec la température

La transcription des fonctions internes dans le langage Java est aussi simple que dans le cas des équations analytiques. Leur dérivation n'implique aucune difficulté supplémentaire par rapport à la dérivation des formules analytiques. Le moteur RAMA de calcul formel, réagit dans la même philosophie que celle montrée sur la figure III.3.

III.2.3 Les variables internes

Toutes les variables présentes dans un modèle *sml* sont par défaut des variables de dimensionnement. Ceci signifie que le concepteur a la possibilité dans le processus d'optimisation de définir des contraintes sur ces variables ou de choisir une fonction objectif parmi les sorties.

```
sml:
intern nom variable = constante | expression mathématique ;
```

exemple d'utilisation

```
sml:
intern pi  = 3.14;      //constante;
intern mu0 = 4*pi*1e-7; //variable temporaire;
```

FIGURE III.6 – La syntaxe pour déclarer des variables internes dans le langage *sml*

Il existe cependant des situations où certaines variables ne sont pas dimensionnables, et d'autres où le concepteur désire les cacher complètement du processus d'optimisation.

C'est notamment le cas des constantes mathématiques ou physiques comme π, e, μ_0 des variables intermédiaires qui servent seulement à évaluer des variables de sortie. Le langage *sml* propose une stratégie permettant au concepteur de masquer des variables non-dimensionnables dans le processus de dimensionnement. Ceci peut se réaliser en utilisant le mot clé `intern` comme le montre la figure III.6.

III.2.4 La modélisation semi-analytique en utilisant des fonctions externes

Souvent, dans la modélisation, certaines fonctions ne peuvent pas être représentables sous la forme analytique, mais plutôt par un programme informatique ou par des routines mettant en œuvre des algorithmes numériques. Ces algorithmes peuvent implémenter des méthodes numériques dédiées, comme par exemple l'intégration des fonctions, des algorithmes de résolution des systèmes d'équations implicites non linéaires, etc. Le langage *sml* donne au concepteur la possibilité de combiner la modélisation analytique avec des algorithmes numériques complexes utilisant toutes les structures de contrôle supportées par les langages de programmation existantes, comme des instructions répétitives `for, while`, ou des instructions conditionnelles `if-then-else, switch-case`.

L'analyse de la partie numérique d'un modèle de dimensionnement n'est pas supportée directement par le compilateur du langage *sml*. Cette tâche aurait impliqué l'existence du type entier de données pour les compteurs des boucles répétitives, ainsi qu'une analyse complexe de chaque instruction. Suite aux travaux de thèse de D. Duret [33] et aux notre décrits en [38], la solution proposée et d'utiliser un langage de programmation existant pour définir ces algorithmes numériques et d'adapter le compilateur de langage pour les intégrer sans s'intéresser à leur contenu intérieur. La modélisation semi-analytique repose donc sur l'utilisation des fonctions décrites dans un langage externe au *sml*. Un bon compromis est le langage Java, sachant que toute l'architecture de l'environnement CADES est décrite dans ce langage.

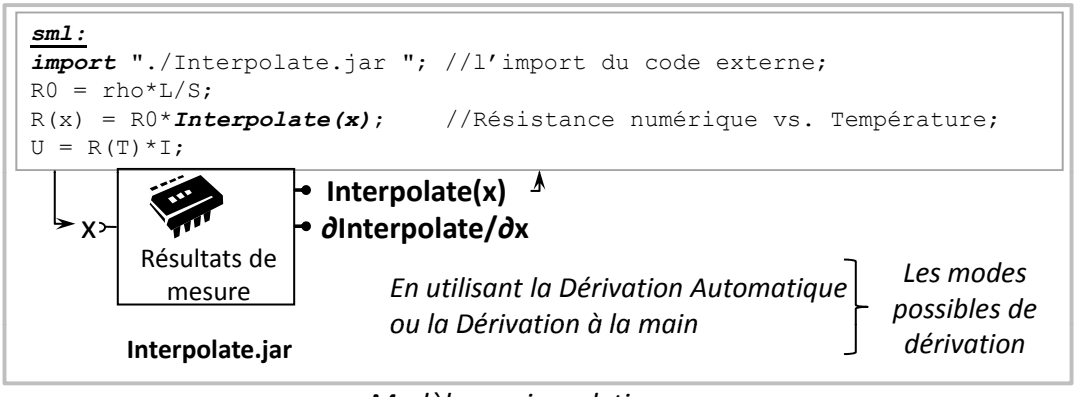


FIGURE III.7 – Modélisation semi-analytique dans le langage *sml* en utilisant des fonctions externes

Pour améliorer la compréhension de ces explications, reprenons l'exemple donné sur la figure III.2 et supposons cette fois-ci que la dépendance de la résistance à la température ne soit plus définie par une fonction analytique, mais à partir d'une base de données obtenue par mesure, interpolable par un code numérique. Cette formulation est donnée sur la figure III.7.

Ici la fonction `Interpolate` s'appelle fonction externe, car elle n'est pas une fonction de base (`sin, exp`...), ni une fonction interne comme celle spécifié au paragraphe III.2.2. Elle est définie et compilée par le concepteur dans une classe Java comme celle illustrée sur la figure III.8.

Le calcul formel des dérivées d'un modèle de dimensionnement semi-analytique implique l'existence des gradients de toutes les fonctions externes utilisées dans le modèle. Cela est la responsabilité du concepteur. Ainsi une fonction définie dans le langage Java et utilisée dans un modèle de dimensionnement, n'est valide que si la fonction correspondante au calcul de son gradient est définie. Pour ceci, le langage *sml* impose le standard de définition d'une fonction externe illustré sur la figure III.8 :

```
Java:
double nom_fonction(double argument,...);
double[] jacobian_nom_fonction(double argument,...);
```

FIGURE III.8 – Le standard de définition d'une fonction externe dans le langage Java, utilisable dans la modélisation semi-analytique sous *sml*

La dérivation d'une fonction externe peut se faire dans certains cas en exploitant les propriétés mathématiques des calculs implémentées dans le code. C'est typiquement ce qui est fait pour la dérivation du programme de résolution de fonctions implicites, où on utilise le théorème des fonctions implicites. Grâce à ce théorème, les dérivées de la solution du système III.1 par rapport aux paramètres peut se formuler comme en III.2.

$$f(X,P)=0, \quad X\in\mathbb{R}^n, \quad P\in\mathbb{R}^p, \quad f:\mathbb{R}^{n\times p}\to\mathbb{R}^m \tag{III.1}$$

$$\frac{\partial X}{\partial P}=-\frac{\frac{\partial f}{\partial P}}{\frac{\partial f}{\partial X}}, \quad \frac{\partial X}{\partial P}\in\mathbb{R}^{n\times m} \tag{III.2}$$

On retrouve ce genre de techniques en au chapitre V. Cependant, ceci reste exceptionnel. En général, il faut procéder soit par la méthode de différences finies avec une précision risquée, soit essayer de dériver directement le code informatique avec la technique de Dérivation Automatique, ce qui n'est pas envisageable pour l'ingénieur électrotechnicien, peu formalisé avec ce genre de technique.

III.2.5 La modélisation semi-analytique en utilisant des fonctions de fonctions externes

Il peut exister des situations où une fonction externe comme celle présentée au paragraphe III.2.4 doit être définie non seulement pour des arguments de type `double` (à voir figure III.8), mais aussi pour des arguments représentant des fonctions internes comme celles définies dans la paragraphe III.2.2. Ceci est le cas d'utilisation d'une *fonction de fonction externe* et ce concept a été développé pendent les travaux de thèse de H. L. Rakotoarison [75] et de nos travaux décrits en [38][39]. Pour une meilleure compréhension, supposons que dans le modèle mathématique défini sur la figure III.5 le rayon du fil électrique varie comme le montre la figure III.9 :

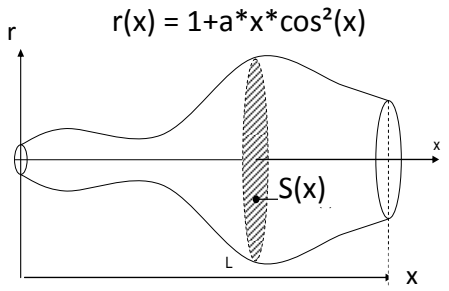


FIGURE III.9 – La variation du rayon du fil électrique considéré

A partir de cette variation, nous proposons de calculer le rayon moyen en utilisant l'intégrale :

$$r_{moy} = \frac{1}{L} \cdot \int_0^L 1 + a \cdot x \cdot \cos^2 x \; dx, \quad a > 0 \tag{III.3}$$

La résistance du fil électrique à $0°C$ change selon l'équation III.4 (section constante du fil conducteur) :

$$R_0 = \rho \cdot \frac{L}{\pi \cdot r_{moy}^2} \tag{III.4}$$

Le modèle mathématique de dimensionnement appelant l'intégrale peut s'écrire comme le montre la figure III.10 :

```
sml:
import "./Integral.jar ";            //l'import de l'intégrale;
r(a,x) = 1+a*x*cos(x)*cos(x);        //variation du rayon (fonction interne);
rmoy = 1/L*integral(r(a,_),0,L);     //utilisation de la fonction de fonction
R0 = rho*L/(Pi*pow(rmoy,2));         //calcul de la résistance à 0°C
R(x) = R0*(1+A*x+B*pow(x,2));//variation de la résistance avec la température
U = R(T)*I;                          //loi d'Ohm
```

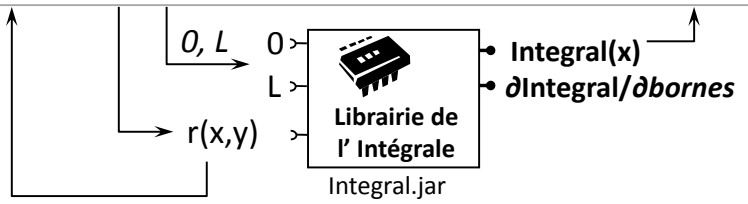
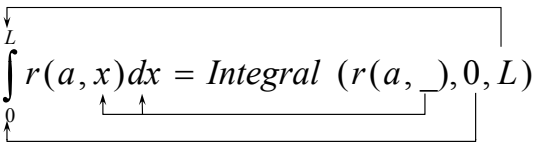
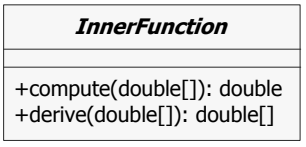


FIGURE III.10 – Modélisation semi-analytique dans le langage *sml* en utilisant des fonctions de fonctions externes

La syntaxe proposée dans le langage *sml* pour l'appel de la fonction de fonction externe qui approxime l'intégrale dans l'équation III.3 est donnée dans la figure III.11a



a.



b.

FIGURE III.11 – Le formalisme de fonctions de fonctions ; a. Passage des arguments ; b. La fonction argument d'une fonction de fonctions externe

Cette syntaxe est formalisée, c'est-à-dire qu'elle est applicable pour toute fonction de fonctions hétérogène définies par le concepteur, autres que l'intégrale. Les significations des éléments présents dans cette syntaxe sont les suivantes :

1. *Integral* - le nom de la fonction de fonctions. Cette fonction est définie par le concepteur dans le langage Java. Le standard de définition d'une fonction de fonction est donné sur la figure III.12 :

```
Java:
double nom_fonction(InnerFunction argument,... double argument,...);
double[] jacobian_nom_fonction(InnerFunction argument,... double argument,...);
```

FIGURE III.12 – Le standard de définition d'une fonction de fonction externe dans le langage Java, utilisable dans la modélisation semi-analytique sous *sml*

2. *r* - la fonction argument qui représente une fonction interne définie dans le modèle de dimensionnent. Dans le cas de l'intégrale, cette fonction représente l'intégrande. Le service de génération du code existant au niveau du compilateur du langage *sml*, génère automatiquement une instance de la classe spécifiée sur la figure III.11b. Les

méthodes Java comme `compute(), derive()` permettent de calculer la valeur et le gradient de la fonction argument dans un point. Par exemple, les équations III.5 et III.6 évaluent la valeur de l'intégrande dans la borne inférieure et respectivement dans celle supérieure. A noter que ces valeurs sont indispensables, dans la plupart des cas, pour l'algorithme implémentant la méthode numérique.

$$\begin{aligned} r(a,0) &= rObject.compute(\{0\}) \\ r(a,L) &= rObject.compute(\{L\}) \end{aligned} \tag{III.5}$$

$$dr(a,0) = \left\{ \frac{\partial r}{\partial arg_1}(a,0), \frac{\partial r}{\partial arg_2}(a,0) \right\} = rObject.derive(\{0\}) \tag{III.6}$$

3. _ - est un mot clé utilisé pour notifier au compilateur la position de l'argument à passer à la fonction interne lors de l'appel de la fonction `compute()` ou `derive()`. Dans le cas de l'intégrale ce mot clé désigne la variable d'intégration.

De même que pour les fonctions externes *classiques* le gradient des fonctions de fonctions externes doit être implémenté par le concepteur.

III.3 Spécifications pour améliorer la modélisation initiale

Dans ce paragraphe, nous discutons les principales limitations du langage de modélisation *sml* qui nous ont conduit à faire évoluer sa *puissance d'expressivité* (trouvée aussi dans littérature des langages de programmation sous le nom de *puissance de computabilité/calculabilité ("computability")* [80][18]). Ces limitations sont évidentes au premier regard, vu que les modèles de dimensionnement pour l'optimisation peuvent être construits seulement en utilisant des formules analytiques explicites scalaires combinées éventuellement avec des algorithmes numériques dérivés en externe. Les principales difficultés de cette logique de modélisation sont liées au manque des vecteurs et au fait que le concepteur doit définir d'une manière ou d'une autre les gradients de ses fonctions externes. Notre but principal est d'éliminer totalement ou partiellement ces difficultés de modélisation (*purement analytique scalaire* ou *semi-analytique*) en faisant appel à la Dérivation Automatique.

III.3.1 Spécifications pour la modélisation vectorielle

Dans un premier temps, nous allons aborder l'inconvénient de la modélisation *purement analytique* et l'impact que les variables vectorielles pourraient l'avoir. Reprenons l'exemple de la figure III.7 de l'appel d'une fonction externe et supposons cette fois-ci que la fonction `Interpolate` doit retourner non seulement une valeur mais plusieurs. Ceci est d'un intérêt majeur pour la résolution des systèmes d'équations implicites, traités dans le chapitre V, où la valeur de retour représente la solution estimée avec un algorithme de type Newton implémenté dans une fonction externe. Vu que le langage de modélisation est scalaire,

afin de récupérer la solution entière de notre système, il est nécessaire de faire appel à cette fonction d'un nombre de fois égal à la taille de la variable de retour. Ceci est représenté dans la figure III.13 en adoptant une syntaxe approximative au langage *sml* pour faciliter la compréhension. La difficulté apparaît dans la définition de cette fonction, où il est nécessaire d'implémenter une stratégie dite *de persistance* (figure III.13). Lors du premier appel de la fonction scalaire f, le rôle de la persistance est celui d'appeler une seule fois une variante vectorielle de la fonction (f_v dans la figure III.13) par calcul du modèle et de stocker la valeur de retour dans une variable globale membre de la classe de résolution. Ainsi, à chaque appel, la fonction f retournera la valeur stockée à l'indice i passé en argument. Sachant qu'il n'existe pas une manière générale d'implémenter la persistance, une solution évidente et efficace est l'appel direct à la fonction f_v, mais cela suppose l'existence d'une variable vectorielle en sortie. Ceci implique l'implémentation d'une version vectorielle du langage de modélisation.

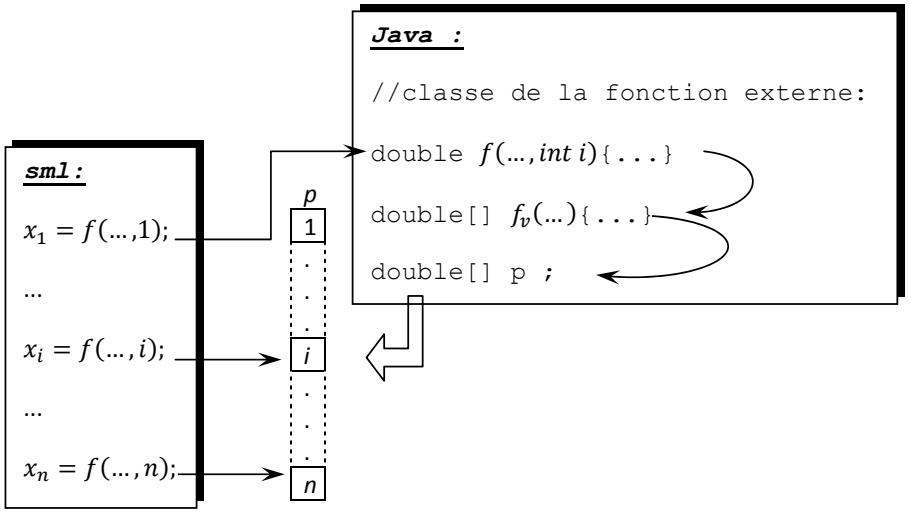


FIGURE III.13 – Persistance pour les fonctions externes vectorielles

III.3.1.1 Intérêt de la vectorisation

L'utilisation des variables vectorielles répond non seulement au besoin d'utilisation des fonctions externes, mais également à tous problèmes de programmation quel que soit le langage :

- les vecteurs fournissent un mécanisme pour déclarer et accéder à plusieurs données élémentaires seulement avec un mon de variable, simplifiant de ce fait la tâche de la gestion de données.
- augmentation du niveau de généralité du code.

III.3.1.2 Dérivation des modèles vectoriels

Le compilateur du langage doit donc gérer la génération du code de calcul du modèle et du calcul des sensibilités pour un modèle de dimensionnement contenant des variables

vectorielles. Les moteurs de calcul formel actuels sont normalement spécialisés pour dériver des fonctions vectorielles avec un effort de programmation de la fonction de calcul des sensibilités comparable avec un outil de dérivation de code, comme montré en [30]. De plus, en [30], les auteurs estiment que le temps de génération du code source de calcul des sensibilités d'un moteur de dérivation symbolique, tel que Macsyma [72], augmente vite avec le nombre des variables de la formule à dériver, sinon impossible d'aboutir. Au contraire, la Dérivation Automatique n'est pas limité en ce sens (voir paragraphe I.4.5.1). La possibilité de dériver des modèles vectoriels permet d'envisager des modèles de dimensionnement beaucoup plus importants en variables de dimensionnement. Nous proposons donc la Dérivation Automatique pour évaluer les sensibilités.

III.3.1.3 Typage de données pour la modélisation vectorielle

Nous proposons dans ce paragraphe, une solution permettant la modélisation vectorielle dans l'environnement CADES. Cette stratégie implique de nombreuses modifications au niveau du compilateur du langage *sml*. Avant tout, le passage en vectoriel demande l'introduction d'un nouveau type de variables. En général, la coexistence des types de données multiples dans un langage informatique peut être à l'origine d'un comportement indésirable suite aux erreurs de typage. Par exemple, l'affectation suivante $u = v$ est fausse dans toutes les conditions d'exécution, si les variables dans les deux membres n'appartiennent pas au même type de données. Certains compilateurs détectent les erreurs à la compilation si le typage n'est pas respecté. De ce point de vue, les langages de programmation sont distingués en théorie en langages à *typage statique fort* et à *typage statique faible* [10]. A ceci s'ajoute le typage dynamique qui permet la conversion à l'exécution de certaines variables d'un certain type en autre similaire.

Le langage *sml scalaire* est fortement typé statiquement puisqu'il autorise seulement l'utilisation des variables scalaires. Il est important de garder cette propriété lors du passage en vectoriel. Cela facilite la compréhension du langage et la modélisation pour l'ingénieur concepteur qui n'est ni informaticien ni quelqu'un qui maîtrise les techniques de dérivation. Dans l'environnement CADES, le programme de calcul représentant le modèle mathématique de dimensionnement est compilé dans un composant logiciel de calcul (voir paragraphe I.5.2). Les langages cible de génération [4] (Java ou C++) sont fortement typés statiquement, cependant les compilateurs associés peuvent intercepter toute erreur due à un conflit de typage. Néanmoins, il existe des situations dans la modélisation vectorielle, où certaines erreurs ne sont pas détectables et elles peuvent être à l'origine d'erreurs graves ou fatales à l'exécution. De ce fait, le Composant Générateur vectoriel est supposé implémenter un mécanisme de typage statique très strict afin d'éliminer au maximum les erreurs de ce type.

4. Le langage cible de génération est le langage utilisé à la transcription du code de calcul correspondant à un modèle mathématique écrit en *sml*.

III.3.1.4 Spécifications du langage de modélisation vectoriel

DÉFINITION III.1
Un vecteur *dans le langage* sml *représente une liste ordonnée de variables scalaires ou de valeurs numériques de type* `double`.

A. Déclaration explicite des vecteurs

Par défaut, dans le langage de modélisation vectoriel, toutes les variables sont scalaires. Cependant, un vecteur peut être déclaré explicitement. Les modalités de déclaration explicite d'un vecteur sont représentées dans la figure III.14.

```
   sml:
01 declare A[nombre entier];
02 A = [scalaire, ... | expression scalaire, ...];
03 A = f(...);
04 A = vecteur;
```

FIGURE III.14 – Possibilités de déclarations explicites des variables vectorielles

Ainsi, il existe quatre possibilités pour définir explicitement une variable vectorielle.

1. en utilisant le mot clé `declare` ; cela réalise l'allocation d'un tableau avec pour taille le nombre entier spécifié entre parenthèses.
2. Affectation avec une liste ; cela copie tous les éléments de la liste dans le vecteur à créer. A noter que la liste des scalaires en membre droit peut être toute expression mathématique qui retourne un scalaire. Ceci, évidement, n'est pas valide pour les listes situées en membre gauche puisqu'on retombe sur la représentation implicite des formules mathématiques, ce qui n'est pas autorisé actuellement en *sml*.
3. Affectation avec une fonction vectorielle [5] ; cela copie tous les éléments du vecteur renvoyé par la fonction ;
4. Copie des éléments d'un vecteur à un autre ; cela copie toutes les éléments du vecteur du membre droit dans le vecteur du membre gauche de l'égalité ;

En ce qui concerne les fonctions internes, leur syntaxe présentée sur la figure III.4, change lors de la vectorisation pour devenir ce qui est représenté sur la figure III.15. Ainsi, les fonctions peuvent accepter en argument une liste, un vecteur, ou un scalaire. Egalement, elles peuvent retourner une liste, un vecteur ou un scalaire.

5. Une fonction (interne ou externe) vectorielle représente une fonction qui retourne un tableau ou une liste indépendamment du type de ses arguments.

```
sml:
nom (
          nom argument |
          nom argument [nombre entier ] |
          [nom argument,...] ,
          ...
    ) =
          expression mathématique |
          [expression mathématique, ...];
```

FIGURE III.15 – La syntaxe des fonctions internes vectorielles

B. Mode d'allocation des vecteurs

L'allocation de toutes les variables vectorielles est statique, c'est-à-dire que la taille de chacune d'entre elles est connue à la compilation. Cet aspect est imposé par le fait que le composant généré est ultérieurement utilisé dans les services de dimensionnement (e.g. Calcul, Optimisation). Le service d'optimisation offre la possibilité au concepteur de formuler le cahier des charges d'optimisation pour chacune de variables, qu'elle soit en entrée ou sortie. Ce principe devra s'appliquer pour tous les éléments d'un vecteur. Pour cela le nombre d'éléments (la taille du tableau) doit être connu avant l'exécution du calcul du composant.

C. Déclaration implicite des vecteurs

La déclaration implicite d'un vecteur est présentée sur la figure III.16. Etant donné que l'argument de la fonction définie en interne est un vecteur, une conversion implicite pourrait transformer la variable passée en argument (V, qui est par défaut scalaire) en variable vectorielle de taille 2 (la taille de l'argument de la fonction). Ceci est interdit puisque la variable V peut être utilisée par plusieurs fonctions d'un modèle de dimensionnement avec des arguments de tailles différentes.

```
sml:
fonction(A[2]) = sqrt( pow(A[0],2) + pow(A[1],2);
U = fonction(V);
```

FIGURE III.16 – Exemple de typage implicite (interdit) des variables vectorielles. La variable à typer : V

A noter que l'erreur dans le modèle de dimensionnement dans la figure III.16 pourra être éliminée en utilisant une des déclarations vectorielles explicites de la variable V, proposées sur la figure III.14.

D. L'appel des fonctions externes vectorielles

L'utilisation des fonctions externes vectorielles représente un cas spécial. Pour une meilleure compréhension, supposons le troisième cas de déclaration d'une variable vectorielle illustré dans la figure III.14 (`A = f(...)`) et considérons que la fonction du membre

droit est une fonction externe vectorielle. Il n'existe aucun mécanisme permettant de connaître avant l'exécution la taille de retour d'une fonction décrite dans un langage autre que *sml* (Java). Nous imposons donc de préciser la taille du vecteur à affecter (i.e. la taille de A) en adoptant la première possibilité de déclaration explicite dans la figure III.14 (`declare A[nombre_entier]`).

L'utilisation d'une fonction interne vectorielle n'implique pas une déclaration du vecteur à affecter, puisque elles peuvent retourner une liste (qui a une taille fixe, connue à la compilation) ou un vecteur dont la taille est connue. Cette affirmation est évidement fausse si la fonction en question appelle elle-même une fonction externe. Dans ce cas, l'utilisateur sera amené à considérer la même démarche proposée pour un appel direct d'une fonction externe.

III.3.2 Spécifications pour la modélisation semi-analytique

III.3.2.1 Les limites de la modélisation semi-analytique

L'approche de la modélisation semi-analytique discutée aux paragraphes III.2.4 et III.2.5, a un inconvénient majeur : le concepteur doit fournir les dérivées partielles de tout algorithme numérique qu'il souhaite utiliser. Cette tâche devient très complexe et le temps alloué pour aboutir augmente vite avec la complexité de l'algorithme. Nous envisageons dans ce paragraphe d'éliminer totalement cette limite en s'appuyant sur la Dérivation Automatique. L'idée est de dériver automatiquement et de façon transparente pour le concepteur les algorithmes numériques importés dans les modèles de dimensionnement.

III.3.2.2 Spécifications pour la Dérivation Automatique des fonctions externes

Afin d'éliminer totalement la limitation des modèles de dimensionnement évoquée avant, nous proposons une solution évidente, i.e. dériver l'algorithme numérique de la fonction externe avec un outil de Dérivation Automatique. Le deux techniques de Dérivation Automatique discutées au chapitre II peuvent être utilisées. Il est plus convenable d'utiliser un outil implémentant la technique *source-to-source*, puisque nous avons montré que le code de la fonction de base reste inchangé. La technique de *surcharge d'opérateurs* exige des modifications dans le code source pour réaliser la dérivation. Cependant, ces modifications sont mineures et nous allons également les considérer au chapitre IV.

III.4 Conclusion

Ce chapitre spécifie les besoins d'évolution du langage de modélisation *sml*, en discutant, dans une première étape, les limitations majeures avec lesquelles les ingénieurs concepteurs ont été confrontés. Il existe ainsi deux limitations majeures, i.e. le manque des vecteurs et la dérivation des fonctions décrites dans des programmes externes Java.

Ce chapitre montre que la notion des vecteurs dans le langage proposé, implémenté dans l'environnement de conception CADES, n'a du sens que si les variables associées ont une taille précisée lors de la phase de modélisation. C'est typiquement l'allocation statique de mémoire, existante dans la plupart des langages de programmation. De plus, nous imposons aussi le typage de données, lors de l'introduction des vecteurs. Le compilateur de langage *sml* (le *Component Generator*) doit donc être responsable de la vérification du type de chaque variable rencontrée dans le modèle de dimensionnement, et de l'affichage des messages d'erreurs, lors d'une utilisation non conforme.

Un aspect très important concernant les manipulations vectorielles, que nous désirons implémenter dans le langage de modélisation *sml*, fait l'objet des fonctions externes. Leur vectorisation élimine totalement la persistance qui est un aspect difficile à implémenter dans une fonction externe du langage *sml*.

Afin d'offrir sous *sml* la possibilité d'utiliser des algorithmes, nous proposons d'exploiter l'aspect de fonctions externes, existant initialement. Ceci nous permet d'éviter d'implémenter un parseur de langage extrêmement compliqué, qui aurait été nécessaire si les structures de contrôle responsables avec les boucles répétitives ou conditionnelles auraient été décrites directement en *sml*. Ainsi, avec un effort raisonnable d'implémentation, nous nous appuyons sur un langage externe de programmation. Notre but reste d'évaluer les gradients de ces algorithmes en employant la technique de Dérivation Automatique, d'une façon transparente pour le concepteur.

Chapitre IV

Architecture logicielle pour la Dérivation Automatique des modèles de dimensionnement

My thesis is that the software industry is weakly founded, in part because of the absence of a software components subindustry. A components industry could be immensely successful

M.D. McIlroy - 1968

Sommaire

Résumé

Dans ce chapitre nous proposons d'évaluer, d'une façon automatique pour le concepteur, les gradients des modèles de dimensionnement pour l'optimisation. Sachant que la Dérivation Automatique de programmes permet d'évaluer ces gradients sans difficultés majeures, nous proposons d'utiliser cette technique puissante, afin de répondre aux spécifications d'un langage de modélisation évoquées au chapitre III. Nous proposons ici le même support de travail, qui est le langage sml *déjà existant au début de nos travaux.*

Puisqu'il existe différents outils de Dérivation Automatique, une des premières nécessités en démarrant ce chapitre est de spécifier l'outil le mieux adapté à notre problématique. Une fois adopté l'outil de Dérivation Automatique, nous nous intéressons dans un premier temps, à implémenter les fonctionnalités disponibles initialement dans le langage sml*. Autrement dit, nous appliquerons la Dérivation Automatique pour des modèles de dimensionnement analytiques et semi-analytiques scalaires. Pour les modèles semi-analytiques, il est imposé dans cette phase, de dériver des modèles qui appellent des fonctions Java dérivées en externe.*

Nous proposons dans la suite du chapitre des solutions architecturales permettant de faire évoluer le langage de modélisation selon les spécifications du chapitre III. Au final nous comptons disposer d'un nouveau module de génération des modèles de dimensionnement qui pilote automatiquement et d'une façon transparente pour l'utilisateur la Dérivation Automatique pour évaluer les sensibilités requises par les algorithmes d'optimisation basés sur gradients. D'ailleurs, la particularité principale de ce chapitre est qu'il étudie les difficultés, les méthodologies, ainsi que les conséquences du pilotage automatique d'un outil de Dérivation Automatique.

IV.1 Introduction : vers une architecture logicielle pour la description et la résolution de problèmes d'optimisation

La difficulté majeure d'un compilateur d'un langage de modélisation est d'offrir, d'une manière automatique, les gradients des modèles de dimensionnement. Nous considérons que ces modèles sont décrits dans un langage de modélisation et qu'ils sont formulés par des équations analytiques explicites avec la possibilité d'utilisation des programmes décrits dans des langages externes. Pour les formules analytiques, le calcul formel est efficace et facilement automatisable pour l'évaluation des gradients, mais ces aspects se compliquent pour les algorithmes. Ceci-dit, nous proposons d'utiliser la Dérivation Automatique de programmes. Afin d'implémenter effectivement ces aspects, il est nécessaire de trouver des solutions architecturales logicielles accessible au non informaticien. Ces architectures visent principalement le compilateur du langage est le programme cible.

Le support de travail de ce chapitre est le langage de modélisation *sml*, son compilateur (le *Component Generator*) et le programme cible introduit au chapitre I sous le nom de composant logiciel de calcul implémentant la norme *ICAr*.

Le compilateur existant au début de nos travaux, utilise le calcul formel des dérivées utilisables seulement pour des formules analytiques. La solution que nous proposons dans ce chapitre est de concevoir une stratégie de dérivation basée sur la Dérivation Automatique tout en maintenant le compilateur déjà existant. Ceci nous permet d'offrir au concepteur plusieurs alternatives de dérivation et donc de génération des composants de calcul. Ainsi, nous concevons un deuxième module de génération entièrement basé sur la dérivation de code. La stratégie d'intégration de ce module de génération dans l'architecture du *Composant Générateur* est détaillée dans ce chapitre. Ce nouveau générateur doit intégrer les fonctionnalités de son prédécesseur. De plus, nous y ajouterons les fonctionnalités permettant de répondre aux limites du langage de modélisation discutées auparavant.

IV.2 Architecture du composant logiciel de calcul

Le composant logiciel de calcul est le programme cible d'un modèle de dimensionnement écrit dans le langage source *sml*, généré par le compilateur de langage. Il a été élaboré et mis en place dans l'équipe intégrant nos travaux, se conformant au paradigme du composant logiciel décrivant un modèle mathématique défini par le concepteur. Ce composant doit être réutilisable et distribuable, ce qui sont d'ailleurs les caractéristiques d'un composant logiciel en général et il doit contenir tout ce dont il a besoin pour un fonctionnement autonome. Afin de répondre à ce besoin, nous avons conçu et déployé une norme restrictive permettant de mettre en œuvre un tel composant.

Toutes les caractéristiques du composant logiciel de calcul évoquées précédemment, ont pour objectif d'assurer la composition avec d'éventuels services, en vue notamment

d'optimisation. Il est donc nécessaire que ce composant soit capable de spécifier ce qu'il fournit. Ceci est le rôle des facettes.

IV.2.1 La norme ICAr

Au cours des divers travaux précédents effectués au Laboratoire de Génie Electrique de Grenoble, plusieurs types de composants de calcul sont apparus, implémentant des normes diverses. Nous rappelons, au titre informatif, les normes COB [1], CoRe, CoSi, définies pour des besoins spécifiques lors des travaux de thèse de E. Atienza [3] et V. Fischer [40].

Le composant de calcul, pour nos travaux de thèse, implémente la version 2.0 de la norme *Icar* [2] [3]. Cette norme a été définie au cours des travaux de thèse de V. Fischer [40] et B. du Peloux [71] et elle a évoluée au cours de nos propres travaux et de ceux d'autres chercheurs de notre équipe. Les caractéristiques de cette norme sont :

1. Le langage de base de la norme est Java, qui est un langage portable sur plusieurs plateformes, grâce à son système de bytecode. Ainsi, le format du fichier du composant de calcul est une archive *.jar* [4]. Le langage Java offre des mécanismes d'interconnexion avec d'autres langages (typiquement C/C++, FORTRAN, ...) et peut contenir la version du code compilé pour différentes plates-formes. On peut ainsi réaliser des composants utilisables sur différents systèmes d'exploitation cible.
2. Un jeu d'interfaces permettant le chargement et l'accès aux différentes facettes. Nous rappelons qu'une facette est l'abstractisation d'une fonctionnalité implémenté par un composant de calcul (voir paragraphes I.5.2 et IV.2.2).
3. Une norme qui définit les règles de stockage des fichiers de ressources dans le composant de calcul. Le composant de calcul dévient capable d'encapsuler dans son archive *jar* non seulement des fichiers de classe, mais aussi des ressources, comme par exemple des fichiers décrivant sa géométrie pour la visualisation du composant, des dépendances sous la forme des librairies en C/C++ ou Fortran, compilées pour différents systèmes d'exploitation cible, ou tout autre fichier qu'un composant de calcul particulier peut utiliser au cours de son exécution.
4. Un système de manifeste qui répertorie le contenu du composant ICAr. Le manifeste facilite le chargement de l'icar et la vérification de la cohérence de l'implémentation de la norme.

Dans la modélisation orienté-objets, le postulat de *plusieurs facettes contenues dans un composant* repose sur la représentation du composant de calcul par un objet qui stocke

1. Le terme *COB* est l'acronyme pour *Computational Object*.
2. Le terme ICAr est l'acronyme de *Interfaces for Component Architecture*.
3. On adopte souvent, lors d'un abus de langage, le terme *composant ICAr* pour les composants de calcul.
4. Le terme *jar* est l'acronyme de *Java Archive*. Ces archives ont été définies dans les normes de composants *javabean*

les références de toutes les facettes disponibles. Cet aspect est visible sur le diagramme UML [5] donnée sur la figure IV.1.

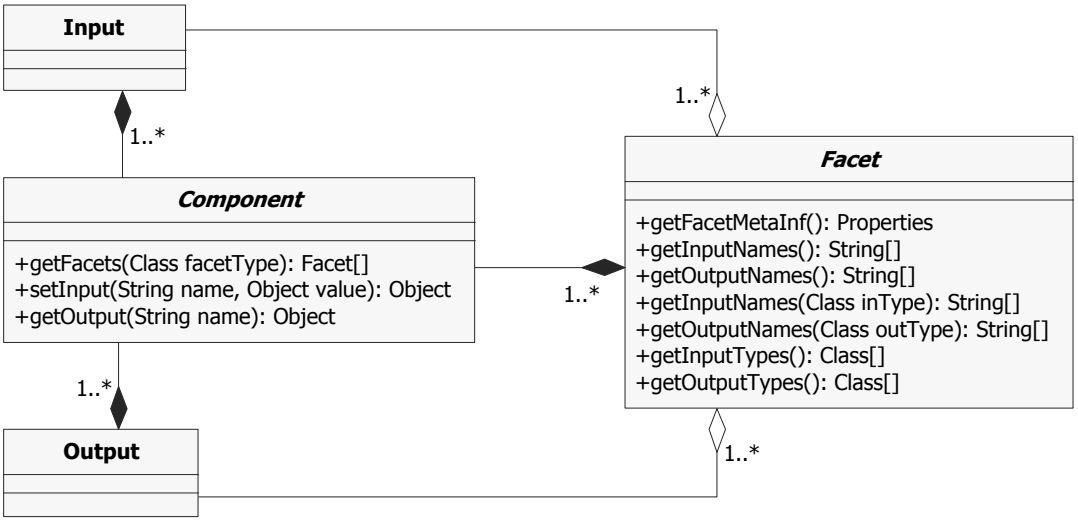


FIGURE IV.1 – L'architecture du composant de calcul implémentant la norme *ICAr*

Dans la représentation donnée sur la figure IV.1, la relation de composition illustre qu'un ICAr contient au moins une ou plusieurs facettes. Une facette est un agrégat de plusieurs entrées qui, grâce à une fonctionnalité, résout un agrégat de sorties. Une facette n'a donc de sens qu'avec au moins une sortie et au moins une entrée.

En plus de ces éléments spécifiant le composant de calcul, il existe une série d'outils qui permettent facilement son exploitation. Ces outils portent notamment sur le chargement, l'écriture et l'exploration des ressources. Toutes ces fonctionnalités assurent l'évolutivité et la portabilité sur plusieurs plateformes d'exploitation. Cette norme donne la possibilité à un composant de calcul d'implémenter une série de facettes que nous détaillons dans la suite de ce paragraphe.

IV.2.2 Les facettes d'un composant de calcul implémentant la norme ICAr

Les facettes donnent au composant *ICAr* l'aspect schizomorphe [6][40], qui représente le caractère multiple d'un composant de calcul. Comme on l'a vu précédemment, un composant rend une série de services correspondants à des tâches particulières, par exemple, le calcul du modèle, le calcul des gradients, etc. Pour chaque service rendu (facette), le composant a une nature différente donc il peut être vu différemment sous des angles multiples. La notion de schizomorphisme caractérisée par des vues multiples du composant est illustrée sur la figure IV.2

5. *UML* est l'acronyme de *Unified Modeling Language* - "langage de modélisation unifié" qui est un langage graphique utilisé dans la modélisation objet en Génie Logiciel.

6. Le terme *schizomorphisme* vient du grec *schizo* - "séparé" et *morphè* - "forme".

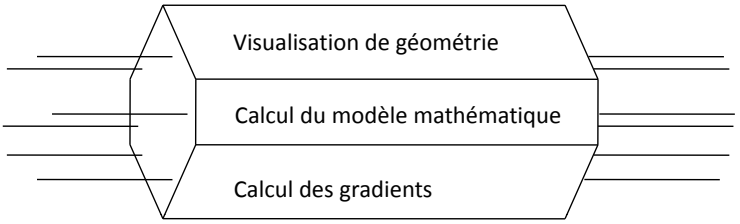


FIGURE IV.2 – Le schizomorphisme du composant logiciel du calcul

Nous introduisons dans ce paragraphe une collection de facettes utiles pour l'optimisation, mais également pour le dimensionnement en général. L'utilisateur a la possibilité sous CADES de choisir dans cette liste, les facettes appropriées à son besoin. A noter qu'en tant qu'utilisateur, il n'existe pas de moyen dans CADES de définir ses propres facettes. La définition des facettes est réservée aux développeurs de cet environnement logiciel. En tant que développeurs, nous tenons à offrir les fonctionnalités nécessaires au concepteur, par l'intermédiaire de la création des facettes et de leurs services correspondants.

IV.2.2.1 La facette de calcul

La facette de calcul, baptisée dans CADES du nom de `ModelSolver`, implémente le calcul de sorties du modèle mathématique à partir des valeurs d'entrées. Il s'agit donc de la loi qui relie les entrées avec les sorties par le biais du modèle mathématique défini par le concepteur. Dans la figure IV.3 est représentée une vue générique de cette facette ainsi que son modèle UML permettant de réaliser le calcul effectif des valeurs des sorties.

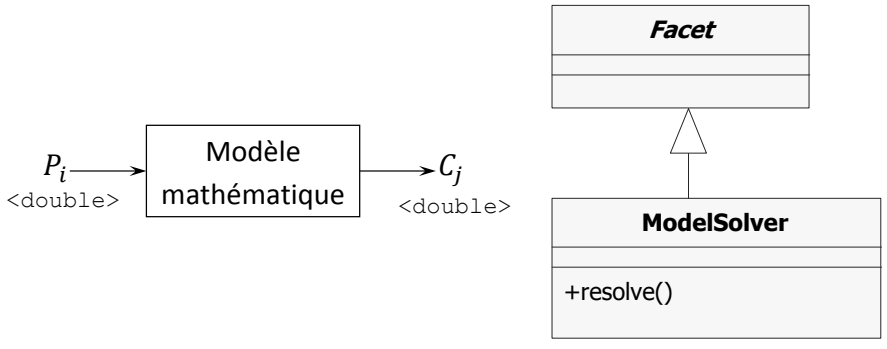


FIGURE IV.3 – La facette de calcul ; a. Le principe de calcul ; b. Le modèle de la facette de calcul

Cette facette est fondamentale. Elle est utilisée par tous les algorithmes du service d'optimisation. D'autres services implémentés sous CADES utilisent cette facette, par exemple le service de traçage des courbes des sorties en fonction d'une entrée (voir l'Annexe B.3). La méthode `resolve()` permet le calcul de sorties en spécifiant à priori les valeurs pour toutes les entrées de cette facette en appelant la méthode `setInput()` de la classe `Component`. Les valeurs de sorties sont récupérées de la même manière, en appelant la

méthode membre `getOutput()` du composant contenant la facette (voir figure IV.1).

IV.2.2.2 La facette de calcul des gradients

La facette de calcul des gradients, baptisée sous CADES du nom de `JacobianSolver`, implémente le calcul des dérivées partielles de toutes les sorties par rapport aux entrées. De même que pour la facette de calcul, les entrées sont reliées aux sorties par le modèle mathématique. Contrairement à la facette de calcul, cette facette n'évalue pas des sorties de type `double`, mais une seule sortie de type objet `Jacobian` qui a la possibilité de fournir la matrice jacobienne complète.

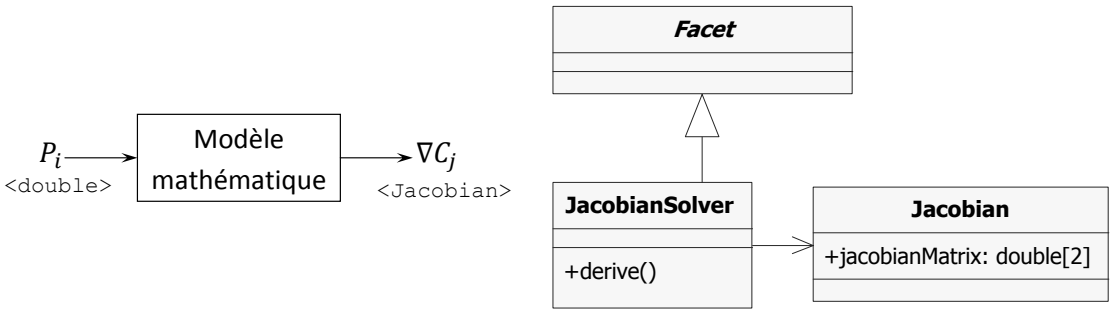


FIGURE IV.4 – La facette de calcul des gradients ; a. Le principe de calcul des gradients ; b. Le modèle de la facette de calcul des gradients

Cette facette est notamment indispensable pour les services d'optimisation utilisant des algorithmes basés sur le calcul des gradients.

IV.2.2.3 Autres facettes

Une liste contenant d'autres facettes existantes dans la version 2.0 de la norme *ICAr* est illustrée dans l'Annexe B.1. Ces facettes ne font pas forcement l'objet de nos travaux, bien que certaines d'entre elles ont été développées au cours de nos activités.

IV.3 Architectures des modules de CADES

Le composant logiciel de calcul, présenté dans le paragraphe IV.2, est utilisable dans une série des services, grâce à son système de facettes. Il est très probable que le processus d'utilisation d'un tel composant, déployé sur l'architecture logicielle implémentant la norme *ICAr*, représente une tâche difficile pour le concepteur qui n'est pas un ingénieur du Génie Logiciel. C'est pourquoi, l'environnement logiciel CADES propose une série de modules (logiciels) qui permettent l'utilisation facile du composant *ICAr* en éliminant tout effort de programmation venant du concepteur. Ces modules réalisent notamment le chargement automatique du composant et ils implémentent une série des services dans le but de faciliter l'utilisation de ses facettes.

Ce paragraphe décrit le module en charge de l'optimisation appelé *Component Optimizer*, celui responsable du calcul sur la base d'un composant baptisé *Component Calculator* et le compilateur du langage *sml* (le *Component Generator*).

IV.3.1 Le *Component Optimizer*

Comme son nom l'indique, le Component Optmizer permet de réaliser des itérations d'optimisation sur un composant de calcul. La figure IV.5 illustre le principe de fonctionnement du Component Optimizer.

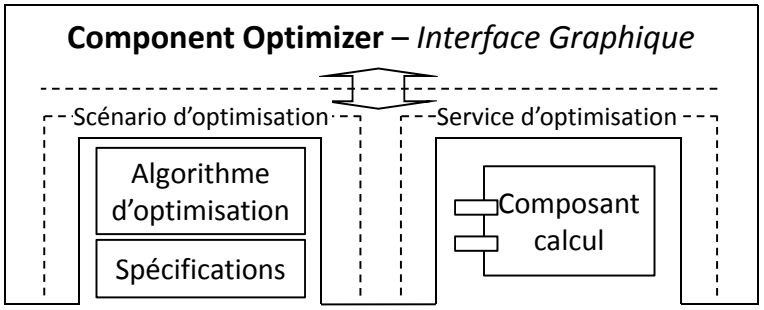


FIGURE IV.5 – La structure du *Component Optimizer*

Le module d'optimisation propose une interface graphique facile à utiliser permettant au concepteur de choisir son algorithme d'optimisation et de définir un cahier des charges pour un composant de calcul *ICAr*. Dans l'Annexe B.2 est présentée une vue générale de cette interface homme-machine.

Le module d'optimisation implémente les fonctionnalités suivantes :

1. *Chargement d'un algorithme d'optimisation* - le concepteur a notamment le choix entre deux catégories d'algorithmes d'optimisation. Il s'agit des algorithmes déterministes comme VF13 [84], implémentant la méthode SQP [7] [49] et ceux stochastiques représentés dans le module d'optimisation par un jeu d'algorithmes génétiques implémentés lors des travaux de thèse de N. H. Hieu [57]. Il existe aussi la possibilité d'appliquer une stratégie pareto qui prend en compte un cahier des charges particulier et un algorithme d'optimisation mentionné auparavant.
2. *La définition des valeurs initiales d'optimisation* - cette fonctionnalité concerne seulement les variables d'entrée du composant de calcul. Elle a du sens seulement pour les algorithmes d'optimisation qui utilise un point de départ (notamment les algorithmes déterministes).
3. *Définition d'un cahier des charges* - cette fonctionnalité permet au concepteur de définir, charger ou sauvegarder un cahier des charges pour un composant de calcul *ICAr*. Le cahier des charges se formule par un ensemble de valeurs initiales pour les variables d'entrée, un ensemble de contraintes et une fonction objectif. Selon la

7. SQP est l'acronyme du terme anglais *Sequential Quadratic Programming*.

nature des variables de dimensionnent, entrées ou sorties, les différentes types de contraintes applicables pour la définition d'un cahier des charges sont illustrés sur le tableau IV.1

TABLE IV.1 – Différentes types de contraintes disponibles à formuler dans un cahier de charges dans le module d'optimisation

Contraint	Variable de dimensionnement	
	Entrée	Sortie
Valeur initiale	√	-
Egalité	√	√
Intervalle	√	√
Fonction objectif	-	√

4. *Le service d'optimisation* - ce service se charge de la gestion du procédé d'optimisation jusqu'à la convergence de l'algorithme utilisé. Ce service utilise notamment la facette de calcul du composant *ICAr*, présentée au paragraphe IV.2.2.1, pour évaluer les valeurs de toutes les variables de sortie utilisées par les algorithmes d'optimisation déterministe à chaque itération ou par un algorithme génétique pour créer des populations. En plus, pour les algorithmes SQP, ce service utilise aussi la facette de calcul des gradients, ou bien des gradients sélectifs (voir l'Annexe B.1.1) si cette dernière est disponible dans le composant de calcul.
5. *la sauvegarde et la visualisation des résultats d'optimisation.*

IV.3.2 Le *Component Calculator*

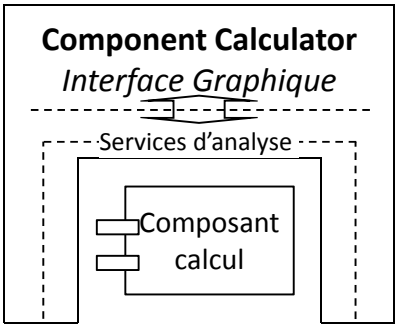


FIGURE IV.6 – La structure du *Component Calculator*

Au cours des itérations d'optimisation, il peut exister des situations pour lesquelles les domaines de définition de certaines fonctions (ou des dérivées) du modèle mathématique de dimensionnement ne respectent pas les contraintes définies par le concepteur. Le module d'optimisation présenté dans le paragraphe IV.3.1 n'offre pas d'informations liées à la consistance mathématique du cahier des charges. Ainsi, quand une optimisation ne converge pas, il peut être très difficile d'en trouver la cause. Pour faciliter la recherche d'un

cahier des charges correct, une analyse du modèle de dimensionnement peut être réalisée dans le module s'appelant *Component Calculator*. De la même façon, ce module charge un composant de calcul *ICAr* et propose une série de services, tous dans le but d'accélérer le processus de dimensionnement. Ceci est illustré sur la figure IV.6.

Les services implémentés par ce module sont donnés dans la liste suivante :

1. *Le service d'affichage des valeurs de sortie* - ce service actualise les valeurs des variables de sortie dans l'interface graphique du *Component Calculator*. Il utilise la facette de calcul pour un ensemble de valeurs d'entrée spécifié par le concepteur.
2. *Le service d'affichage des valeurs de dérivées partielles* - ce service utilise la facette de calcul des gradients, sélectifs ou non, pour actualiser les valeurs de toutes les dérivées partielles de sorties en fonction des variables d'entrée spécifiées par le concepteur.
3. *Le service de traçage des variations des variables de sortie* - ce service utilise la facette de calcul du composant de calcul *ICAr* pour évaluer les valeurs d'une variable de sortie en faisant varier une seule variable d'entrée, spécifiée par le concepteur. Toutes les valeurs sont affichées dans l'interface home-machine à l'aide d'un graphique mathématique.
4. *Le service de traçage des variations des dérivées partielles* - ce service utilise la facette de calcul des gradients, sélectifs ou non, du composant de calcul *ICAr*, pour évaluer les valeurs des dérivées partielles d'une variable de sortie en faisant varier une seule variable d'entrée, spécifiée par le concepteur. Tout se matérialise sous la forme d'un graphique mathématique.
5. *Le service de visualisation* - ce service utilise la facette de visualisation pour actualiser un dessin du dispositif en fonction des paramètres géométriques (voir l'Annexe B.1.5).
6. *Le service de calcul de sensibilités* - ce service utilise la facette de calcul des gradients, sélectifs ou non, pour actualiser les influences d'un ensemble de variables d'entrée sur un jeu de sorties. Ces influences sont calculées et affichées dans l'interface graphique du Component Calculator sous la forme de variations absolues ou en pourcentage. Ce service est utile pour l'analyse des sensibilités du composant de calcul.

   Des vues graphiques détaillées de l'action de ces services sont données dans l'Annexe B.3.

IV.3.3 Le *Component Generator* - le compilateur du langage *sml*

Ce paragraphe présente un module particulier de l'environnement logiciel CADES permettant la génération automatique d'un composant logiciel de calcul. C'est le compilateur du langage *sml* baptisé du nom de *Component Generator*. Ceci est le seul module de CADES qui n'implémente aucun service utilisant les facettes. Son but est de générer, à partir d'un modèle mathématique, un composant de calcul tel qu'il a été présenté en IV.2.

Le *Component Generator* agit dans la même philosophie qu'un compilateur de langage informatique. Ainsi, un programme écrit dans le langage source *sml* est transformé dans le

langage cible du composant de calcul : une archive Java (*jar*). L'architecture du *Component Generator* est illustrée sur la figure IV.7 :

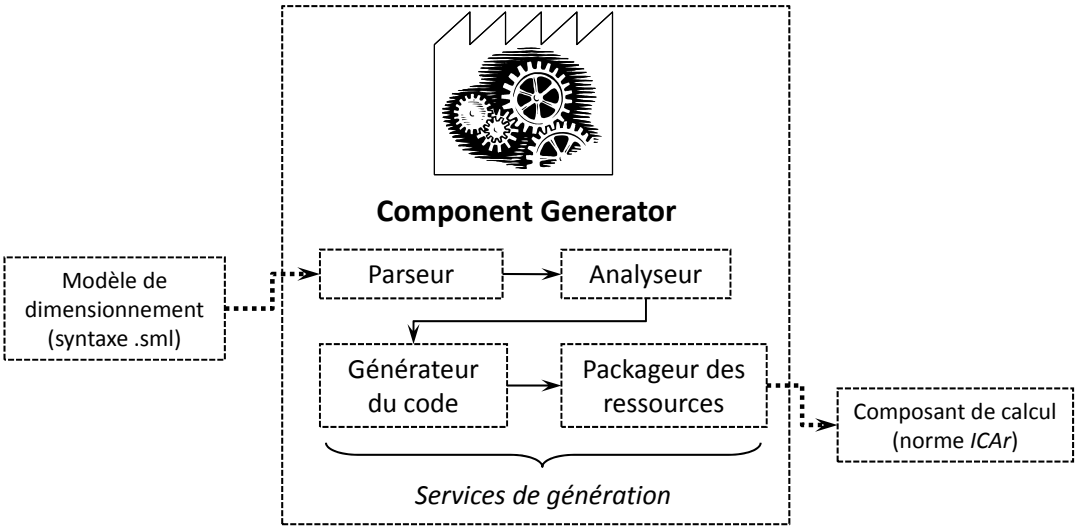


FIGURE IV.7 – La structure du *Component Generator* - le compilateur du langage *sml*

Grâce à sa architecture, le Component Generator peut être vu comme un module implémentant une série de *services de génération*. Ceci est à ne pas confondre avec les services qui rendent possible l'utilisation des facettes du composant *ICAr*. Ainsi, la signification des services de génération est la suivante :

1. *Le service de parsing* - effectue l'analyse lexicale et syntaxique du modèle mathématique en s'appuyant sur un jeu de règles grammaticales.
2. *Le service d'analyse* - effectue l'analyse sémantique en s'appuyant sur les informations fournies par le parseur.
3. *Le service de génération du code de calcul* - génère le code source dans le langage Java, de toutes les classes implémentant la norme *ICAr*, ainsi que l'ensemble de toutes les facettes spécifiées par le concepteur. Ces sources sont enfin compilées avec un compilateur Java.
4. *Le service de packaging* -encapsule toutes les ressources générées par le générateur de code dans une archive Java, le résultat étant le composant de calcul.

IV.4 Dérivation Automatique des modèles de dimensionnement

La Dérivation Automatique de programmes décrits dans un langage de modélisation peut se réaliser en implémentant un outil spécialisé pour ce langage suivant la théorie du chapitre II rélative aux modes de propagation des dérivées et à la technique implémentée. Il est ensuite nécessaire d'intégrer cet outil de Dérivation Automatique dans le compilateur

de langage (s'il s'agit d'un outil source-to source) ou dans le programme cible du modèle (s'il s'agit d'un outil basé sur la surcharge d'opérateurs). La difficulté majeure ici est d'implémenter cet outil de Dérivation Automatique, de le maintenir et de l'actualiser régulièrement conformément aux innovations réalisées dans le domaine. Une deuxième stratégie est d'utiliser un outil de Dérivation Automatique déjà existant pour un autre langage. Ici, l'intégration de cet outil dans le compilateur de langage de modélisation est plus difficile, mais sa maintenance est sa évolutivité peuvent être considérablement réduites, elles étant accomplies dans certains cas par ses développeurs spécialisés. Nous préférons cette deuxième stratégie avec un coût important d'intégration dans un compilateur de langage de modélisation. Nous proposons en même temps des critères de choix exigeantes, afin de diminuer le plus possible sa maintenance et sa évolutivité.

IV.4.1 Définition des critères de choix d'un outil de Dérivation Automatique

Dans la communauté de la Dérivation Automatique, il existe une vaste collection d'outils (une quarantaine [8]). Ces outils supportent divers langages de programmation et implémentent diverses fonctionnalités mathématiques autres que la dérivation de premier ordre (par exemple dérivation d'ordre supérieur, ou dérivation des fonctions inverses, etc.).

Nous sommes particulièrement intéressés par un outil de Dérivation Automatique dans le but de l'intégrer, éventuellement de le faire évoluer dans l'environnement de conception CADES et qu'il soit pérenne. Afin de répondre à ce besoin, nous avons spécifié nos attentes en terme d'outil de dérivation, puis défini une démarche de sélection afin de trouver le meilleur compromis parmi les outils existants. Nos exigences sont élevées pour une utilisation usuelle comme celle montrée dans le chapitre V. Elles sont acceptables dans ce contexte particulier où l'outil doit être piloté automatiquement et d'une manière transparente pour l'utilisateur. Nous avons classé ces critères par catégories et par ordre d'importance décroissante sur la figure IV.8.

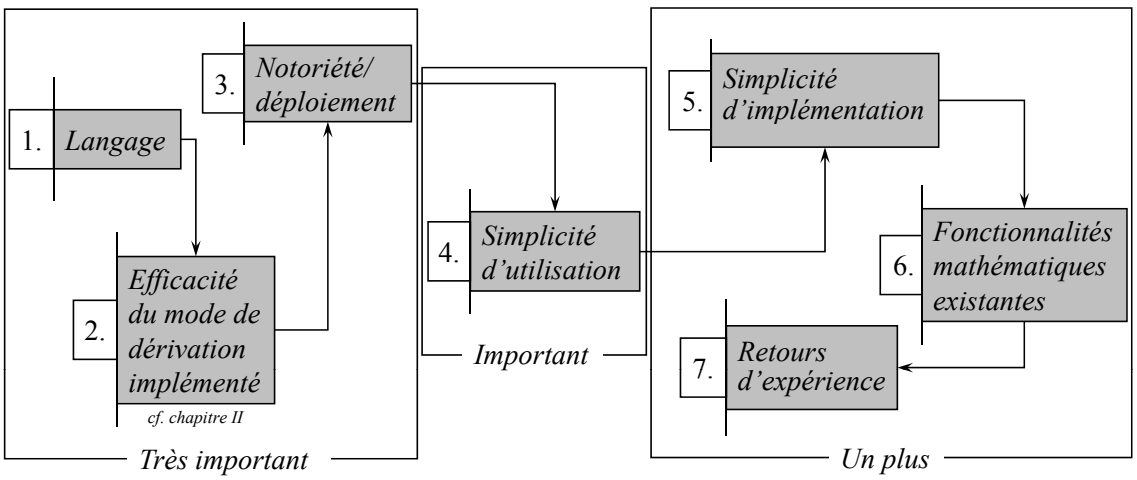


FIGURE IV.8 – Critères de choix d'un outil de Dérivation Automatique

Les critères les plus importantes concernent les performances d'un outil de Dérivation Automatique en termes de simplicité d'utilisation du langage supporté, d'efficacité de(s) mode(s) de dérivation implémenté(s) (e.g. mode(s) *directe* et/ou *inverse* cf. chapitre II) et toutes propriétés qui peuvent se retrouver dans la notoriété de l'outil dans la communauté.

La catégorie suivante concerne la simplicité d'utilisation. Elle peut caractériser la quantité des instructions informatiques à ajouter afin de préparer ou de faire fonctionner correctement l'outil de Dérivation Automatique.

La dernière catégorie concerne des éléments qui pourront représenter un plus du point de vue d'utilisateur, ainsi que de développeur d'un outil de Dérivation Automatique. En premier, il est important de considérer la simplicité d'implémentation de nouvelles fonctionnalités. Deuxièmement, disposer de fonctionnalités mathématiques supplémentaires, autres que celles de dérivation de premier ordre, représente un avantage certain. Elles sont souvent trouvées dans la littérature sous le nom de *pilotes* ("*drivers*"). Le Tableau IV.2 illustre quelques exemples de fonctionnalités mathématiques supplémentaires de certains outils de Dérivation Automatiques existants. Un dernier avantage pourra être le retour et le niveau d'expérience d'utilisation d'un outil.

TABLE IV.2 – Fonctionnalités additionnelles implémentées par divers outils de Dérivation Automatique. *Légende* : *DS* = évaluation des Dérivées d'ordre Supérieur ; *AT* = Arithmétique des coefficients de Taylor ; *CPLX* = Arithmétique ComPLeXe ; *IIMPL* = fonctionnalités quelconques liés aux fonctions Inverses et/ou IMPLicites ; *ODE* = fonctionnalités pour les Equations Différentielles Ordinaires ; *TS* = technique de Transformation Source-to-source ; *SO* = la technique de Surcharge d'Opérateurs ;

Outil AD	Langage	Technique	Fonctionnalités additionnelles				
			DS	AT	CPLX	IIMPL	ODE
ADIC	C	TS	OUI	–	–	–	–
ADIFOR	FORTRAN77	TS	–	–	OUI	–	–
ADiMat	Matlab	TS/SO	OUI	–	OUI	–	–
ADOL-C	C/C++	SO	OUI	OUI	–	OUI	OUI
FADBAD	C/C++	SO	OUI	OUI	–	–	OUI
TAF	FORTRAN95	TS	–	–	OUI	–	–
TAMC	FORTRAN77	TS	OUI	–	–	–	–
TAPENADE	FORTRAN95	TS	–	–	–	–	–
TayIUR	FORTRAN95	SO	OUI	OUI	–	–	–

Source : `www.autodiff.org`

IV.4.2 Analyse des critères de choix d'un outil de Dérivation Automatique

IV.4.2.1 Le langage de dérivation

Le premier critère qui nous a guidé dans le choix de l'outil de dérivation de code est le langage d'utilisation. Nous avons vu que le cœur de calcul du composant *ICAr* est entièrement décrit et intégré dans une architecture Java. Il est donc idéal pour nos

besoins de considérer un outil de dérivation de code supposé supporter des programmes décrites dans ce même langage. Un avantage majeur du langage Java est la simplicité d'utilisation caractérisée par le fait qu'il est fortement typé, que l'utilisation des pointeurs n'est pas autorisé et qu'il gère la récupération des emplacements en mémoire qui ne sont plus utilisés par le système (cette action est connue sous le nom de *Garbage Collection*). Par conséquent, le langage Java est sécurisé du point de vue mémoire et typage des variables [54]. En pratique, la Dérivation Automatique a été implémentée pour des langages comme FORTRAN ou C/C++.

En ce qui concerne Java, ce langage est pratiquement inexistant dans la communauté de Dérivation Automatique. Toutefois il existe des tentatives qui ne sont pas finalisées. Par exemple, dans [81] les auteurs proposent un outil qui implémente une simulation (avec un pré-processeur) de la technique de surcharge d'opérateurs, introduite dans le chapitre II. Cet outil implémente le mode direct de dérivation et ce uniquement pour des fonctions scalaires ; ce que représente évidement un inconvénient majeur. Un deuxième outil présenté en [42] qui s'appelle *JavaDiff*, utilise un préprocesseur qui transforme toutes les opérations mathématiques (i.e $a * b$) en méthodes Java (i.e. $a.mult(b)$). Cette stratégie, mise en œuvre au sein du laboratoire *G2ELab*, est loin d'être applicable pour un programme Java complexe, fonctionnant seulement pour des formules. Cependant, d'autres alternatives en termes de langage supporté par un outil de Dérivation Automatique sont à considérer.

Comme il n'existait pas d'un outil permettant dériver des programmes Java, nous nous sommes orienté vers un autre langage. Evidement, nous visons un langage pouvant communiquer avec Java, le langage du composant *ICAr*. Le tableau IV.2 regroupe d'autres outils permettant de dériver des programmes écrits dans d'autres langages, comme montre la base de données sur les outils de Différentiation Automatique issue de [8]. Nous pouvons distinguer trois langages différents (FORTRAN, C/C++, Matlab) qui sont supportés par les outils les plus reconnus dans la littérature. Parmi eux, bien que le langage Matlab soit facilement interconnectable avec le langage Java, nous l'avons rejeté car il est lié à son environnement de programmation. Ainsi, uniquement les outils supportant les langages FORTRAN et C/C++ ont retenu notre attention.

IV.4.2.2 L'efficacité du mode de dérivation

Le deuxième critère proposé est strictement lié à l'efficacité du mode de dérivation implémenté. Nous sommes particulièrement intéressés par le mode direct de dérivation puisque souvent dans les modèles de dimensionnement le concepteur peut faire appel aux algorithmes itératifs, comme par exemple les résolutions des systèmes d'équations implicites. Nous rappelons que le mode inverse de dérivation n'est pas recommandé à ce genre d'algorithmes. Toutefois, le mode inverse n'est pas ignoré complètement dans notre travail, puisqu'il est plus efficace de l'appliquer au cas où le nombre de variables de sortie est inférieur au nombre des entrées (voir paragraphe II.2.2.4).

Dans le chapitre II, nous avons montré que les outils implémentant la transformation

source-to source sont plus efficaces pour le calcul de dérivées, que les outils qui implémentent la technique de *surcharge des opérateurs*. En revanche cette dernière technique gagne en utilisant la stratégie dite *trace de calcul* qui permet d'obtenir une efficacité remarquable sinon comparable avec son concurrent, lors des évaluations multiples des dérivées. En effet, dans un scénario d'optimisation, les gradients sont évalués pour chaque itération ; cette stratégie devient alors très efficace et elle retient notre attention. Ainsi, après ce raffinement de raisonnement, les outils potentiels implémentent la transformation source-to-source ou évaluent les dérivées suite à une trace de calcul.

IV.4.2.3 La notoriété

En ce qui concerne la notoriété des outils, ceci étant le critère suivant proposé, le tableau IV.3 regroupe les outils les plus reconnus selon la [8]. Les indicateurs de *notoriété* sont liés au nombre des citations dans les publications de la spécialité et leur dernière utilisation.

IV.4.2.4 La simplicité d'utilisation

La simplicité d'utilisation caractérise dans ce contexte toute la chaine nécessaire d'instrumentation/génération du code de calcul des dérivées jusqu'à leur évaluation et utilisation ultérieure. Un outil de transformation source-to-source est évidement plus facile à utiliser puisque l'utilisateur n'a pas à instrumenter le code source initial à dériver. Au contraire, la surcharge d'opérateurs exige des modifications du code source initial. Toutefois, ces modifications sont mineures et nous pourrons les prendre en charge automatiquement.

IV.4.2.5 L'utilisabilité du langage supporté

L'utilisabilité du langage supporté par le dérivateur pourra représenter une difficulté majeure dans ce contexte. Nous rappelons que notre besoin gravite aussi autour d'un langage facilement connectable avec Java. La figure IV.9 montre les possibilités d'interconnexion Java/C++ et Java/FORTRAN.

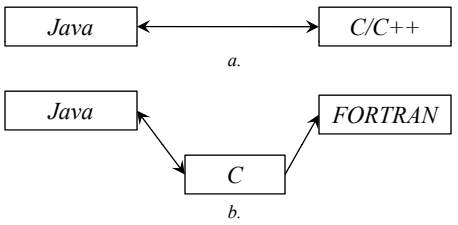


FIGURE IV.9 – Interconnexion du langage Java ; a. avec C/C++ ; b. avec FORTRAN

Un bon compris pourra être le langage C/C++ qui ressort de la figure IV.9 comme étant plus facile à interconnecter avec Java. Il donne aussi un maximum de souplesse

en programmation et qui nous assure une évolutivité augmentée de notre environnement de conception. Ce langage implémente beaucoup de structures de contrôle ainsi que des aspects orienté-objet (polymorphisme, héritage, friendship, etc.).

IV.4.2.6 La simplicité d'implémentation

En ce qui concerne la simplicité d'implémentation d'un outil de Dérivation Automatique nous rappelons qu'un outil reposant sur la surcharge d'opérateurs est plus facile et plus élégant à implémenter. Ces outils offrent souvent des fonctionnalités mathématiques en complément de la dérivation de premier ordre.

IV.4.2.7 Bilan des critères de choix

Nous faisons le bilan de ces discussions dans le tableau IV.3 en prenant en compte la liste des outils évoqués dans le tableau IV.2. En complément de ceci, s'ajoute la notification que le laboratoire *G2ELab* possède une expérience antérieure avec l'outil de dérivation de code appelé ADOL-C [51, 90] lors des travaux de Thèse de V. Fischer en 2004 [40] et du DEA de E. Dezille [28].

Table IV.3 – Bilan des critères de choix des outils de Dérivation Automatique

	ADIC	ADiMat	ADOL-C	FADBAD	TAMC	TAPENADE
Langage	+	–	+	+	-	-
Eff. Dériv.	+	+	+	+	+	+
Notoriété	2008	2009	2009	2008	2008	2008
Eff. utilis.	++	+++	++	++	+	+
Implément.	+	+	++	++	-	-
Fonc. Addit.	+	++	++++	+++	+	-
Expérience			+			

Le tableau IV.3 montre que l'outil ADOL-C [90] est le meilleur compromis pour la plupart des critères définis dans la figure IV.8. Cet outil implémente la technique de *surcharge d'opérateurs* en stockant tous les opérateurs, opérandes, fonctions mathématiques de base dans une *trace de calcul* pour dériver des programmes informatiques décrits en C/C++. C'est lui que nous avons retenu dans la suite de notre travail.

IV.5 Mise en œuvre des fonctionnalités existantes dans la modélisation initiale

Comme ADOL-C [90] dérive des programmes écrits en C/C++, tout le noyau de calcul du composant *ICAr* devra donc être généré en C/C++. Nous rappelons que l'architecture du composant de calcul *ICAr* est implémentée sous Java.

IV.5.1 Interconnexion Java-C/C++ en vue d'utiliser la Dérivation Automatique

IV.5.1.1 Spécifications du couplage Java-C/C++

Nous devons définir une solution afin d'interconnecter *l'ICAr* avec le noyau de calcul décrit en C/C++ qui réalise la dérivation avec ADOL-C. Cette stratégie repose sur l'utilisation du code dite *natif* en Java [63]. Ces aspects sont illustrés sur la figure IV.10

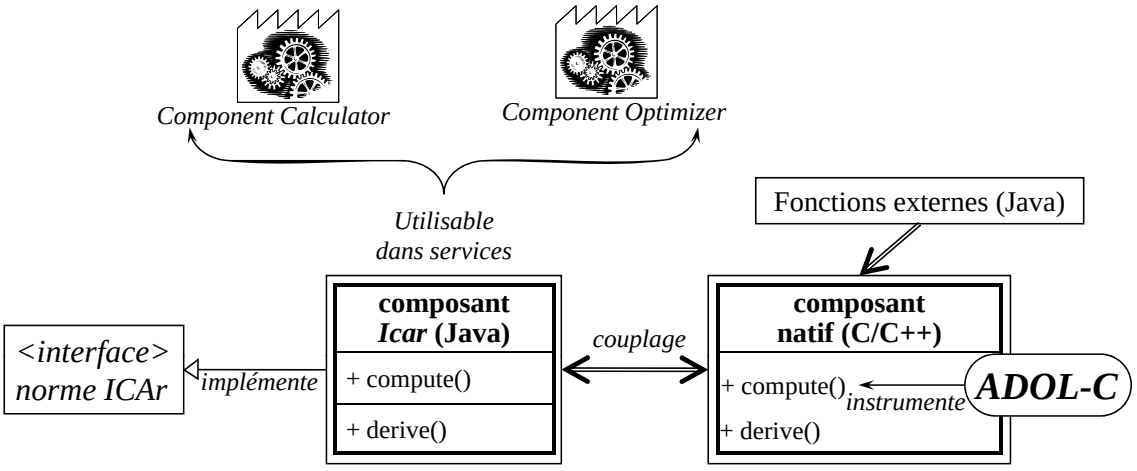


FIGURE IV.10 – Couplage entre le composant de calcul *ICAr* (Java) et le programme de calcul natif (C/C++) instrumenté pour la dérivation avec ADOL-C

De façon plus générale, cela signifie la spécification d'une application Java et d'un programme décrit dans le langage natif C ou C++ et compilé dans un format binaire spécifique à la plateforme hôte (Windows ©, Unix). Cependant, l'application hôte en Java, qui est initialement portable (c'est à dire capable de fonctionner sur différentes plateformes d'exécution) perd cette caractéristique importante en appelant des programmes natifs. Ainsi, tout portage vers d'autres plateformes va nécessiter la recompilation des programmes natifs sur le nouveau environnent d'utilisation.

IV.5.1.2 Possibilités d'interconnexion du composant de calcul *ICAr* en vue d'utiliser la Dérivation Automatique

Pour connecter les deux langages, Java et C/C++, plusieurs solutions sont proposées en [63]. Parmi ces solutions, nous rappelons les suivantes :

1. L'application Java et le(s) application(s) *native*(s) résident dans des processus différents en utilisant des systèmes *client/serveur* :
   - une application Java peut communiquer avec une application native via une connexion *TCP/IP* ou un autre mécanisme de communication interprocessus (*ICP*).
   - une application Java peut se connecter à une base de données via *JDBC* [86].
   - une application Java peut bénéficier de la technologie des objets distribués comme *Java IDL* [62].

2. L'application *native* réside dans le même processus que Java.

Ces deux approches ont des avantages et inconvénients. Nous rappelons que le langage Java est sécurisé du point de vu accès mémoire, contrairement aux langages natifs C/C++. La première solution (i.e. connecter l'application Java avec l'application native dans des processus séparés) a l'avantage de maintenir un niveau élevé d'isolation d'erreurs du programme natif. Un accès à un emplacement mémoire invalide ou une autre erreur quelconque survenue à l'exécution de l'application native n'entraîne pas la destruction du processus de l'application Java.

Un *seul* processus a l'avantage que le transfert de données à partir de l'application Java vers l'application native est beaucoup plus efficace. Dans le contexte du composant de calcul *ICAr* ceci représente un avantage majeur à l'appel des fonctions externes. Supposons, dans la figure IV.10, que le programme natif et la machine virtuelle qui exécute une fonction externe résident dans deux processus différents. Il se peut que le composant de calcul natif appelle à plusieurs reprises la fonction externe Java lors d'un algorithme itératif. Dans ce cas, réaliser la connexion à chaque itération entre le processus du programme natif et la machine virtuelle qui exécute la fonction externe, devient extrêmement coûteux. Ceci est très important dans le contexte de notre application où l'efficacité en terme de temps d'exécution ou des ressources utilisés doit être optimisée au maximum. Ainsi, nous préférons la stratégie qui fait communiquer une application Java avec un programme natif dans un seul processus.

IV.5.1.3 Solution proposé pour le couplage du composant de calcul *ICAr* et le programme natif instrumenté avec ADOL-C

La solution proposé par Sun Microsystems [85] (le principal développeur du langage Java) pour appeler un programme natif dans un seul processus est l'interface appelée *JNI* [63] (qui est l'acronyme de *"Java Native Interface"*). Une première règle générale [63] à respecter avant d'utiliser *JNI*, est de concevoir une architecture Java permettant d'éviter le plus possible l'utilisation du programme natif. Cette règle a pour but de répondre partialement à l'inconvénient d'un *seul* processus évoqué ci-dessus. A la rigueur, la fiabilité d'une application *Java-JNI* réside dans la façon de concevoir l'application.

JNI est une interface puissante qui permet une application Java d'appeler une application native et symétriquement, permet à une application native d'utiliser les objets Java qui résident dans la mémoire du processus associé à la machine virtuelle de l'application hôte. De ce fait, cette interface entre les deux langages est souvent caractérisée comme étant une interface à *double sens*. Cet aspect est très important dans le contexte de notre application, plus précisément pour utiliser des modèles semi-analytiques appelant des fonctions externes, comme sur la figure IV.10. La flèche à double sens (libellé *appelle*) dans la figure IV.11 illustre cet aspect.

La figure IV.11 illustre la nouvelle structure du composant de calcul, intégré dans un programme natif. A noter que norme *ICAr* est conservée, ce qui fait que la composant de

calcul est utilisable dans la liste des services proposés dans l'environnement CADES.

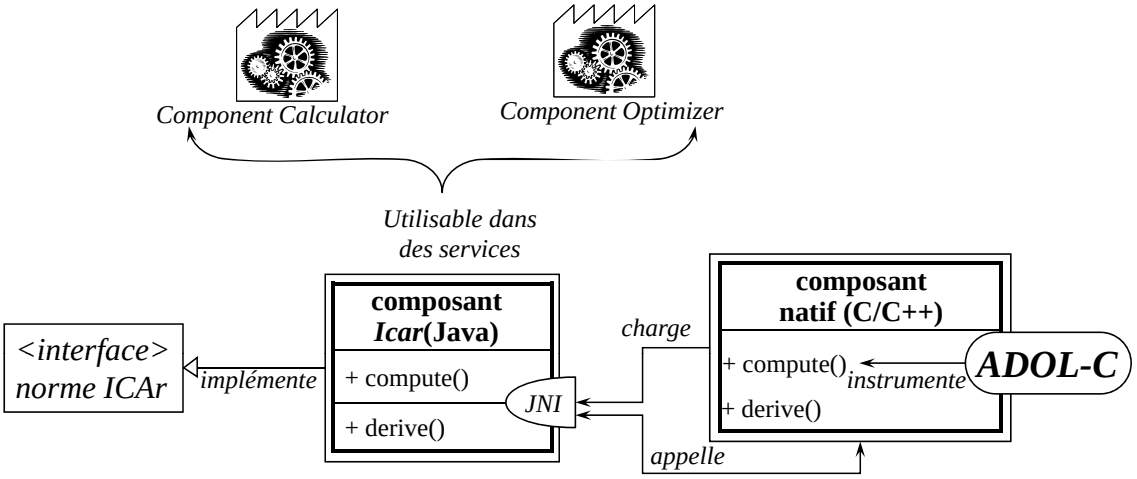


FIGURE IV.11 – Proposition pour une nouvelle structure du composant de calcul, implémentant la norme *ICAr* et utilisant ADOL-C comme outil de Dérivation Automatique

A titre informatif, l'interface JNI liée avec un programme natif ne se résume pas seulement à être pilotée par une application Java. Il est cependant possible qu'un programme natif à son tour invoque une machine virtuelle et crée un environnement Java. Cette alternative répond très bien à nos exigences d'une flexibilité d'utilisation très élevée.

IV.5.2 Un nouveau module du Composant Générateur intégrant la Dérivation Automatique

IV.5.2.1 Spécification de l'architecture de génération modulaire

Afin de réaliser la génération du composant de calcul structuré suivant l'architecture présentée sur la figure IV.11, nous proposons une démarche qui transforme le *Component Generator* présenté sur la figure IV.7, en une entité qui a la particularité de calculer formellement les gradients et de rajouter une deuxième entité qui fourni les dérivées d'un modèle de dimensionnement en utilisant la Dérivation Automatique avec l'outil ADOL-C. Dans un souci de généricité, nous proposons une architecture en plusieurs modules pour le Composant Générateur (voir figure IV.12). Chaque *module de génération* peut s'appeler simplement *générateur*. Nous rappelons que les fonctionnalités des modules prennent le nom de *services de génération*. Un service de génération peut être *commun* s'il est utile pour au moins deux modules de génération, ou *spécifique* à un seul module de génération s'il est redéfini. Ces aspects sont illustrés sur la figure IV.12.

L'approche modulaire présente l'avantage que les services de génération déjà implémentés dans le Générateur initial seront potentiellement exploitables par tout nouveau *générateur*, ou éventuellement par plusieurs modules dans le futur. L'inconvénient majeur est qu'une modification amenée à un service commun évoquée ci-dessous pourra affecter le fonctionnement de tous les modules de génération qui l'utilise. La définition des services

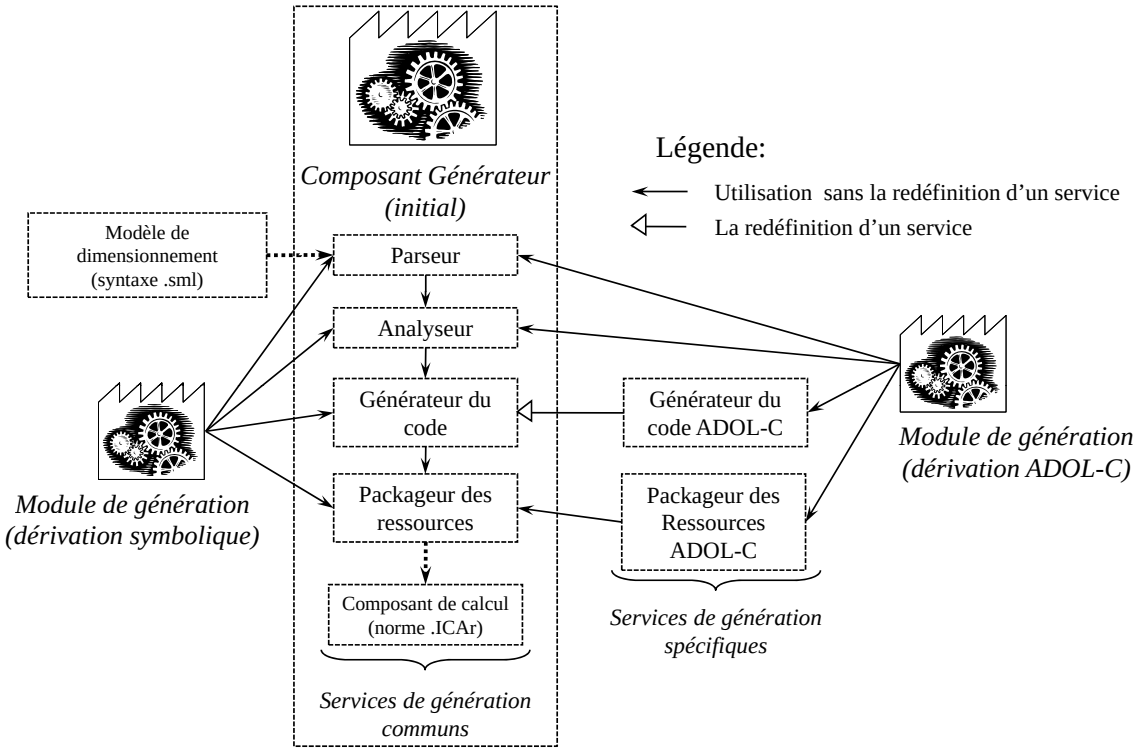


FIGURE IV.12 – Structure des modules de génération utilisant différentes services de génération

spécifiques se réalise en utilisant la notion d'héritage des classes [91], possible dans les langages implémentant les aspects orienté-objet. Plus précisément, les classes d'un service commun vont être spécialisées afin d'implémenter un service spécifique. Sur la figure IV.12, ceci est illustré au niveau du *générateur ADOL-C* où un moteur différent de génération doit être redéfini.

IV.5.2.2 Les services de génération du générateur ADOL-C

A. Les services de génération communs

Symbolisée dans la figure IV.12 par les flèches pleines, la réutilisation des services de génération proposés pour le module ADOL-C est notamment applicable au parsing, à l'analyse et au packaging. Ceci est vrai pour les mêmes spécifications du langage *.sml*.

Nous rappelons que le parseur reconnait toute les éléments spécifiés dans la grammaire du langage *sml*. Ces éléments sont notamment les équations, les fonctions définies en interne et les mots clé autorisés telles que `import, classpath, unit`, etc. Le parsing du module de génération basé sur la Dérivation Automatique reste donc inchangé.

L'analyseur est le responsable de l'analyse sémantique de chaque entité trouvée lors du parsing du modèle de dimensionnement. Nous rappelons qu'il est responsable de l'identification de toutes les variables du modèle de dimensionnement et de leur classification en entrées/sorties. De plus, il agit aussi sur l'ordonnancement des équations, préparant ainsi la génération du code. Ce service reste inchangé pour le module de génération ADOL-C,

ses actions étant suffisantes pour pouvoir passer à l'étape (service) suivante.

Le packaging est responsable de l'encapsulation de toutes les ressources (classes implémentant la norme *ICAr*, librairies, fichiers de description géométrique du composant, etc.) dans une archive Java (*.jar*). Cette archive représente le composant de calcul implémentant la norme *ICAr*. Pour le nouveau module basé sur la Dérivation Automatique, même si ces ressources sont différentes par rapport au générateur initial, ce service reste inchangé.

B. Les services de génération spécifiques

Grâce aux informations fournies par l'analyseur, nous rappelons que le module de génération initial génère le code de calcul et de calcul des dérivées du composant *ICAr* dans le langage Java. Il est imposé pour le nouveau générateur ADOL-C de redéfinir ce service afin de générer le code de calcul en C, l'instrumenter avec l'outil de Dérivation Automatique ADOL-C et JNI et le compiler dans une librairie dynamique propre au système d'exploitation (*.dll* pour l'environnement Windows ©, *.so* pour un environnement UNIX/Linux). Le service de génération devient donc *spécifique* au générateur ADOL-C suite à cette redéfinition.

IV.5.3 Génération des modèles de dimensionnement analytiques

IV.5.3.1 Génération des équations analytiques

Les éléments les plus simples qui peuvent constituer un modèle mathématique de dimensionnement sont les équations analytiques explicites scalaires en combinaison avec les fonctions définies en interne dans le modèle, avec des expressions mathématiques aussi scalaires (pas des algorithmes) et les fonctions mathématiques de base e.g `sin, cos, exp, log`, etc. La dérivation de ces éléments ne représente pas de difficulté majeure pour un moteur de dérivation basé sur le calcul formel [93][41][23]. Cela est valable aussi pour un outil de Dérivation Automatique. Afin d'illustrer le fait que dériver un ensemble d'équations analytiques explicites est assez simple, nous reprenons le modèle analytique de la résistance électrique proposé sur la figure III.2. Ce modèle est d'abord ordonnancé afin de définir la séquence de calcul, puis il est manipulé en vue de le dériver automatiquement. Nous partons ici directement de sa structure ordonnancée, l'ordonnancement des équations n'étant pas le sujet de ce projet [1]. Ce modèle contient un appel à une fonction interne R qui elle même fait appel à une fonction mathématique de base, e.g. `pow` dans notre exemple.

En connaissant la liste complète d'entrées/sorties et les fonctions définies en interne, les éléments suivants doivent être écrits (en respectant l'ordre) dans un programme C++ afin de préparer la dérivation des modèles de dimensionnement avec ADOL-C (l'instrumentation). Plus de détails sur ces étapes sont donnés dans l'annexe A.2.1.

1. Déclarer toutes les variables du type `adouble`.
2. Marquer le début de la trace de calcul.
3. Déclarer les variables indépendantes.

```
sml:
01. R0 = rho*L/S; //Resistance à 0°C
02. R(x) = R0*(1+A*x+B*pow(x,2)); //fonction interne;
03. U = R(T)*I; //loi d'Ohm;
```

```
C++:
//Code C++ généré, instrumenté avec ADOL-C:
...
01 adouble rho, L, S;   //
   adouble A, B, T, I;  /* activation des entrées        */
   adouble R0, U;       /* activation des sorties        */
02
03 trace_on(ID);        /* debout de la trace de calcul  */
       rho <<= rho_;
         L <<= L_;      /* spécifier les indépendantes   */
             ...
04     R0 = rho*L/S;
        U = R(T)*I;     /* le modèle mathématique        */
05
        R0 >>= R0_;     /* spécifier les dépendantes     */
06       U >>= U_;
trace_off();
...
```

FIGURE IV.13 – Génération et instrumentation avec ADOL-C d'un modèle analytique

4. Générer les équations (ordonnancées).
5. Déclarer les variables dépendantes.
6. Marquer la fin de la trace de calcul.

L'ordre de spécification des variables indépendantes/dépendantes donnera l'ordre des dérivées partielles dans la matrice jacobienne. Pour l'exemple présenté sur la figure IV.13, la matrice jacobienne associée sera :

$$J = \begin{pmatrix} \frac{\partial R0}{\partial rho} & \frac{\partial R0}{\partial L} & \cdots \\ \frac{\partial U}{\partial rho} & \frac{\partial U}{\partial L} & \cdots \end{pmatrix}$$

IV.5.3.2 Génération des fonctions internes

Nous rappelons que toutes les fonctions définies en interne dans le modèle de dimensionnement sont analytiques. Pour que la démarche présentée ci-dessus puisse fonctionner correctement il est nécessaire que toutes les fonctions internes définies par le concepteur dans son modèle de dimensionnement, soient générées automatiquement avec les arguments actifs (i.e. `adouble`) ainsi que leur valeur de retour. Cet aspect ne représente pas une difficulté majeure puisque les fonctions internes sont analytiques et scalaires. La génération d'une fonction externe est généralisée sur la figure suivante.

```
sml:
nom ( argument, ...) = expression mathématique ;
```

⇩

```
C++:
//code C++ généré;
adouble nom ( adouble argument, ... ){
    return expression mathématique;
}
```

FIGURE IV.14 – Le syntaxe d'une fonction interne définie dans le langage *.sml* et le code C++ généré

IV.5.3.3 Evaluer les dérivées des modèles analytiques

L'approche de génération du code de calcul des modèles de dimensionnement et d'instrumentation avec ADOL-C est visiblement très simple. Pourtant, il est essentiel puisqu'il représente la base d'utilisation d'ADOL-C dans notre contexte de travail.

```
C++:
zos_forward(        //le pilote pour Zero Order Forward Mode;
        id,         //identificateur de la trace;
        m ,         //nombre des variables dépendantes;
        n ,         //nombre des variables indépendantes;
        k ,         //préparation pour le mode inverse;
        *x,         //les valeurs des variables indépendantes;
        *y,         //les valeurs des variables dépendantes;
)
-----------------------------------------------------------------
jacobian(            //le pilote de calcul de la matrice
                     //jacobienne (dérivation premier ordre);
        id ,
        m  ,
        n  ,
        *x ,
        **J,        //la matrice jacobienne en sortie;
)
```

FIGURE IV.15 – Réutilisation d'une trace ADOL-C pour le calcul des sorties et des dérivées

Une fois construite, la trace de calcul ainsi que les opérateurs et leurs opérandes associés résident dans une mémoire-tampon. Ceci permet sa réutilisation ultérieure. Ces réutilisations concernent soit l'évaluation des valeurs de sortie (les dérivées d'ordre zéro), soit l'évaluation des dérivées du premier ordre, soit l'utilisation des fonctionnalités mathématiques supplémentaires comme celles évoquées dans le tableau IV.2 (systèmes implicites, équations différentielles ordinaires, etc.). Pour offrir une meilleure compréhension, la figure IV.15 illustre la façon de réutiliser la trace identifiée par ID dans la figure IV.13, pour

réaliser le calcul de sorties (le pilote `zos_forward`) et des dérivées partielles du premier ordre (le pilote `jacobian`).

Dans un contexte arbitraire (supposons par exemple une itération d'optimisation) pour un ensemble de valeurs d'entrées quelconque, on peut utiliser, entre autres, les pilotes `zos_forward` et `jacobian` afin d'évaluer les valeurs de sortie et respectivement la matrice jacobienne d'un modèle de dimensionnement. Cela permet d'éviter la réexécution de la trace présentée dans figure IV.13 ce qui sera plus lent. Nous rappelons que les résultats sont correctes, uniquement si toutes les opérations logiques de la trace initiale rendent les mêmes résultats. ADOL-C, ainsi que la plupart des outils implémentant la technique de dérivation de surcharge d'opérateurs, implémente un mécanisme s'appelant *détection de changement de branche conditionnelle*[8]. Grâce à ce mécanisme, ADOL-C notifie l'utilisateur qu'une opération logique a changé lors de l'utilisation de la trace initiale pour un point d'entrée particulier et ceci permet donc de connaître précisément si la trace doit être reconstruite ou non.

En comparaison avec le calcul formel des dérivées, cette approche a été beaucoup plus simple d'implémenter dans le compilateur du langage de conception *sml*. Afin de dériver un modèle de dimensionnement par calcul formel, il est nécessaire d'effectuer pour chaque équation une dérivation et ensuite d'appliquer les règles de dérivation des fonctions composées jusqu'au bas du graphe de calcul (jusqu'aux entrées) afin d'être en mesure d'évaluer les dérivées partielles par rapport aux entrées. Alors qu'avec un outil de Dérivation Automatique, les seules choses qui sont à gérer en plus par rapport au calcul du modèle est l'instrumentation du code de calcul, ce qui représente une modification mineure.

IV.5.4 Génération des modèles de dimensionnement semi-analytiques

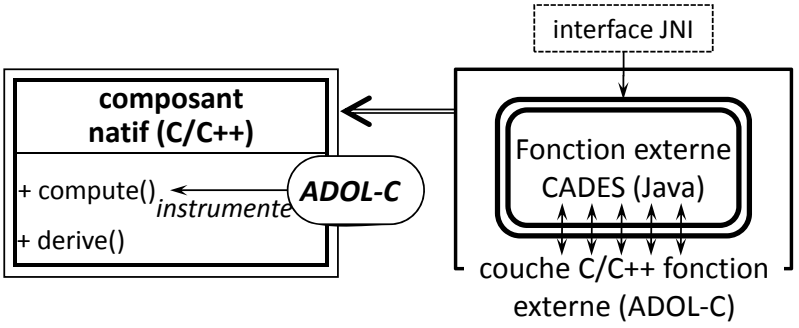


FIGURE IV.16 – Couplage d'un modèle de dimensionnement ADOL-C avec une fonction dérivée en externe Java

Dans ce paragraphe nous nous intéressons à générer le programme cible C++ d'un modèle de dimensionnement semi-analytique écrit en *sml*. La difficulté majeure ici est

8. Le terme *détection de changement de branche conditionnelle* existe en littérature sous le nom de *branch switch detection*.

d'utiliser une fonction externe CADES [9] décrite et dérivée par l'utilisateur en Java ((voir le paragraphe III.2.4)). Ceci implique d'appeler une fonction Java dans la trace de calcul C++ du modèle de dimensionnement. Nous proposons d'utiliser de nouveau JNI afin d'interfacer ces deux langages. L'idée de cette approche est illustrée sur la figure IV.16.

Nous avons implémenté au niveau du générateur ADOL-C cette fonctionnalité qui intègre une fonction externe Java à partir d'une couche C++ native. Afin de créer cette couche nous utilisons le concept de fonctions externes ADOL-C [10] présenté au paragraphe II.3.

IV.6 Mise en œuvre des fonctionnalités pour améliorer la modélisation

Le type de données associé au vecteurs *sml* est dans le langage C++ le `std : :vector<adouble>`. Le choix d'un objet pour représenter une liste de données est justifié par le fait que les opérateurs mathématiques entre les tableaux peuvent être surchargés facilement en C++ afin d'implémenter des opérations comme l'addition, la multiplication, l'inversion de matrices, etc. Même si ces opérations ne sont pas supportées dans le langage actuel de modélisation, elles sont d'un réel intérêt puisqu'elles peuvent grandement faciliter la modélisation.

IV.6.1 Implémentation de la vectorisation

IV.6.1.1 Les équations vectorielles analytiques

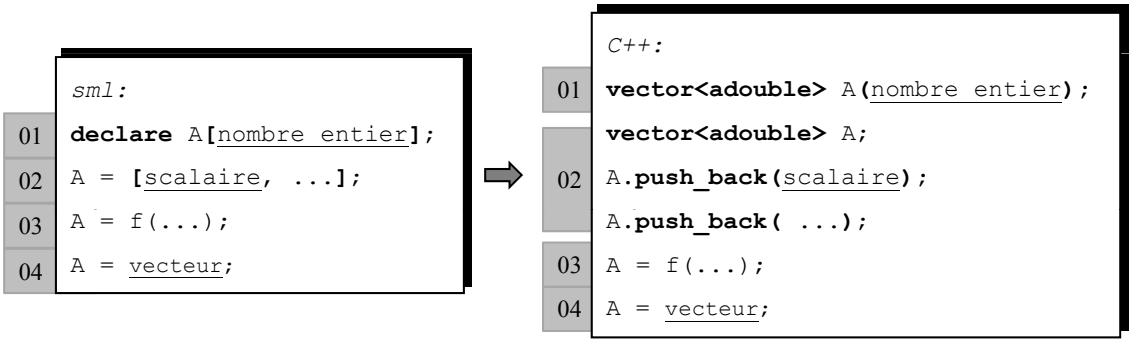


FIGURE IV.17 – Possibilités de déclarations explicites des variables vectorielles et le code C++ associé et instrumenté avec ADOL-C

9. Une fonction externe *CADES* est une fonction utilisée dans un modèle de dimensionnement décrite dans un autre langage que *sml*, notamment Java, qui doit fournir sa valeur et ses dérivées partielles dans tous les points de son domaine de définition.

10. Une fonction externe *ADOL-C* est une fonction appelée dans une trace de calcul des dérivées ADOL-C qui doit fournir sa valeur et ses dérivées partielles dans tous les points de son domaine de définition. Ces fonctions ont une signature spécifique et elles sont d'un réel intérêt au cas où leur code n'est pas disponible en C/C++.

Dans la figure IV.17 nous reprenons les quatre possibilités de déclaration ou construction de vecteurs montrées sur la figure III.14 et on spécifie le code de dérivation avec ADOL-C.

IV.6.1.2 Les fonctions internes vectorielles

Les fonctions internes sont générées en C++ avec touts les arguments du type `adouble`, pour les arguments scalaires ou de type `std : :vector<adouble>` pour les arguments vectoriels. Le retour est une variable de type `std : :vector<adouble>` pour une fonction interne vectorielle. Cela nous permet d'assurer l'application correcte de la règle de dérivation des fonctions composées et d'assurer la continuité du graphe de calcul. Nous reprenons la figure III.15 en spécifiant la syntaxe d'une fonction vectorielle en *sml* et en C++ associé pour la Dérivation Automatique avec ADOL-C.

```
sml:
nom (
          nom argument |
          nom argument [nombre entier ] |
          [nom argument,...] ,
          ...
    ) =
          expression mathématique |
          [expression mathématique, ...];
```

⇩

```
C++:
adouble | vector<adouble>  nom (
                    adouble argument |
                    vector<adouble> argument,
                    ...
                    ){
    return expression mathématique;|
    vector<adouble> _ret;
    _ret.push_back(expression mathématique);
    ...;
    return _ret;
}
```

FIGURE IV.18 – La syntaxe des fonctions internes vectorielles et le code C++ associé et instrumenté avec ADOL-C

IV.6.1.3 Conclusion sur la vectorisation

Dans l'approche vectorielle, les difficultés principales sont liées à la transcription des modèles de *sml* vers C++. Nous avons montré que ces difficultés sont mineures. En ce qui concerne la dérivation des modèles vectoriels, celle-ci ne représente aucune difficulté avec la Dérivation Automatique. Elle se réalise dans la même logique, transparente pour l'utilisateur. Nous avons conçu une stratégie permettant de faire le lien entre les variables indépendantes et celle dépendantes en utilisant des objets C++ qui représentent des listes

dont le type de donnée générique est `adouble`. A partir de là, la Dérivation Automatique avec ADOL-C s'applique *naturellement* n'induisant pas d'autres complications.

IV.6.2 Implémentation de la Dérivation Automatique des fonctions externes numériques

Dans ce paragraphe, nous présentons le nouveau module de génération créé spécialement pour la modélisation par des algorithmes. Ce nouveau module a pour but de générer le code de calcul des dérivées d'un algorithme numérique défini par l'utilisateur en s'appuyant sur la Dérivation Automatique. Le module génère un programme informatique utilisable dans un modèle de dimensionnement décrit dans le langage *sml*. Puisque les fonctions externes, présentées au paragraphe III.2.4, s'utilisent dans la même philosophie, une manière très simple de faire le code de calcul des dérivées utilisable par un modèle de dimensionnement sous forme d'algorithme, est de l'intégrer dans une fonction externe. Ainsi, le nouveau module génère une fonction externe CADES (ainsi que la routine de calcul des gradients) à partir d'un algorithme écrit en C/C++ et tout sera utilisable ultérieurement dans un modèle de dimensionnement.

Dans la définition d'une fonction externe toutes les structures de contrôle existantes dans le langage C++ peuvent être utilisée (branches conditionnelles, boucles répétitives, etc.). De plus, les fonctions générées peuvent être récursives ou itératives.

IV.6.2.1 L'architecture du générateur de fonctions externes

Le module de génération de fonctions externes s'intègre à l'architecture du Composant Générateur modulaire comme le montre la figure IV.19.

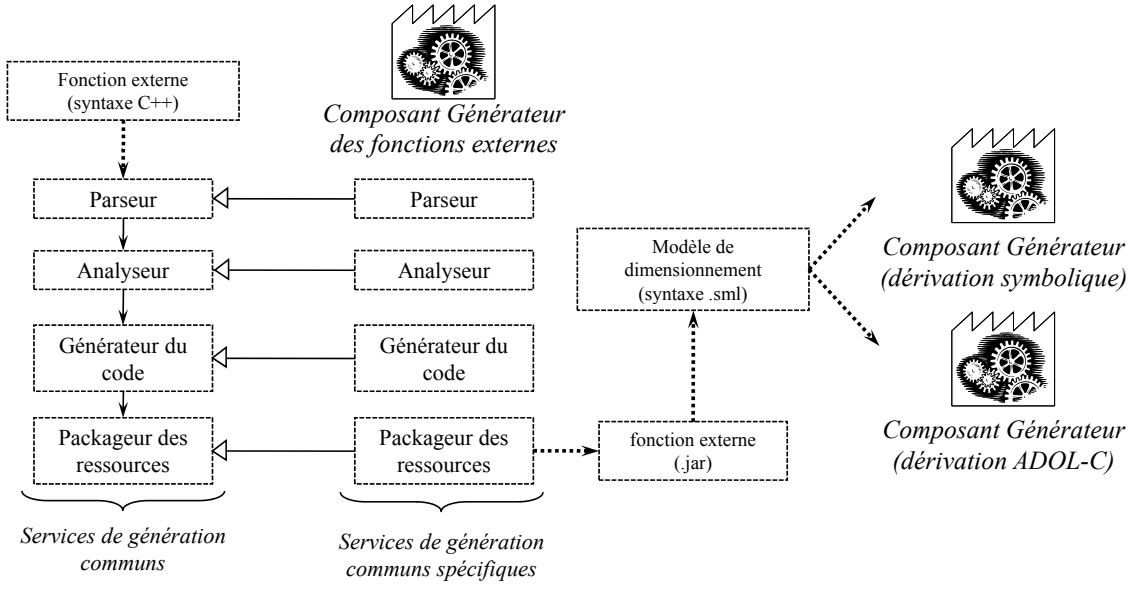


FIGURE IV.19 – Positionnement du module de génération des fonctions externes par rapport aux services de génération du *Composant Générateur*

IV.6.2.2 Les services de génération

Le générateur de fonctions externes [11] redéfinit tous les services de génération, car il n'agit pas sur un modèle de dimensionnement décrit dans le langage *sml*, mais sur une fonction numérique décrite directement en s'appuyant sur la syntaxe C++. Ainsi nous redéfinissons les services ci-dessous :

1. Le parseur - fait l'analyse grammaticale de la signature de la fonction. Plus précisément ce service sauvegarde des informations concernant le type du retour de la fonction et ses arguments ainsi que leur type. Toute déviation de la règle grammaticale de construction d'un prototype d'une fonction C++ est signalée par le parseur en renvoyant des messages d'erreur (i.e. manque des virgules, manque des parenthèses, etc.). Après la signature, le parseur sauvegarde le corps de définition de la fonction en ignorant toute règle grammaticale. Au final, toutes les informations sauvegardées sont renvoyées à l'analyseur.
2. L'analyseur - fait une analyse sémantique du type de retour de la fonction et de ses arguments. Le type de retour peut être `adouble` pour une fonction scalaire ou bien `vector<adouble>` s'il s'agit d'une fonction vectorielle telle qu'elle a été définie auparavant. De même pour les arguments qui peuvent être vectoriels ou scalaires. L'analyseur renvoie des messages d'erreurs pour toute déviation de cette règle. De plus, il renvoie des messages d'erreurs s'il y existe des arguments dupliqués, ce qui n'est pas autorisé en C++. L'analyseur ignore le corps de définition de la fonction.
3. Le générateur - génère le code C++ et Java correspondant à la fonction externe CADES en question en s'appuyant sur les informations concernant la signature et le corps de définition fournies par l'analyseur. En complément ce code est instrumenté avec l'outil ADOL-C pour le calcul des dérivées. L'interface JNI est utilisée pour réaliser le lien avec la classe Java de la fonction externe. De cette façon la fonction est utilisable ultérieurement les modèles de dimensionnement. A la compilation du code C++ évoqué, toute erreur existante dans le corps de la définition de la fonction sera détectée par le compilateur de langage C/C++.
4. Le packageur - encapsule toutes les sources compilées par la générateur sous la forme d'une archive Java *.jar*. Ceci est le produit final qui représente la fonction externe.

IV.6.2.3 Fonction externe optimisée

L'exécution d'une fonction externe peut être optimisée suite au développement de ce module de génération. Nous avons constaté que les appels multiples des programmes natifs en utilisant l'interface JNI ralenti considérablement l'exécution du programme Java. Notamment, les auteurs de [63], signalent que la résolution d'un appel [12] Java - programme

11. Le générateur des fonctions externes est représenté par le nouveau module de génération des fonctions externes.
12. La résolution d'un appel d'une fonction et le processus réalisé par le système d'exploitation ou par l'environnement où le processus s'exécute qui réalise le passage des paramètres à partir du contexte où la

natif est jusque à trois fois plus lent qu'un appel Java - Java.

Si le modèle de dimensionnement est compilé avec le générateur ADOL-C, il devient intéressant de supprimer la couche JNI interfaçant le programme principal et les fonctions externes générées en C++. Ceci est possible en accèdent directement aux ressources C++ dans l'archive de la fonction externe. Ainsi, on supprime deux passages JNI par appel d'une fonction externe (voir figure IV.20b).

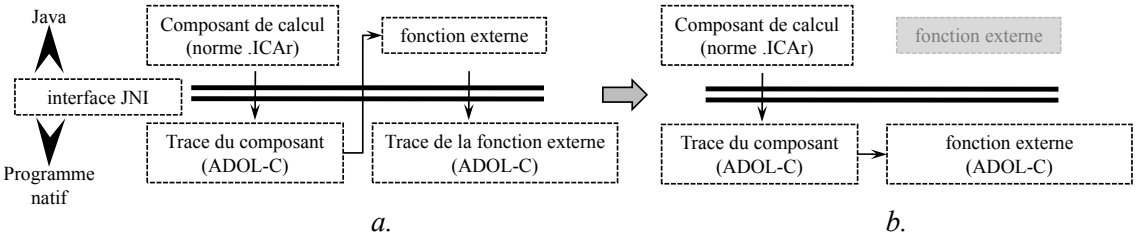


FIGURE IV.20 – Appels d'une fonction externe dans un modèle de dimensionnement exécuté dans une trace ADOL-C ; a. L'appel (*classique*) utilisant deux passages par l'interface JNI ; b. L'appel optimisé en utilisant le code C++ de la fonction externe : aucun passage JNI, appel direct C++ → C++

Le temps d'exécution gagné devient très intéressant dans le cas d'utilisation, par exemple, d'une fonction externe appelée par une fonction de fonction lors d'un processus itératif. Nous avons constaté cet aspect lors de l'utilisation d'une intégrale double sous la forme d'une fonction de fonctions au chapitre V, où l'intégrande représente lui aussi une fonction externe qui évalue une intégrale simple. Le processus étant énormément coûteux en rapidité (jusque à trois fois par rapport à une exécution purement Java), nous considérons que la suppression des passages par l'interface JNI autant que possible est bénéfique dans tous les cas.

IV.6.2.4 Conclusion sur l'approche de Dérivation Automatique des fonctions externes

L'approche de définition d'une fonction externe présentée au-dessus, fonctionne si toutes les variables intermédiaires (définies dans le corps de la fonction) sont de type `adouble`. Les compteurs des boucles ou les indices font exception de cette règle : ils peuvent rester du type entier. Toutefois, le concepteur peut, dans certains cas, utiliser le type de données inactif pour la dérivation (`double`), par exemple quand il s'agit des constantes (telles que précision, erreur, constantes mathématiques, etc.). Le concepteur a cependant la responsabilité d'assurer le lien entre les arguments de la fonction et son retour en passant par des variables actives. Ainsi, nous considérons que notre approche de génération des fonctions externes a deux points faibles :

1. L'utilisateur doit avoir un minimum de connaissances en C++. Toute faute d'allocation de mémoire, utilisation des pointeurs invalides, variables non initialisées, etc.,

fonction a été appelée en respectant certains règles issues d'une convention d'appel.

pénalisera le concepteur à l'exécution avec des erreurs graves accentuées par le fait que le calcul s'exécute dans un programme natif pour la machine virtuelle Java (donc avec des erreurs difficilement analysables).

2. Parfois, l'utilisateur doit avoir un minimum des connaissances sur la structure et les spécificités d'un objet `adouble`.

IV.6.3 Dérivation Automatique des fonctions de fonctions externe

Tout programme ou algorithme dérivé avec ADOL-C est utilisable sous CADES en passant par des fonctions externes ou par la variante plus spécialisée représentée par les fonctions des fonctions (voir paragraphe III.2.5). Afin de dériver avec ADOL-C une fonction de fonctions externe, il est imposé dans un premier temps d'écrire la fonction en question dans le langage C/C++ sur laquelle il faut prévoir une couche Java. Cette couche fait possible l'exploitation de la fonction de fonctions dans un modèle de dimensionnement *sml* et l'appel de la fonction en argument qui est construite sur une architecture Java. Nous réalisons ces aspects manuellement dans nos travaux, mais il peut exister aussi une possibilité automatique.

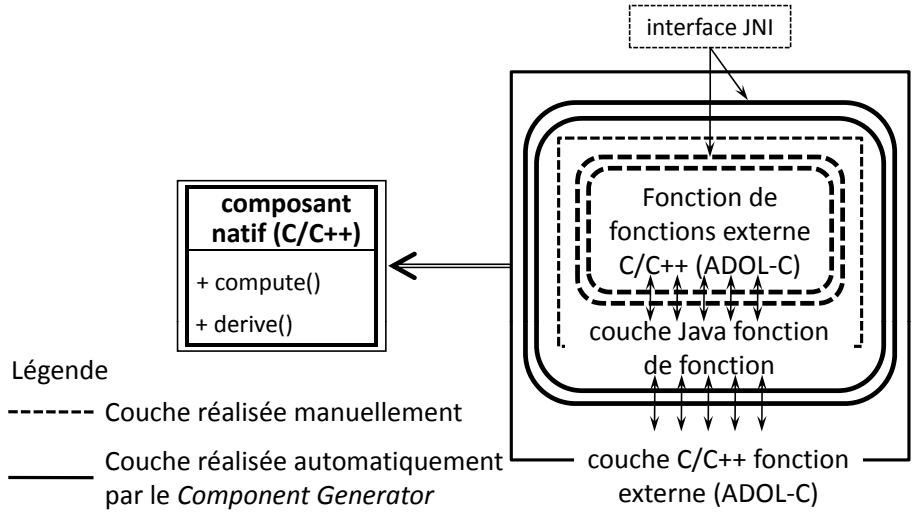


FIGURE IV.21 – Création d'une fonction de fonctions externe en C/C++ et son utilisation dans les modèles de dimensionnement

Le générateur ADOL-C, implémente automatiquement la même stratégie que pour les fonctions externes simples, illustrée sur la figure IV.16 : une couche C/C++ permettant d'encapsuler tout dans une fonction externe ADOL-C.

IV.7 Conclusion

Ce chapitre a été consacré spécialement à l'utilisation d'un outil de Dérivation Automatique intégré dans un environnement logiciel de conception. Plus particulièrement, il détaille le pilotage automatique d'un outil de dérivation de code. Aujourd'hui l'existence

d'une grande diversité d'outils de Dérivation Automatique signifie une grande diversité de philosophies d'utilisation et aussi une grande diversité de fonctionnalités supplémentaire autres que la dérivation de premier ordre. Cependant, il est difficile pour un *débutant* de choisir l'outil qui se positionne le mieux dans son contexte d'utilisation. Nous avons montré, dans un premier temps, certains critères de base à prendre en compte pour le choix d'un outil de Dérivation Automatique. Ces critères répondent bien en général au contexte d'une application orchestrée à piloter automatiquement la Dérivation Automatique.

Le manque d'outil de dérivation de programmes Java et étant donné qu'une grande partie des solutions logicielles ont la tendance aujourd'hui de s'orienter vers ce langage assez puissant, nous proposons des solutions d'interconnexion de ce langage avec un outil de Dérivation Automatique de programmes écrits en C/C++.

Les fonctions externes CADES présentaient une difficulté majeure : le concepteur avait la responsabilité de les dériver d'une manière ou d'une autre. Il était souvent amené à utiliser soit le calcul formel, soit la dérivation à la main. Cependant, il était amené à faire face aux complications bien connues lors de l'utilisation de ces deux techniques. La Dérivation Automatique donne désormais la possibilité en CADES d'utiliser un module spécial qui permet de dériver des algorithmes numériques avec un effort minimal ou au moins nettement inférieur aux techniques initiales.

En ce qui concerne les bénéfices amenés par la Dérivation Automatique dans un environnement permettant l'optimisation utilisant des algorithmes basée sur le calcul des gradients, nous constatons que la Dérivation Automatique est simple d'utilisation. Cette technique puissante nous a amené à gagner beaucoup de temps et d'effort de programmation pour le calcul des dérivées d'une manière formellement exacte. D'ailleurs les complications majeures d'implémentation de tous les nouveaux aspects présentés dans ce chapitre ne sont pas du tout liées à la dérivation, mais plutôt à la façon de bien s'intégrer dans l'architecture existante du compilateur de langage de dimensionnement ou la redéfinition et reimplémentation des services de génération. Ce coût nous a conduit évidement au final à pouvoir définir et utiliser des variables vectorielles et des fonctions numériques décrites dans un langage externe, sans être concerné par leur dérivation.

Chapitre V

Dérivation Automatique appliquée aux méthodes numériques utilisées dans les modèles de dimensionnement

There are several reasons for preffering the... automatic differentation of Newton scenario instead of using implicit theorem*... A very important one is the user's inability or unwillingness to anlayze and modify the various iterative processes that may be going on in several places in a larger evaluation program.*

Andreas Griewank, Andrea Walther - 2008

Sommaire

Résumé

Dans la modélisation des dispositifs électriques, il faut souvent employer des méthodes numériques dans le cas où une modélisation purement analytique n'est pas suffisante. En vue de l'optimisation de ces dispositifs en utilisant des algorithmes de type SQP, les gradients de toute méthode numérique doivent être évalués. Dans ce chapitre, nous proposons des solutions permettant d'évaluer ces gradients pour des méthodes numériques couramment utilisées dans la modélisation des dispositifs électriques. Il s'agit notamment de la méthode de Newton-Raphson pour la résolution des systèmes d'équations implicites non linéaires, des méthodes d'intégration de fonctions à une ou plusieurs dimensions et des méthodes pour la résolution des systèmes d'équations différentielles ordinaires. L'intérêt principal est d'appliquer la Dérivation Automatique sur les algorithmes implémentant ces méthodes et d'analyser son impact. L'applicabilité de ces méthodes en Génie Electrique est ensuite illustrée sur des cas tests d'applications de dimensionnement de dispositifs.

V.1 Introduction : méthodes numériques souvent employées par l'ingénieur concepteur pour formuler les modèles de dimensionnement

Dans le chapitre III, nous avons proposé d'utiliser la Dérivation Automatique afin de dériver tout algorithme hétérogène numérique, en utilisant le générateur des fonctions externes. Il existe cependant certaines méthodes numériques couramment utilisées dans la modélisation des phénomènes en Génie Electrique, pour lesquelles nous nous sommes penché sur la meilleure façon d'exploiter la Dérivation Automatique en la combinant avec des propriétés mathématiques. De ces combinaisons dépend en général l'efficacité d'optimisation pour une grande partie des modèles de dimensionnement de dispositifs électriques.

Dans la première partie de ce chapitre, nous nous sommes intéressés aux algorithmes numériques complexes, contenant de boucles itératives, des branches conditionnelles, des fonctions récursives, etc., dont on ne connaît rien sur les principes mathématiques implémentées liant les entrées aux sorties. Ceci est typiquement le cas de calcul des dérivées d'un algorithme numérique pour lequel on ne désire pas dépenser beaucoup de temps à comprendre le programme l'implémentant. Dans cette logique, nous proposons un algorithme complexe implémentant l'intégration de fonctions, utile pour le calcul des forces dans les micro systèmes magnétiques. Afin de valider les résultats des dérivées fournis par la Dérivation Automatique, nous utilisons une référence pour le calcul de ces dérivées, obtenues en appliquant la dérivation à la main en s'appuyant sur les propriétés mathématiques de l'intégrale [39][46]. L'intégrale fait l'objet de l'utilisation du concept de fonctions de fonctions sous CADES et nous montrons aussi la façon de décrire son utilisation dans le langage *sml*.

Dans la deuxième partie de ce chapitre, nous utilisons la Dérivation Automatique en exploitant en même temps les propriétés mathématiques de la méthode numérique concernée. C'est le cas des méthodes de type Newton-Raphson pour la résolution des systèmes d'équations implicites non linéaires. Elle est appliquée à l'optimisation des modèles de dimensionnement formulés par des réseaux de réluctances. La Dérivation Automatique est employée sous diverses formes afin d'évaluer les gradients. Comme dans le cas de l'intégrale, il existe aussi une méthode de référence pour le calcul de ces gradients. Ce chapitre s'achève sur l'utilisation de la Dérivation Automatique dans un projet plus élaboré reposant sur la résolution de systèmes d'état en l'appliquant à la modélisation dynamique d'actionneurs électromécaniques. Le but est d'être en mesure au final d'optimiser des modèles dynamiques, par exemple sur des critères de temps de réponse. L'intention est de dériver deux familles d'algorithmes de résolution, i.e. Runge-Kutta et l'expansion en série de Taylor, la seule *référence* pour le calcul des gradients étant la méthode des différences finies.

V.2 La Dérivation Automatique des algorithmes d'intégration de fonctions

Dans ce paragraphe, nous nous intéressons à l'utilisation de la Dérivation Automatique dans le but d'évaluer les gradients des fonctions écrites au moyen d'un programme informatique implémentant le calcul des intégrales multidimensionnelles au sens illustré dans le tableau V.1 :

TABLE V.1 – Formulation mathématique du problème de dérivation de l'intégrale

Formule de l'intégrale	Le but - calcul du gradient
$I(u_i, l_i, P) = \int_{l_1}^{u_1} \cdots \int_{l_n}^{u_n} f(x_1, \cdots, x_n, P)\, dx_1 \cdots dx_n$	$\nabla I = \left[\left(\frac{\partial I}{\partial u_i}\right) \left(\frac{\partial I}{\partial l_i}\right) \left(\frac{\partial I}{\partial P_j}\right) \right]$
$i = 1..n$	$j = 1..p$
$I : \mathbb{R}^{2\cdot n+p} \to \mathbb{R}$	$\nabla I : \mathbb{R}^{2\cdot n+p} \to \mathbb{R}^{2\cdot n+p}$
$f : \mathbb{R}^{n+p} \to \mathbb{R}$	

où $f : \mathbb{R}^{n+p} \to \mathbb{R}$ représente l'intégrande et les u_i, l_i les bornes supérieures (*upper*) et respectivement inférieures (*lower*) d'intégration. $P \in \mathbb{R}^p$ représente un ensemble de paramètres non-intégrable (notamment des paramètres de dimensionnement).

V.2.1 Dérivation Automatique des algorithmes d'intégration

Les algorithmes d'intégration numérique utilisés pour le calcul de l'intégrale I, dans le tableau V.1, proviennent de la librairie écrite en C s'appelant Cuba [53]. Selon la dimension de l'intégrale, cette librairie propose l'algorithme Cuhre d'interpolation polynomiale pour le calcul de l'intégrale multidimensionnelle ($n \geqslant 2$) et la méthode de Simpson dans le cas d'une seule dimension ($n = 1$ dans le tableau V.1).

Le tableau V.2 caractérise le contenu de l'algorithme Cuhre d'intégration.

TABLE V.2 – Contenu de l'algorithme Cuhre d'intégration des fonctions

Nombre de lignes de code	≈1100
Nombre d'instructions `if-then-else`	≈30
Nombre d'instructions `for`	≈35
Nombre d'instructions `for` imbriquées	8*
Nombre d'instructions `while`	2
Nombre d'instructions `goto`	2
Nombre de fonctions récursives	1

* Dont 6 instructions répétitives sont imbriquées de niveau 2, 1 de niveau 3 et 1 de niveau 4.

On voit aisément que cet algorithme est très complexe, contenant la plupart des struc-

tures de contrôle proposés par le langage de programmation C (et tout langage de programmation en général). Sa complexité est augmentée par la présence des instructions répétitives imbriquées ou des sauts réalisés par l'instruction `goto`.

L'outil de Dérivation Automatique ADOL-C [51] est utilisé pour évaluer le gradient de l'intégrale (∇I) au sens mathématique donné dans le tableau V.1. Au sens informatique, le programme source de la librairie Cuba, écrit dans le langage C, est instrumenté avec cet outil de Dérivation Automatique [39].

V.2.1.1 Instrumentation avec ADOL-C

Afin d'employer la Dérivation Automatique, indépendamment de l'outil utilisé, sur un programme dont on ne connait pas la structure et les aspects numériques utilisés, il faut avant tout reconnaitre les variables d'entrées et les sorties de l'algorithme. Pour les algorithmes Cuhre et Simpson, ces variables sont les bornes d'intégration plus les paramètres et respectivement la variable associée à la valeur de l'intégrale. Cette exploration suffit pour utiliser un outil de transformation source-to-source. Pour un outil s'appuyant sur la technique de surcharge d'opérateurs, tel qu'ADOL-C, des aspects supplémentaires d'instrumentation doivent être considérés.

Comme nous l'avons vu au paragraphe II.2.3.2 la dérivation par la surcharge d'opérateurs d'ADOL-C utilise une trace de calcul à l'exécution de la fonction à dériver.

Le procédé d'instrumentation de l'algorithme d'intégration Cuhre ou Simpson a impliqué les changements suivants dans le programme original, aussi détaillés dans l'Annexe A.2 :

1. La conversion en C++. Certaines déclarations des fonctions, écrites initialement dans la syntaxe C, ont été converties sous une forme acceptable pour la syntaxe C++.
2. Le changement du type des variables. Toutes les variables `float/double` utilisées par les algorithmes de calcul de l'intégrale ont été changées en `adouble` [1](Les constantes peuvent faire exception).
3. La marque du début de la trace de calcul. Le début de la trace de calcul, donc du programme actif à dériver, a été déclaré en appelant la fonction spéciale d'ADOL-C `trace_on(short ID)`.
4. Définition des variables indépendantes. Les bornes supérieures et inférieures d'intégration $(u_1, l_1, \cdots, u_k, l_k)$ ainsi que l'ensemble de variables non-intégrables (P_i) ont été définies comme *variables indépendantes* en utilisant l'opérateur spécial, <<=, d'ADOL-C. Ces définitions se font juste après le début de la trace de calcul (réalisée auparavant).
5. L'appel de l'algorithme d'intégration. L'algorithme d'intégration modifié dans les étapes 1 et 2 a été appelé avec les variables indépendantes en entrée.

1. L'`adouble` est le type de données réel pour lequel l'outil ADOL-C surcharge toute opération élémentaire ou fonction mathématique de base.

6. Définition des variables dépendantes. Le résultat d'intégration (I) a été défini comme *variable dépendante* en utilisant l'opérateur spécial >>=.
7. La marque de la fin de la trace de calcul. Le final de la trace de calcul, donc du programme actif à dériver, a été déclaré en appelant la fonction spéciale d'ADOL-C `trace_off()`.

Ce processus d'instrumentation n'a pas engendré de complications majeures, sa durée de réalisation étant de quelques heures. La partie la plus coûteuse est représentée par les étapes 1 et 2 du processus total d'instrumentation. A ces étapes, nous avons réalisé certaines opérations qui ne sont pas forcement nécessaires, mais qui augmentent légèrement l'efficacité en mémoire et le temps d'exécution du programme instrumenté. Il s'agit de l'identification de toutes les constantes du programme et le fait de ne pas les déclarer comme variables actives (`adouble`).

V.2.1.2 Dérivation Automatique avec ADOL-C en mode direct

Nous rappelons que le mode direct scalaire de dérivation évalue efficacement les dérivées de toutes les sorties d'une fonction par rapport à une seule entrée. Généralement, le mode direct devient plus efficace que le mode inverse pour une fonction au nombre de sorties plus élevé que le nombre d'entrées. Ces aspects ont été détaillés dans le paragraphe II.2.2.4. Pour les algorithmes d'intégration, le mode direct de dérivation ne s'applique pas très efficacement puisqu'on s'intéresse à l'évaluation des dérivées partielles d'une fonction scalaire par rapport à toutes ses entrées. Cependant, nous considérons ici ce mode pour des raisons théoriques, dans le but d'analyser les performances des modes de dérivation de la Dérivation Automatique (notamment en termes de temps CPU d'exécution et de quantité de mémoire utilisée).

Le gradient des algorithmes d'intégration s'obtient seulement en mode direct vectoriel, pour un ensemble de valeurs d'entrée. Pour ceci, nous appliquons le pilote `fov_forward` pour la trace construite au paragraphe V.2.1.1, afin d'évaluer le gradient complet donné dans le tableau V.1.

V.2.1.3 Dérivation Automatique avec ADOL-C en mode inverse

Le mode inverse de dérivation évalue efficacement un seul gradient d'une fonction décrite au moyen d'un programme informatique. En général, ce mode de dérivation est plus efficace pour la dérivation des fonctions ayant plus d'entrées que de sorties. Dans le cas des algorithmes d'intégration de fonctions, ce mode de dérivation est mieux adapté que le mode direct.

Pour le calcul d'un seul gradient, le mode inverse scalaire suffit, contrairement au mode direct qui doit s'appliquer dans la variante vectorielle. Nous utilisons ici le pilote `fos_reverse` [2] d'ADOL-C.

2. Le terme `fos_reverse` vient de First Order Scalar differantiation in REVERSE mode.

V.2.2 Utilisation sous CADES

Les algorithmes d'intégration des fonctions Simpson et Cuhre sont entièrement écrits dans le langage C/C++, dans le but de les rendre dérivables par l'outil ADOL-C qui supporte ce langage de programmation. Dans ce paragraphe, nous sommes intéressés par l'utilisation de ces programmes sous CADES, qui est un environnement reposant principalement sur Java. Nous avons formalisé une solution afin de permettre d'intégrer des fonctions dans les modèles de dimensionnement définis sous CADES dans le langage *sml*. Cet aspect exploite la possibilité offerte par le langage *sml* de réaliser la modélisation semi-analytique en utilisant des fonctions de fonctions externes. L'idée est d'encapsuler le programme C des algorithmes d'intégration dans une couche Java d'une fonction de fonctions externe, suivant la stratégie proposée au paragraphe IV.6.3. Ensuite tout est intégré dans une archive Java, baptisée `Cuhre.jar`.

Le modèle de dimensionnement formalisé pour l'utilisation de l'intégrale, `Cuhre.jar`, dans la syntaxe *sml* est illustré sur la figure V.1.

```
sml:
import "Cuba.jar";//l'import de l'intégrale(fonction de fonctions);
f(x, y, z) = ...;  //fonction interne analytique;
I1 = Simpson(f(_, y, z), a0, b0);              //
I2 = Cuhre(f(_, _, z), a0, b0, a1, b1);        //;
I3 = Cuhre(f(_, _, _), a0, b0, a1, b1 a2, b2); //;
```

FIGURE V.1 – La modélisation semi-analytique en utilisant des fonctions de fonctions pour le calcul des intégrales en *sml*

Là, l'intégrande, f, est une fonction interne analytique et les intégrales simple, double et respectivement triple sont définies comme ci-dessous :
$I_1 = \int_{a_0}^{b_0} f(x,y,z)dx$, $I_2 = \int_{a_0}^{b_0}\int_{a_1}^{b_1} f(x,y,z)dxdy$, $I_3 = \int_{a_0}^{b_0}\int_{a_1}^{b_1}\int_{a_2}^{b_2} f(x,y,z)dxdydz$.

V.2.3 Application pour le dimensionnement des micro-actionneurs électromagnétiques

Dans ce paragraphe, nous proposons le dimensionnement d'un micro-actionneur électromagnétique par optimisation avec l'algorithme SQP basé sur le calcul de gradients. Ce micro-actionneur a été modélisé dans les travaux de thèse de H. L. Rakatoarison [75] et il a été aussi repris par B. Delinchant en [24].

Le micro-dispositif est composé d'une bobine qui crée un champ magnétique actionnant un aimant avec une force le long de l'axe de la bobine. La figure V.2a montre le schéma du micro-actionneur :

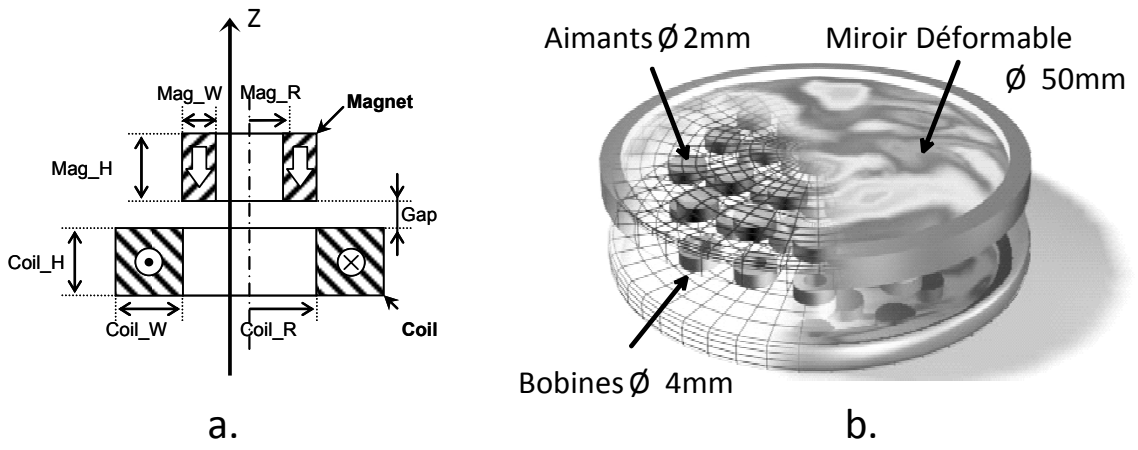


FIGURE V.2 – Le micro-actionneur électromagnétique ; a. La structure de l'ensemble bobine-aimant ; b. Miroir déformable actionné par une matrice d'ensembles bobine-aimant

L'actionneur est intégré dans un dispositif d'optique adaptative utilisé en astrophysique ou ophtalmologie [29]. Un miroir déformable, comme celui montré sur la figure V.2b, est actionné par une matrice d'aimants identiques dans le but de changer la surface de réflexion et d'équilibrer les turbulences atmosphériques. Chaque aimant est actionné séparément par une bobine positionnée sur la base du dispositif. L'intérêt est de dimensionner un seul ensemble aimant-bobine afin de contrôler la déformabilité du miroir.

V.2.3.1 Cahier des charges d'optimisation

TABLE V.3 – Le cahier des charges d'optimisation du micro-actionneur électromagnétique

Paramètre	Déscription	Unité	Valeur initiale	Nature	Contraint
Coil_W	Epaisseur bobine	mm	1	Entrée	fixé
Coil_H	Hauteur bobine	mm	1	Entrée	fixé
Coil_R	Rayon intérieur bobine	mm	1	Entrée	fixé
J	Densité courant	A/mm	100	Entrée	fixé
Mz	Magnétisation aimant	T	1	Entrée	fixé
Gap	Entrefer	mm	0.5	Entrée	fixé
Mag_R	Rayon intérieur aimant	mm	0.5	Entrée	$[0.001; 1]$
Mag_W	Epaisseur aimant	mm	0.5	Entrée	$[0.001; 1]$
Mag_ff	Facteur de forme($\frac{Mag\_W}{Mag\_H}$)	-	0.25	Entrée	$[0.01; 1]$
Mag_H	Hauteur aimant	mm	-	Sortie	libre
Mag_Force	Force sur l'aimant	mN	-	Sortie	égalité
Mag_vol	Volume aimant	mm^3	-	Sortie	$\min[1, 3]$

Le but de l'optimisation du micro-actionneur dans la figure V.2a, est d'atteindre une

force de 30 mN au niveau de l'aimant, afin de limiter la déformabilité du miroir avec un volume minimal de l'aimant. Le cahier des charges d'optimisation est défini dans le tableau V.3.

Le problème d'optimisation, a seulement trois degrés de liberté qui sont les dimensions de l'aimant. Le facteur de forme assure que l'aimant soit fabriqué avec la micro-technologie actuelle. Nous imposons donc que la hauteur de l'aimant soit supérieure à son épaisseur. Les domaines de variation des paramètres géométriques de l'aimant sont contraints d'une manière permettant d'assurer la validité de la géométrie du dispositif. La convergence de l'algorithme d'optimisation peut être gravement affectée si ces contraintes ne sont pas bien posées.

V.2.3.2 Modélisation du micro-actionneur

Nous utilisons la modélisation semi-analytique de CADES, présentée au chapitre III. La modélisation numérique du modèle de dimensionnement, correspondant à la structure donnée sur la figure V.2a, comprend l'utilisation de l'algorithme d'intégration pour le calcul de la force et du champ magnétique créé par la bobine. Le calcul de cette force, fait l'objet de l'application d'une intégrale double du champ crée par la bobine [24]. Pour le champ émis par la bobine nous considérons seulement la composante z, qui est calculée en appliquant la loi de Biot-Savard, par une formule faisant intervenir la densité du courant et une intégrale simple d'une fonction décrivant la géométrie du dispositif. Nous précisons que cette modélisation pourrait aussi se réduire à une seule intégrale triple, grâce aux propriétés des intégrales. Ces aspects sont détaillés dans l'Annexe C.1.

TABLE V.4 – Les grandeurs ou les phénomènes magnétiques pris en compte par la modélisation semi-analytique du micro-actionneur

Grandeur	Type de modélisation	Eléments de modélisation *sml*
Géométrie	analytique	fonction interne - l'intégrande du champ
H^z_{ext}	numérique	fonction de fonction - intégrale simple(Simpson)
σ_S	analytique	équations
$\sigma_S \cdot H^z_{ext}$	analytique	fonction interne - l'intégrande de la force
Force	numérique	fonction de fonction - intégrale double(Cuhre)

Pour la modélisation en *sml*, nous utilisons l'algorithme de Simpson pour le calcul de l'intégrale à une dimension et celui de Cuhre pour l'évaluation de l'intégrale à deux dimensions. Le but est de créer un modèle de dimensionnement en s'appuyant sur le principe formalisé au paragraphe V.2.2, de générer un composant de calcul *ICAr* avec le *Component Generator* et ensuite d'exploiter ce dernier dans le module *Component Optimizer* de CADES en utilisant un algorithme d'optimisation SQP en définissant le cahier des charges illustré sur le tableau V.3. Les éléments intervenant dans la modélisation du micro-actionneur sont présentés sur le tableau V.4.

V.2.3.3 Résultats

Dans ce paragraphe, nous appliquons une optimisation du modèle en considérant le cahier des charges du tableau V.3, en utilisant un algorithme SQP. Avant de présenter les résultats obtenus, nous allons conduire une étude préliminaire concernant la justesse et les performances du calcul des dérivées avec l'outil de Dérivation Automatique ADOL-C.

A. Validation des dérivées obtenues avec ADOL-C

Dans ce paragraphe, une étude comparative est menée entre les dérivées partielles de l'algorithme d'intégration de fonctions, obtenues avec ADOL-C et une approche de référence de dérivation manuelle qui utilise les propriétés mathématiques de l'intégrale. L'équation V.1 donne l'ensemble d'expressions exactes des dérivées partielles de l'intégrale par rapport aux bornes d'intégration, alors que l'équation V.2 calcule les dérivées partielles par rapport aux paramètres non-intégrables. Ces formules sont applicables en considérant les bornes d'intégration indépendantes par rapport aux paramètres [39][46].

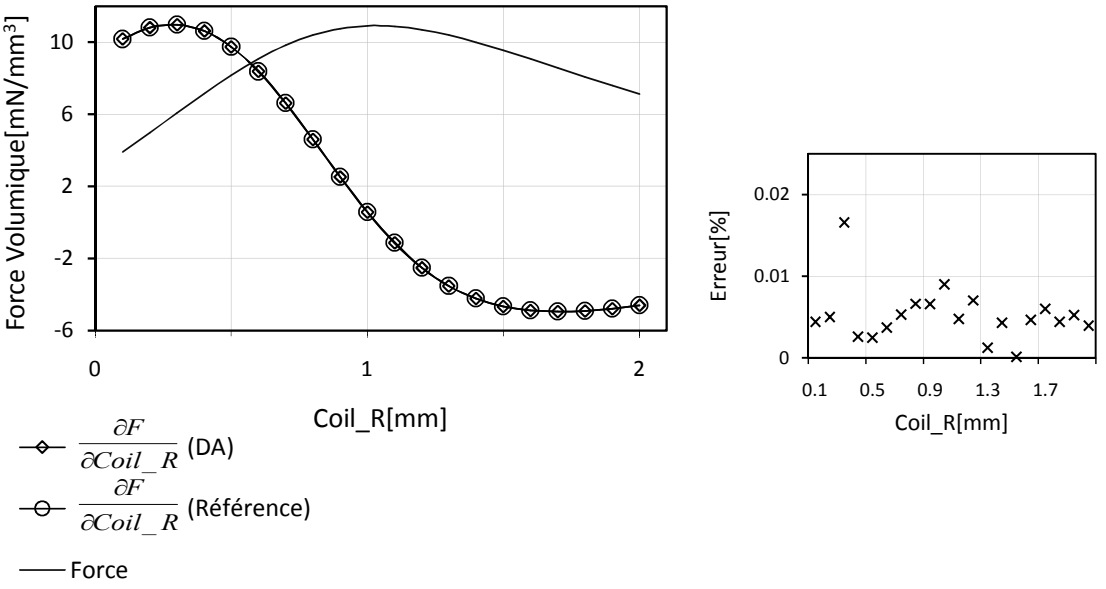
$$\frac{\partial I}{\partial b_i} = -1^k \int_{l_1}^{u_1} \cdots \int_{l_{i-1}}^{u_{i-1}} \int_{l_{i+1}}^{u_{i+1}} \cdots \int_{l_n}^{u_n} f(\cdots, x_{i-1}, b_i, x_{i+1}, \cdots, x_n, P) \cdots dx_{i-1} dx_{i+1} \cdots dx_n$$
$$\forall b_i \in \{l_i, u_i\}, \quad k = \begin{cases} 1, b_i = l_1 \\ 0, b_i = u_i \end{cases} \tag{V.1}$$

$$\frac{\partial I}{\partial P_i} = \int_{l_1}^{u_1} \cdots \int_{l_n}^{u_n} \frac{\partial f(x_1, \cdots, x_n, P)}{\partial P_i} dx_1 \cdots dx_n \quad \forall i = 1..p \tag{V.2}$$

Au cas où les bornes d'intégration dépendent de paramètres non-intégrables, la formule de calcul de dérivées de l'intégrale s'écrit comme en V.3

$$\frac{\partial I}{\partial P_i} = f(x_1, \cdots, x_n, P) \cdot \left(\sum_{i=1}^{n} \frac{\partial u_i}{\partial P_i} - \sum_{i=1}^{n} \frac{\partial l_i}{\partial P_i} \right) + \int_{l_1(P_i)}^{u_1(P_i)} \cdots \int_{l_n(P_i)}^{u_n(P_i)} \frac{\partial f}{\partial P_i} dx_1 \cdots dx_n \tag{V.3}$$

En faisant varier le rayon de la bobine de notre dispositif entre $[0\,mm, 2\,mm]$, sur la figure V.3, nous présentons une comparaison de résultats obtenus pour les dérivées partielles de la force volumique, entre l'approche de référence illustrée auparavant et celle de Dérivation Automatique.



a. b.

FIGURE V.3 – Comparaison des résultats de dérivation ; a. Variation de la force volumique avec le rayon de la bobine et ses dérivées ; b. L'erreur relative entre l'approche de référence et ADOL-C ; relation de calcul de l'erreur relative : $\epsilon_{r_i}[\%] = \left| \frac{x_i - x_{ref_i}}{\max\{|x_{ref_i}|\}} \right| \times 100$

Il est évident que la Dérivation Automatique produit les dérivées partielles avec une précision élevée, quasiment les mêmes que celles obtenues avec l'approche de référence. L'erreur maximale pour les dérivées tracées se situe à environ 0.017%. Une cause possible de cette erreur est le bruit numérique dû au fait qu'on travaille avec un ordinateur de précision finie. Elle est acceptable pour les algorithmes d'optimisation basés sur le calcul de gradients, cette marge n'affectant pas leur convergence.

B. Performances des modes de dérivation avec ADOL-C

Les performances de la Dérivation Automatique de l'intégrale, obtenues en utilisant l'outil ADOL-C en mode direct vectoriel (avec le pilote `fov_forward`) et le mode inverse scalaire (avec le pilote `fos_reverse`) sont illustrées sur le tableau V.5 en termes de temps CPU d'execution et de quantité de mémoire utilisée. Ces résultats sont obtenus pour la configuration initiale d'optimisation du micro-actionneur.

TABLE V.5 – Performances des modes de dérivation de l'intégrale

Temps CPU/Mémoire	Mode Direct	Mode Inverse	MD/MI
Préparation dérivation[ms]	297	2670	0.11
Calcul Dérivées[ms]	875	266	3.29
Préparation dérivation[$kbytes$]	2587	3111	0.83
Calcul Dérivées[$kbytes$]	2688	3509	0.77

Ces performances ont été obtenues avec l'outil *mpatrol* de profilage des programmes C/C++ ; voir `http://mpatrol.sourceforge.net/doc/`

Le tableau V.5 montre que le mode inverse s'avère 3 fois plus rapide que le mode direct, c'est qui est conforme avec la théorie des modes de dérivation pour cette application. Cependant, son temps de préparation est environ 9 fois plus élevé par rapport au mode direct. La quantité de mémoire requise pour la préparation du mode inverse est plus élevée. Or, l'allocation de mémoire consomme elle-même du temps d'exécution. Ceci engendre donc un temps plus grand de préparation.

Le mode direct de dérivation est plus lent, mais de 1.2 fois moins couteux en mémoire que le mode inverse. Il est cependant difficile de trouver un bon compromis entre la mémoire utilisée et le temps d'exécution. Si la mémoire disponible est suffisante, pour les problèmes de dimensionnement, en général c'est le critère temps d'exécution qui est plus intéressant.

C. Résultats d'optimisation

Dans ce paragraphe, l'algorithme SQP est utilisé pour optimiser le micro-actionneur. La figure V.4 montre la configuration initiale et finale du dispositif étudié. L'algorithme d'optimisation converge pour les deux approches de dérivation (ADOL-C et la référence donnée en équation V.1) en 8 itérations pour une précision de 10^{-4}. Ceci est normal, puisque nous avons montré que les valeurs des dérivées correspondantes à ces deux approches sont quasi-identiques.

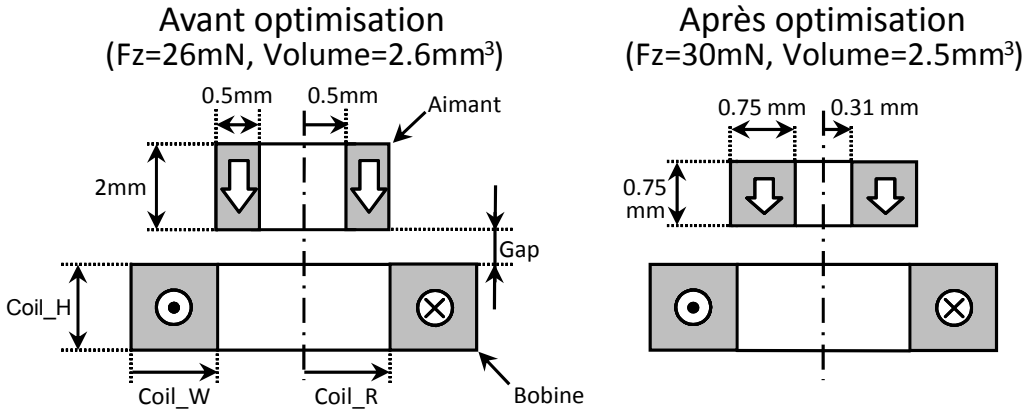
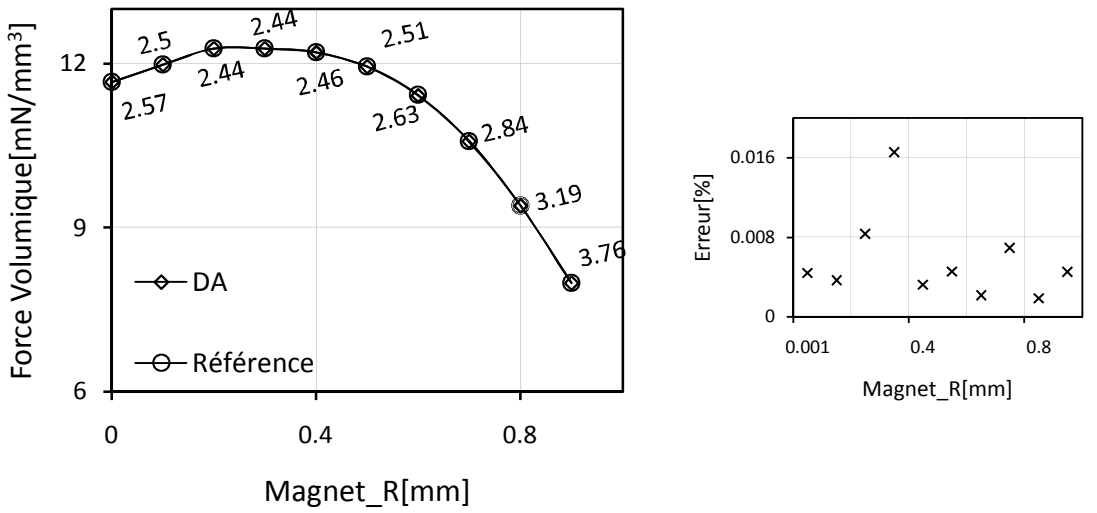


FIGURE V.4 – La configuration du micro-actionneur avant et après l'optimisation

Afin d'évaluer la fiabilité de la Dérivation Automatique de l'intégrale, nous proposons des optimisations successives pour le même cahier des charges en faisant varier le rayon de l'aimant dans la plage de valeurs $[0mm, 0.9mm]$. En dehors de cette plage, aucune solution d'optimisation n'a été trouvée pour les deux approches de dérivation. Sur la figure V.5 nous avons tracé la variation de la fonction objectif en fonction du rayon de l'aimant et les erreurs relatives entre les valeurs de la fonction objectif obtenues avec l'algorithme d'optimisation SQP pour les deux stratégies d'évaluation de dérivées.



a. b.

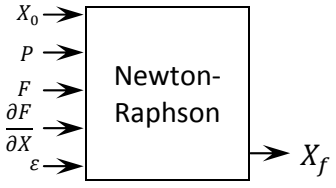
FIGURE V.5 – Fiabilité de la Dérivation Automatique ; a. Variation de la force volumique de l'aimant (fonction objectif) en fonction du rayon de l'aimant ; b. Erreur relative entre les optimums obtenus avec le calcul des dérivées par l'approche de référence et celle d'ADOL-C

Les optimisations successives donnent des valeurs pour la fonction objectif quasi-identiques pour les deux approches de dérivation. La convergence de l'algorithme d'optimisation est atteinte avec le même nombre d'itérations. L'erreur maximale entre les objectifs est de 0.16%, ce qui confirme de nouveau la précision élevée du calcul des dérivées avec la Dérivation Automatique.

V.3 La Dérivation Automatique des algorithmes de résolution des systèmes d'équations implicites

Nous proposons dans ce paragraphe des solutions permettant d'évaluer les gradients d'un modèle de dimensionnement caractérisé par un système d'équations implicites résolu par un algorithme de type Newton-Raphson, comme celui illustré dans le tableau V.6. En bref, un algorithme de résolution de type Newton-Raphson effectue des itérations à partir d'un ensemble de valeurs initiales pour les inconnues à rechercher, jusqu'à un ensemble de valeurs finales qui annulent le système considéré avec une précision spécifiée.

TABLE V.6 – Formulation mathématique du problème de dérivation de l'algorithme Newton-Raphson appliqué à la résolution des systèmes d'équations implicites non-linéaires

Le problème implicite	Le but - calcul des gradients
$F(X,P)=0$	
$\Updownarrow$	
$\begin{cases} f_1(x_1,\cdots,x_n,P)=0 \\ \vdots \\ f_n(x_1,\cdots,x_n,P)=0 \end{cases}$	
	$\nabla_i X_f = \left[\frac{\partial X_{f_i}}{\partial P}\right]$ $\forall i = 1..n$
$P \in \mathbb{R}^p$	
$F : \mathbb{R}^{n+p} \rightarrow \mathbb{R}^n$	$\nabla_i X_f : \mathbb{R}^p \rightarrow \mathbb{R}$

Ici F dénote le système d'équations implicites non-linéaires, P représente l'ensemble de paramètres non-implicites et ϵ représente la précision avec laquelle on souhaite obtenir la solution X_f qui annule le système implicite. Les gradients à évaluer sont composés de toutes les dérivées partielles de la solution obtenue, X_f, par rapport aux paramètres non-implicites, P.

V.3.1 Motivation pour l'évaluation des gradients des implicites

Il existe différentes façons de traiter un système implicite dans un problème d'optimisation. Nous discutons deux d'entre elles. Ces aspects ont été étudiés ces dernières années dans les travaux de C. Coutel [21], [20] et de B. du Peloux [71], [31].

V.3.1.1 Résolution par optimisation

Le système implicite du tableau V.6, intégré dans un modèle de dimensionnement, peut être résolu avec l'algorithme d'optimisation en parallèle avec un cahier de charges. Pour réaliser ceci, il est simplement nécessaire d'égaler chaque équation implicite avec les variables fictives c_i et ensuite de contraindre ces derniers à zéro [3]. Ce processus est illustré sur la figure V.6.

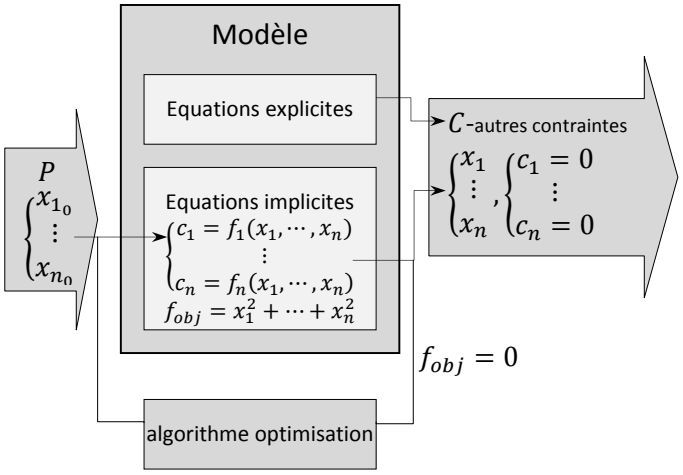


FIGURE V.6 – L'approche de résolution des systèmes implicites par l'algorithme d'optimisation

Cette stratégie est utilisée dans la plupart de travaux qui utilisent la Dérivation Automatique dans les réseaux électriques illustrés au Chpitre II et elle a été évaluée par C. Coutel en [21]. C'est une alternative très simple et très efficace à court terme lorsqu'on fait de l'optimisation, qui n'utilise pas le calcul des gradients de la solution du système par rapport aux variables de dimensionnement. Il suffit donc de dériver uniquement les fonctions du système implicite ($\frac{\partial f_i}{\partial P} \equiv \frac{\partial c_i}{\partial P}$), ce qui représente une tâche simple, étant donnée que dans la plupart des cas pour nos modèles de dimensionnement les f_i sont purement analytiques.

Cependant, cette alternative implique certains inconvénients importants. Premièrement, lors de l'introduction des variables fictives à annuler par l'algorithme d'optimisation, celles-ci n'ont aucun sens physique et le cahier des charges se dénature. Par ailleurs, plus le système d'équations implicites est grand, plus l'algorithme d'optimisation doit gérer de contraintes. Deuxièmement, la résolution des implicites est complètement indissociable de l'algorithme d'optimisation. Pour des systèmes d'équations implicites avec une convergence difficile, il est préférable d'utiliser des versions d'algorithmes Newton spécialisées ou de choisir à la limite d'autres algorithmes de résolution (sécante, bissection, etc.). Les algorithmes d'optimisation, qui ne sont pas conçus pour résoudre des systèmes implicites, peuvent facilement diverger dans ces cas.

3. Une stratégie plus convenable est de contraindre la somme quadratique de toutes ces variables fictives

V.3.1.2 Séparation des tâches

La seconde approche propose de séparer la résolution du système implicite de l'algorithme d'optimisation. Cette approche permet d'éliminer les inconvénients évoqués précédemment, et elle est illustrée sur la figure V.7.

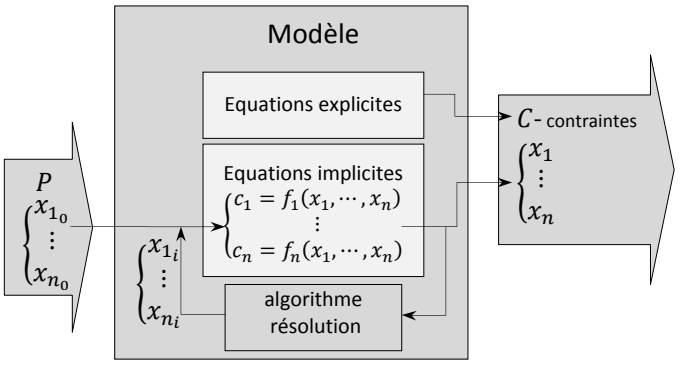


FIGURE V.7 – L'approche de résolution des systèmes implicites indépendamment de l'algorithme d'optimisation

Ici, il est donc nécessaire d'évaluer les gradients dans le sens montré au tableau V.6. Cela permet d'accélérer l'optimisation, le nombre d'itérations étant considérablement moins élevé pour des systèmes implicites de grande taille, que dans le cas d'un seul problème d'optimisation. Dans [31], les auteurs estiment ce gain de temps d'optimisation d'un facteur de 60 pour un système implicite de 10 équations avec une quarantaine des contraintes additionnelles.

Ainsi, nous retenons cette approche pour la suite. Ici nous allons nous focaliser sur l'analyse des techniques d'évaluation des dérivées via la Dérivation Automatique [37].

V.3.2 Dérivation Automatique des algorithmes de type Newton-Raphson

Le but de ce paragraphe est d'illustrer l'approche de Dérivation Automatique d'un algorithme Newton-Raphson de résolution des systèmes implicites. Afin de réaliser effectivement cette dérivation, nous faisons une étude mathématique préliminaire de convergence des dérivées. L'outil de Dérivation Automatique ADOL-C est proposé pour la dérivation de ce type d'algorithmes.

V.3.2.1 Aspects mathématiques de la Dérivation Automatique des algorithmes Newton

A. Description de l'algorithme de Newton-Raphson

L'algorithme de Newton-Raphson existe sous différentes formes pour des problèmes de résolution spécifiques [31] [71]. Dans ce paragraphe, nous nous intéressons à la version de base générale de cet algorithme [74]. Le principe de la méthode Newton-Raphson est de faire évoluer la récurrence :

$$X_{k+1} = X_k - \left[\frac{\partial F(X_k, P)}{\partial X}\right]^{-1} \cdot F(X_k, P) \tag{V.4}$$

à partir d'un ensemble de valeurs initiales, X_0, jusqu'à X_f telle que $F(X_f, P) = 0 \pm \epsilon$, où ϵ dénote la précision de résolution. Afin d'améliorer la compréhension de cet algorithme, ainsi que sa dérivation nous nous appuyons sur un exemple [4] à une seule dimension ($n = 1$) et à un seul paramètre ($p = 1$).

Exemple V.1
Soit $f : \mathbb{R}^2 \to \mathbb{R}$, définie implicitement par $f(x, p) = e^{p \cdot x} - a = 0$, qui admet une solution unique acceptable près de $x_f = 8.0472$ pour $a = 5.0$ et $p = 0.2$.

L'algorithme de Newton-Raphson [74] est appliqué pour une valeur initiale de la solution, $x_0 = 0.0$ et il converge dans neuf itérations vers une valeur finale, $x_f = 8.0471895622$, qui annule la fonction implicite avec une précision de $\epsilon = 10^{-13}$.

B. Propagation Automatique des dérivées

De manière générale, un outil de Dérivation Automatique est capable de propager les dérivées partielles de X_{k+1} dans la récurrence de Newton, comme le montre l'équation [5] V.5, construite en appliquant une dérivation à la main en s'appuyant sur les règles de dérivation d'un rapport et des fonctions composées :

$$\begin{aligned} X_{k+1} &= X_k - f_k/\dot{f}_k \\ X'_{k+1} &= X'_k - \frac{\dot{f}_k \cdot f'_k + {\dot{f}_k}^2 \cdot X'_k}{{\dot{f}_k}^2} + f_k \cdot \frac{\dot{f}'_k + \ddot{f}_k \cdot X'_k}{{\dot{f}_k}^2} \end{aligned} \tag{V.5}$$

où on adopte les notations matricielles suivantes à l'itération k de résolution :
$f_k = f(X_k, P)$
$X'_k = \left[\frac{\partial X_k}{\partial P}\right]$
$f'_k = \left[\frac{\partial f(X_k, P)}{\partial P}\right]$
$\dot{f}_k = \left[\frac{\partial f(X_k, P)}{\partial X}\right]$
$\dot{f}'_k = \left[\frac{\partial^2 f(X_k, P)}{\partial X \partial P}\right]$
$\ddot{f}_k = \left[\frac{\partial^2 f(X_k, P)}{\partial X^2}\right]$

Un outil de Dérivation Automatique propage les dérivées de la récurrence V.5 que si on fournit le programme de calcul de f et $\dot{f}$. Le programme de $\dot{f}$ assure la propagation des dérivées secondes $\frac{\partial^2 f(X_k, P)}{\partial X \partial P}$ et $\frac{\partial^2 f(X_k, P)}{\partial X^2}$.

C. Désactivation de la dérivée seconde

4. Cet exemple est réalisé en dehors de l'environnement logiciel CADES.
5. Les simplifications, à la rigueur, dans cette formule n'ont pas été fait en mode exprès, pour montrer ce que propage en réalité un outil de Dérivation Automatique.

La propagation des dérivées de l'équation V.5 implique le calcul des dérivées secondes, ce qui peut demander une grande quantité de mémoire et de temps de calcul. Pour un système implicite de taille élevée ayant beaucoup de paramètres, l'évaluation des dérivées d'ordre second peut augmenter le calcul total jusqu'à 70% [96].

On peut cependant ignorer complètement ce terme dans la propagation de dérivées. Cette stratégie on l'appelle *Désactivation de la dérivée seconde*. En effectuant les simplifications dans la récurrence de dérivées, on obtient :

$$X'_{k+1} = -\frac{f'_k}{\dot{f}_k} + f_k \cdot U_k^2 \tag{V.6}$$

où U_k^2 représente le terme complet dépendant des dérivées secondes. En effet, le produit $f_k \cdot U_k^2$ peut être en effet complètement ignoré, grâce à ses propriétés de convergence. Dans [50] et [52], les auteurs démontrent que le terme U_k^2 est uniformément borné autant que F converge à zéro. Ainsi, le produit $f_k \cdot U_k^2$ disparaît progressivement au fur et à mesure que l'algorithme de résolution converge. Avec ou sans ce terme, les dérivées de la solution convergent vers la même valeur qui est précisément exacte. Ces aspects sont illustrés sur les courbes de convergence des dérivées de la solution calculées en ignorant et en considérant ce terme, ainsi que sur sa courbe de convergence, tracées pour l'exemple V.1 :

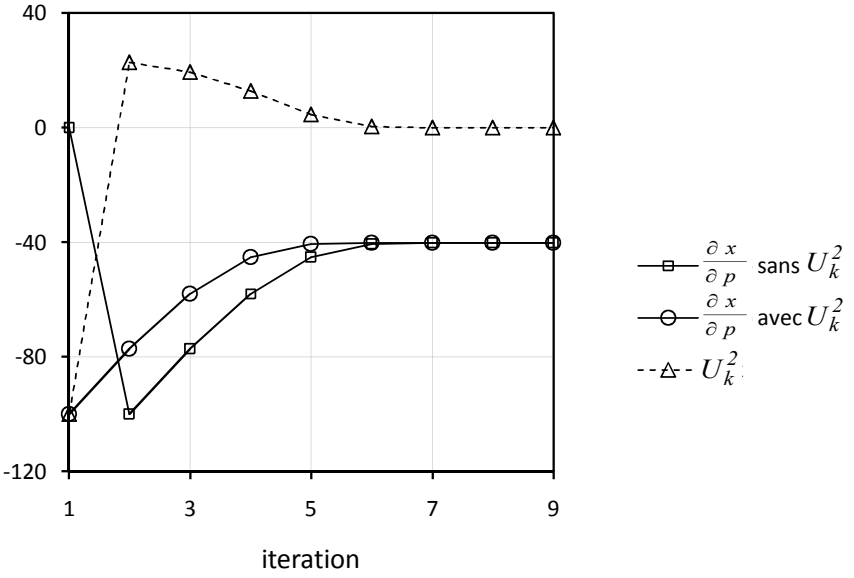


FIGURE V.8 – Convergence des dérivées partielles de la solution de la fonction $f(x,p) = e^{p \cdot x} - a = 0$ par rapport au paramètre p, obtenues en considérant ou non les dérivées secondes. La valeur de la dérivée partielle obtenue dans les deux cas : $\dfrac{\partial x}{\partial p} = -40.2359478109$

La désactivation des dérivées secondes est optionnelle, celle-ci n'influençant pas le résultat des dérivées. Aucun outil de Dérivation Automatique ne réalise automatiquement cette désactivation. C'est l'utilisateur qui en est responsable. Avec un outil de surcharge d'opérateurs, ceci se réalise en général en déclarant simplement une variable de type inactif

pour stocker les valeurs de f'_k. Par exemple, avec l'outil de dérivation ADOL-C, on peut simplement déclarer cette variable passive en la laissant de type `double` (non pas du type `adouble`).

Nous proposons une seconde possibilité pour désactiver la dérivation seconde. Celle-ci nous a permis d'ailleurs d'utiliser l'approche de dérivation de l'algorithme de Newton-Raphson dans l'environnement CADES. L'idée est de définir pour la fonction du système f une fonction externe ADOL-C (donc il faut aussi définir $\dot{f}$) et de l'appeler dans la boucle itérative de l'algorithme Newton-Raphson en effectuant aussi sa division avec son propre jacobien ($\dot{f}$) pour obtenir la tangente d'incrementation ($\Delta x = f/\dot{f}$). Grâce à cet aspect, ADOL-C ne propage pas les dérivées secondes puisqu'il ne connait aucun mécanisme pour prendre en compte les dérivées secondes d'une fonction externe.

V.3.2.2 Dérivation Automatique avec ADOL-C en mode direct ou inverse

Dans ce paragraphe, nous appliquons les modes de dérivation inverse et direct d'ADOL-C, afin d'évaluer les dérivées partielles du problème implicite du tableau V.6, en s'appuyant sur une trace de calcul instrumentée dans la même logique que pour l'intégrale (voir paragraphe V.2.1.1). ADOL-C offre quatre possibilités de réutilisation de cette trace de calcul, selon le nombre des paramètres ou variables implicites, dans le but d'évaluer les dérivées désirées. Elles sont illustrées dans le tableau V.7.

TABLE V.7 – Possibilités de réutilisation de la trace de calcul instrumentée avec ADOL-C pour l'évaluation des dérivées de la solution d'un système implicite de taille n par rapport à p paramètres

Scénario	Pilote - mode direct	Pilote - mode inverse
$p = 1,\ n = 1$	`fos_forward`	`fos_reverse`
$p = 1,\ n > 1$	`fos_forward`	`fov_reverse`
$p > 1,\ n = 1$	`fov_forward`	`fos_reverse`
$p > 1,\ n > 1$	`fov_forward`	`fov_reverse`

Rémarque : pour le mode inverse il est nécessaire d'appeler préalablement un pilote de dérivation en mode direct. En général on appelle `zos_forward(ID, n, p, k, ...)` pour $k = 1$ afin de préparer le mode inverse pour stocker toutes les variables intermédiaires.

Nous rappelons qu'en général, le mode inverse n'est pas désirable pour tout algorithme itératif donc aussi pour Newton-Raphson (voir paragraphe II.2.2.4).

V.3.3 Utilisation sous CADES

Nous nous focalisons ici, sur la modélisation d'un système d'équations implicites et sa résolution. Dans un modèle de dimensionnement les équations de ce système sont en général analytiques. Sur la figure V.9a nous illustrons cette modélisation, en adoptant la syntaxe du langage *sml*.

a.

```
sml:
n = constante; //taille du système implicite;
f1(x1,...,xn)=...;
...                //système implicite(scalaire);
fn(x1,...,xn)=...;
```

b.

```
sml:
import "NewtonRaphson.jar";//l'import de l'algorithme NR (fonction de fonctions);
x1 = NewtonRaphson(f1(_,_,...,_,_),
                   ...
                   fn(_,_,...,_,_),x01,...,x0n, eps, 1); //résolution(scalaire);
...
xn = NewtonRaphson(f1(_,_,...,_,_),
                   ...
                   fn(_,_,...,_,_),x01,...,x0n, eps, n); //résolution(scalaire);
```

c.

```
sml:
import "NewtonRaphson.jar";//l'import de l'algorithme NR (fonction de fonctions);
f(x[n]) = [f1(x[0],...,x[n-1]),...,fn(x[0],...,x[n-1])];
declare x0[n];
declare Xf[n];
Xf = NewtonRaphson(f(_),x0,eps); //résolution(vectorielle);
```

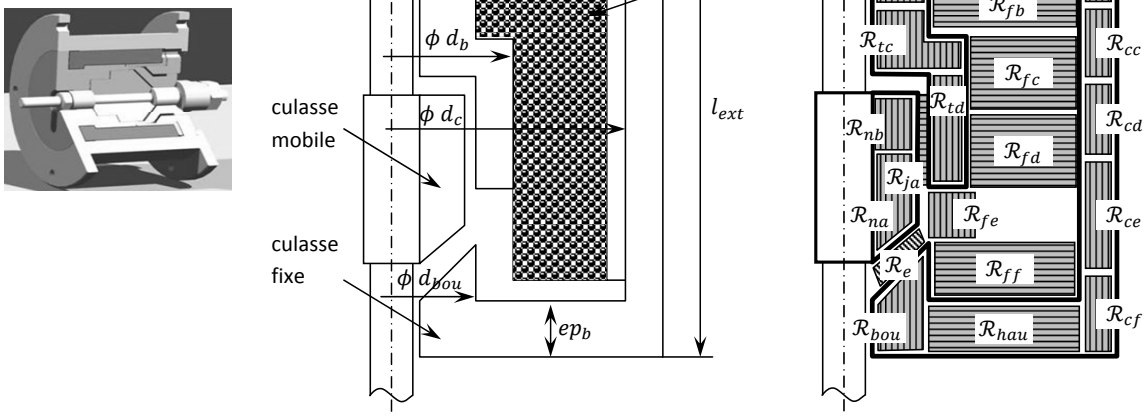
FIGURE V.9 – Modélisation sous *sml* des systèmes implicites et leur résolution ; a. Modélisation analytique du système implicite ; b. Appel à la résolution numérique scalaire ; c. Appel à la résolution numérique vectorielle

Nous adoptons cette modélisation analytique par des fonctions internes, afin de pouvoir utiliser une résolution numérique du système implicite en appelant des fonctions de fonctions (définies dans le langage Java). Nous rappelons que les fonctions de fonctions acceptent en argument des fonctions internes définies dans le modèle de dimensionnement. Le code *sml* associé à l'appel de la fonction est illustré sur la figure V.9b en variante scalaire et sur la figure V.9c en variante vectorielle. Notons que la formulation scalaire est plus difficile à implémenter en raison de la gestion de la persistance, discutée au paragraphe III.3.1. Nous préférons donc la modélisation vectorielle qui est plus facile et plus élégante à l'utilisation.

V.3.4 Application pour le dimensionnement des actionneurs électromagnétiques

Dans ce paragraphe, un actionneur linéaire cylindrique 3D, illustré sur la figure V.10, est proposé pour le dimensionnement par optimisation avec un algorithme SQP, basé sur le calcul de gradients. Cette structure, proposée par C. Chillet en [17], est composée d'une

culasse mobile actionnée par une force magnétique dans un entrefer de travail incliné. La bobine est positionnée à l'intérieur d'une culasse fixe qui assure la circulation du flux magnétique. La pièce mobile se déplace dans un entrefer de glissement vertical.



a. b. c.

FIGURE V.10 – Actionneur linéaire ; a. La vue 3D du dispositif fabriqué ; b. La demi-section transversale de l'actionneur, complétée avec les dimensions géométriques ; c. modélisation à base de réseaux de réluctances

Dans la phase opérationnelle du dispositif, la culasse mobile est attirée avec une force magnétique vers le bas le long de l'axe z et un ressort est utilisé vers le haut. Ni la dynamique, ni le dimensionnement du ressort ne sont pris en compte pour ce dispositif. Nous dimensionnons seulement sa structure à l'instant où la force apparaît dans l'entrefer de travail. Cette étude repose donc sur une modélisation statique de ce dispositif.

V.3.4.1 Cahier des charges d'optimisation

TABLE V.8 – Contenu du cahier des charges utilisé pour l'optimisation de l'actionneur linéaire

Type de contraint	Nature des variables	Quantité
valeur fixe	entrées	17
intervalle	entrées	8
valeur fixe	sorties	2
intervalle	sorties	27
valeur libre	sorties	88
fonction objectif	sortie	1

L'objectif de l'optimisation de l'actionneur linéaire est de trouver une masse minimale du dispositif pour une force magnétique, dans la position *ouverte* dans l'entrefer de travail

incliné, égale à la force résistante du ressort pour assurer l'attraction de la culasse mobile vers le noyau fixe. Sachant que dans cette position la force est la plus faible, cette supposition assure le déplacement de la culasse mobile vers le bas. Le contenu de ce cahier des charges est donné sur le tableau V.8

Ce cahier des charges comporte huit degrés des libertés selon huit paramètres d'entrées géométriques. Les entrées sont les distances géométriques données sur la figure V.10. En sortie, nous fixons la force dans l'entrefer de travail. Les contraintes d'intervalle sur les sorties concernent principalement les grandeurs autres que les dimensions géométriques, par exemple les inductions correspondantes à chaque réluctance du circuit magnétique de la culasse fixe et mobile (voir figure V.10c) afin d'éviter le passage en régime saturé du matériau ferromagnétique utilisé (FeCo) ($B < 3T$).

V.3.4.2 Modélisation sous *sml*

Le modèle mathématique de l'actionneur linéaire présenté sur la figure V.10 est formulé en utilisant les réseaux de réluctances. Le système implicite apparaît dans la définition des lois de Kirchhoff du circuit réluctant, les inconnues étant les flux de mailles. A partir de ces flux, on peut calculer l'énergie et la coénergie du système, en supposant qu'il existe un seul dégrée de liberté de mouvement. La dérivée de la coénergie par rapport à la position permet le calcul de la force dans l'entrefer du travail. Ces aspects sont détaillés dans l'Annexe C.2.4.

Le modèle mathématique de l'actionneur linéaire est construit seulement à partir d'équations analytiques et intègre une partie numérique de résolution du système d'équations implicites. Dans le tableau V.9, nous précisons les éléments de cette modélisation semi-analytique.

TABLE V.9 – Les grandeurs ou les phénomènes magnétiques pris en compte par la modélisation semi-analytique de l'actionneur linéaire

Grandeur ou phénomène	Type de modélisation	Eléments de modélisation *sml*
Réluctances	analytique	équations
Caractéristique ferromag. FeCo	analytique	fonction interne
Coénergie	analytique	équation
Force	analytique	équation
Système d'équations implicites	analytique	fonctions internes
Résolution du système implicite	numérique	fonction de fonctions

La définition des systèmes d'équations implicites ainsi que leur résolution repose sur la modélisation semi-analytique formalisée dans le paragraphe V.3.3. Ce système modélise les lois de Kirchhoff appliquées au circuit réluctant. Le système comporte sept équations implicites de mailles du circuit réluctant équivalent de l'actionneur (voir l'Annexe C.2).

V.3.4.3 Résultats

Dans ce paragraphe, nous proposons un scénario d'optimisations via SQP de l'actionneur linéaire modélisé au paragraphe V.3.4.2, en nous appuyant sur le cahier des charges du paragraphe V.3.4.1. Sachant que les itérations de l'algorithme SQP peuvent être gravement affectées par la précision des dérivées, nous utilisons ces optimisations pour valider la justesse des gradients obtenus avec la Dérivation Automatique en les comparant avec une approche de référence utilisant le théorème des fonctions implicites.

A. Le théorème des fonctions implicites

Pour évaluer les gradients de la solution d'un système d'équations implicites, la Dérivation Automatique n'est pas la seule possibilité. On peut aussi utiliser le théorème des fonctions implicites [9] qui permet obtenir ces gradients avec une bonne précision [21].

THÉORÈME V.2

Soit les ouverts $\mathcal{U} \subset \mathbb{R}^n$ *et* $\mathcal{V} \subset \mathbb{R}^n$, $H : \mathcal{U} \times \mathcal{V} \to \mathbb{R}^m$ *continue et différentiable,* $\bar{x} \in \mathcal{U}$, $\bar{y} \in \mathcal{V}$, $H(\bar{x}, \bar{y}) = 0$ *et* $\left[\frac{\partial H}{\partial y}\right]$ *inversible. Alors, si nécessaire,* $\mathcal{U}$ *et* $\mathcal{V}$ *peuvent être choisi pour représenter des voisinages plus petits de* $\bar{x}$ *et respectivement de* $\bar{y}$ *afin de garantir l'existence d'une fonction dérivable* $Y : \mathcal{U} \to \mathcal{V}$ *satisfaisant* $Y(\bar{x}) = \bar{y}$ *et* $\forall x \in \mathcal{U}$, $H(x, Y(x)) = 0$. *De plus, la fonction* $\tilde{Y} : \mathcal{U} \times \mathcal{U} \to \mathbb{R}^m$ *définie par*

$$\tilde{Y}(x,u) = Y(x) - \left[\frac{\partial H(x,Y(x))}{\partial y}\right]^{-1} \cdot H(u, Y(x)),$$

satisfait $\tilde{Y}(x,x) = Y(x)$ *et*

$$\frac{\partial Y(x,x)}{\partial u} = \frac{\partial Y(x)}{\partial u} = -\left[\frac{\partial H(x,Y(x))}{\partial y}\right]^{-1} \cdot \left[\frac{\partial H(x,Y(x))}{\partial x}\right]$$

En appliquant ce théorème au système implicite du tableau V.6, il vient que :

$$[\nabla_i X_f] = \left[\frac{\partial X_{f_i}}{\partial P}\right] = -\left[\frac{\partial f_i(X_f,P)}{\partial X}\right]^{-1} \cdot \left[\frac{\partial f_i(X_f,P)}{\partial P}\right] \tag{V.7}$$

où X_f dénote une solution du système implicite. Plus particulièrement, pour le système caractérisant les équations de mailles du circuit réluctant :

$$\begin{cases} f_1(\phi_1, \cdots, \phi_n, P_1, \cdots, P_p) = 0 \\ \vdots \\ f_n(\phi_1, \cdots, \phi_n, P_1, \cdots, P_p) = 0 \end{cases} \tag{V.8}$$

l'application de ce théorème s'écrit :

$$\begin{bmatrix} \frac{\partial \phi_{f_1}}{\partial P_1} \cdots \frac{\partial \phi_{f_1}}{\partial P_p} \\ \vdots \\ \frac{\partial \phi_{f_n}}{\partial P_1} \cdots \frac{\partial \phi_{f_n}}{\partial P_p} \end{bmatrix} = - \begin{bmatrix} \frac{\partial f_1(\phi_f,P)}{\partial \phi_1} \cdots \frac{\partial f_1(\phi_f,P)}{\partial \phi_n} \\ \vdots \\ \frac{\partial f_n(\phi_f,P)}{\partial \phi_1} \cdots \frac{\partial f_n(\phi_f,P)}{\partial \phi_n} \end{bmatrix}^{-1} \cdot \begin{bmatrix} \frac{\partial f_1(\phi_f,P)}{\partial P_1} \cdots \frac{\partial f_1(\phi_f,P)}{\partial P_p} \\ \vdots \\ \frac{\partial f_n(\phi_f,P)}{\partial P_1} \cdots \frac{\partial f_n(\phi_f,P)}{\partial P_p} \end{bmatrix} \tag{V.9}$$

pour l'ensemble $\phi_f = [\phi_{f_1}, \cdots, \phi_{f_n}]$ dénotant une solution du système implicite.

Ce théorème est appliqué dans un grand nombre d'applications scientifiques, souvent dans le contexte d'optimisation avec des algorithmes basés sur le calcul des gradients [21][31][9]. Il nécessite une résolution du système implicite, les matrices jacobienne de dérivées partielles du système par rapport aux paramètres implicites et non implicites, une inversion et une multiplication matricielle.

Il existe plusieurs stratégies d'évaluation de la formule V.7 du théorème des foncions implicites :

1. l'évaluation des matrices jacobienne de dérivées partielles en utilisant un outil de calcul formel de dérivées (si le système implicite est analytique). Ici, il est nécessaire avant tout de résoudre le système implicite, puis de réaliser une inversion et une multiplication matricielle.

2. l'évaluation des matrices jacobienne de dérivées partielles en utilisant un outil de Dérivation Automatique (pour des systèmes implicites analytiques ou algorithmiques). Ici, il est aussi nécessaire de trouver une solution du système, d'invertir et de multiplier les matrices de dérivées partielles.

3. l'utilisation du pilote `inverse_tensor_eval` d'ADOL-C [51]. Ce pilote applique directement le théorème des fonctions implicites en éliminant la tâche d'inversion et de multiplication de matrices. Il donne aussi la possibilité d'obtenir les dérivées d'ordre supérieur de la solution par rapport aux paramètres. L'utilisation de ce pilote nécessite aussi de trouver une résolution du système implicite (analytique ou algorithmique).

Les approches no. 2 et 3 ont l'avantage d'appliquer le théorème des fonctions implicites pour des systèmes implicites non seulement analytiques, mais aussi définis par des algorithmes contenant des instructions comme `for, while, if...then...else`, etc. Notons que l'outil ADOL-C offre une nouvelle fonctionnalité, autre que la dérivation du premier ordre. Cependant, sachant que le système implicite de l'actionneur est analytique, nous considérons par la suite l'approche qui applique le théorème des fonctions implicites par calcul formel (l'approche no. 1) comme référence de comparaison avec la Dérivation Automatique de l'algorithme de résolution.

B. Validation des dérivées obtenues avec ADOL-C

Sur la figure V.11, nous présentons les courbes de variation de la force dans l'entrefer de travail en faisant varier le diamètre extérieur (ϕd_{ext}) du dispositif entre $[60.0mm; 75.0mm]$.

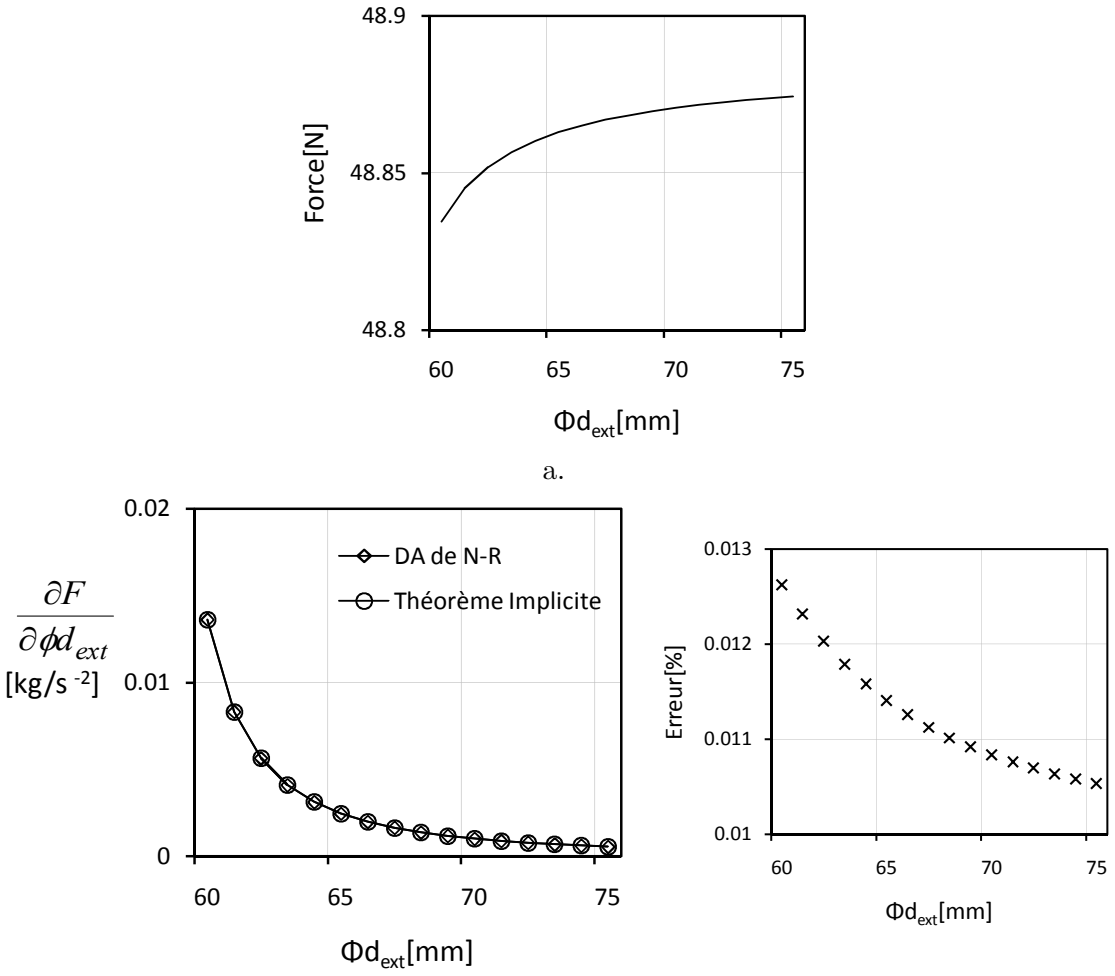


FIGURE V.11 – Validation des dérivées de la force dans l'entrefer ; a. Variation de la force avec le diamètre extérieur du dispositif ; b. Les dérivées partielles de la force par rapport au diamètre extérieur calculées par l'approche de Dérivation Automatique de l'algorithme Newton et en appliquant le théorème des fonctions implicites (la référence) ; c. Erreur relative entre les valeurs des dérivées ; relation de calcul : $\epsilon_{r_i}[\%] = \left| \frac{x_i - x_{ref_i}}{\max\{|x_{ref_i}|\}} \right| \times 100$

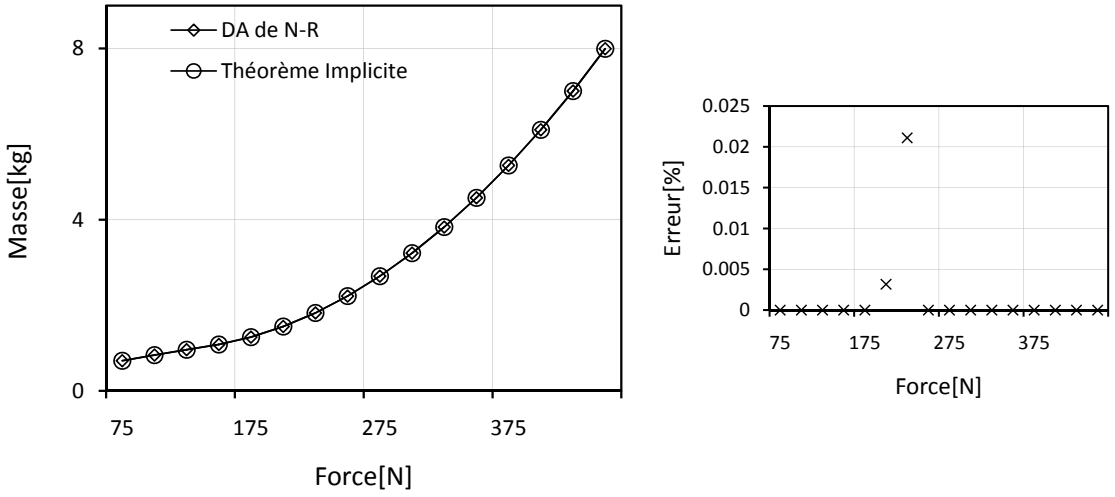
Les résultats des dérivées obtenus avec la Dérivation Automatique de l'algorithme de résolution des flux des mailles du circuit réluctant sont quasi-identiques avec celles fournies par le théorème des fonctions implicites, avec une erreur maximale près de 0.013%.

C. Les résultats d'optimisation

Ici nous allons comparer en optimisation, l'approche discutée au paragraphe V.3.2 par Dérivation Automatique avec celle de référence utilisant le théorème des fonctions implicites. Ces comparaisons se font sur la précision des dérivées et la rapidité des calculs. Nous faisons des optimisations successives dans le but de minimiser la masse du dispositif pour des valeurs de forces imposées dans l'entrefer de travail de l'actionneur, qui sont discrétisées sur la plage de valeurs $[75N; 400N]$, avec une valeur incrémentale de $25N$. Les autres spécifications du cahier des charges restent inchangées pour chaque valeur discrète

de la force.

Sur la figure V.12, nous illustrons la variation de la masse optimale du dispositif obtenue pour des valeurs fixes de la force dans l'entrefer. Les résultats de la masse optimale fournis par l'algorithme d'optimisation pour les deux approches de dérivation des flux de mailles du modèle de dimensionnement sont quasi-identiques, avec une erreur près de 0.02%.



a. b.

FIGURE V.12 – La variation de la masse optimale trouvée avec l'algorithme d'optimisation SQP en imposant la force dans l'entrefer dans une plage de valeurs discrète $[75.0N; 450.0N]$ $\Delta F = 25N$; a. Les optimums obtenus dans l'approche de Dérivation Automatique de l'algorithme Newton et en appliquant le théorème des fonctions implicites ; b. L'erreur relative entre les valeurs optimales

Sur la figure V.12, on retrouve que plus la force demandée est importante, plus la masse du dispositif augmente (variation proche d'une exponentielle).

TABLE V.10 – Comparaison des performances en temps CPU des optimisations SQP entre l'approche de calcul des gradients par la dérivation de l'algorithme Newton et l'approche de référence utilisant le théorème des fonctions implicites

Force[N]	75	100	150	200	250	300	350	400	450	M
TI[s]	9	8.66	6.75	5.92	3.7	3.3	3.6	3.6	2.6	5.2
DA de N-R[s]	46	63.5	52.6	55.2	30	28	31	32	36	41.7
TI/DA	5.1	7.3	7.8	9.33	8.2	8.4	8.8	8.9	13	8.4

M - moyenne, DA de N-R - les temps CPU d'optimisation en considérant l'approche de dérivation de l'algorithme de résolution, TI = les temps CPU d'optimisation en considérant le théorème des fonctions implicites ; Le temps CPU est mesuré en utilisant des outils de profiling Java sur une machine Professional Windows XP ™, Intel Core ® 2 CPU @ 2.13 GHz 2.13GHz, 3 Go Ram.

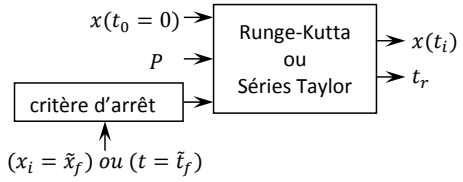
Dans le tableau V.10, nous donnons les temps CPU d'exécution de chaque optimisation correspondante aux valeurs imposées pour la force dans l'entrefer. Il en ressort que

l'approche par Dérivation Automatique de l'algorithme de résolution ralentit jusqu'à huit fois l'algorithme d'optimisation appliqué dans les mêmes conditions d'exécution que celle du modèle de dimensionnement contenant l'approche de dérivation de référence. Ceci est dû au fait que la Dérivation Automatique dérive toutes les itérations de résolution de l'algorithme Newton, alors que le théorème des fonctions implicites applique une formule de multiplication des deux matrices jacobienne (pas coûteuses) d'une fonction vectorielle (le système implicite) analytique, une fois le système résolu.

V.4 La Dérivation Automatique d'algorithmes d'intégration des systèmes d'équations différentielles

Nous proposons dans ce paragraphe l'utilisation de la Dérivation Automatique pour évaluer les gradients de la solution discrète des systèmes d'équations différentielles ordinaires (EDO) [6] continues, intégrés dans un domaine de variation de la variable indépendante, le temps, par des algorithmes numériques. Nous sommes concernées particulièrement par deux familles d'algorithmes d'intégration, i.e. Runge-Kutta (RK) et le développement en série de Taylor (ST). En principe, les résultats de ces deux types d'algorithmes sont identiques : à partir d'un ensemble de valeurs initiales, les algorithmes numériques calculent la solution en faisant évoluer le temps jusqu'à atteindre un critère d'arrêt. Ceci est un type particulier d'intégration s'appelant *Problème aux valeurs initiales* et il fait exclusivement l'objet de cette étude. Dans le tableau V.11, nous posons la problématique de ce paragraphe :

TABLE V.11 – Formulation mathématique du problème de dérivation des algorithmes d'intégration d'équations différentielles ordinaires (EDO)

Le problème d'EDO	Le but - calcul des gradients
$\dot{x} = f(x, P)$	$\nabla_i x = \left[\left[\frac{\partial x_i(t_r)}{\partial x_0}\right]\left[\frac{\partial x_i(t_r)}{\partial P}\right]\left[\frac{\partial x_i(t_r)}{\partial \tilde{x}_f}, \frac{\partial x_i(t_r)}{\partial \tilde{t}_f}\right]\right]$
ou	
$\dot{x} = f(x, t, P)$	$\nabla t_r = \left[\left[\frac{\partial t_r}{\partial x_0}\right]\left[\frac{\partial t_r}{\partial P}\right]\left[\frac{\partial t_r}{\partial \tilde{x}_f}, \frac{\partial t_r}{\partial \tilde{t}_f}\right]\right]$
	
$\dot{x} = \frac{dx}{dt}$	$\nabla_i x : \mathbb{R}^{n+p+2} \to \mathbb{R}$
$P \in \mathbb{R}^p$	$\nabla t_r : \mathbb{R}^{n+p+2} \to \mathbb{R}$
$f : \mathbb{R}^{n+p} \to \mathbb{R}^n$	$\forall i = 1..n$

où :

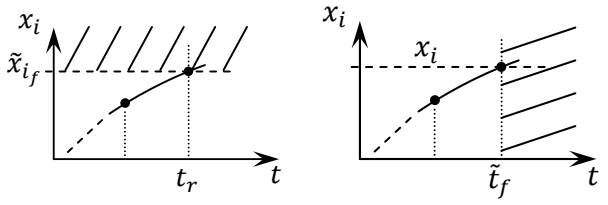
- $x \in \mathbb{R}^n$ - l'ensemble de variables d'état.
- f - la fonction modélisant le système d'état.
- $P \in \mathbb{R}^p$ - l'ensemble de paramètres de dimensionnement.

Dans le tableau V.11, nous introduisons deux formes principales de représentation de la fonction f . La première est une représentation de systèmes autonomes, où le temps n'apparaît pas explicitement dans la formule du système et la deuxième est une représentation non autonome. Nous écrivons intentionnellement le système d'état sous ces deux formes,

6. Nous utilisons aussi dans ce paragraphe le terme de *systèmes dynamiques* ou *système d'état* associé aux équations différentielles.

car comme on le verra tout au long de ce paragraphe, leur Dérivation Automatique est différente.

Les critères d'arrêt notifient l'algorithme d'intégration (résolution) qu'une valeur d'état, $\tilde{x}_f$ a été atteinte, ou un instant de temps, $\tilde{t}_f$ défini initialement a été dépassé. Ces deux critères d'arrêt sont illustrés graphiquement sur la figure V.13 :



a. b.

FIGURE V.13 – Les critères d'arrêt des algorithmes de résolution des systèmes d'état ; a. Le critère d'arrêt *état final* prescrit ; b. Le critère d'arrêt *temps final* prescrit

Grâce à ces considérations, nous serons capables de définir des contraintes en optimisation pour :

- l'état initial - x_0.
- les paramètres de dimensionnement - P.
- les paramètres du critère d'arrêt - $\tilde{x}_f$, $\tilde{t}_f$.
- l'état final (la réponse du système dynamique) - x_f.
- le temps de réponse - t_r qui correspond au moment d'atteinte de $\tilde{x}_f$ par x_i.

Notons que la variable de *dimensionnement*, t_r - le temps de réponse du système d'état, est utile en optimisation pour trouver un compromis entre les performances *classiques* d'un dispositif, e.g. rendement, coût, masse, etc. et la rapidité de la mise en œuvre d'un événement. Ainsi, son gradient doit être évalué et nous proposons des solutions dans ce sens par la suite.

V.4.1 Les algorithmes numériques d'intégration d'équations différentielles ordinaires

Les algorithmes d'intégration sont obligatoirement utilisés pour la résolution de la plupart des systèmes d'équations différentielles non linéaires. En général, ces systèmes n'admettent pas de solution analytique et pour cela, ces algorithmes numériques sont indispensables dans la plupart des cas. Ils fournissent l'évolution de l'état du système aux valeurs discrètes, et non pas continues, d'instants temporels. Ces valeurs discrètes peuvent être échantillonnées de façon fixe ou variable (pas de calcul adaptatif). Le pas de calcul variable peut être, par exemple trouvé en utilisant un schéma d'intégration, comme ceux proposés en [73]. Nous nous sommes particulièrement intéressés à ces schémas d'intégration, puisqu'ils sont capables d'évaluer un pas maximal entre deux états successifs tout en gardant une précision donnée. Ainsi, le nombre total d'appels de la fonction décrivant le système peut se réduire considérablement, en ayant une très bonne précision de calcul.

La plupart des résolutions de systèmes d'équations différentielles implémentées dans des programmes informatiques, séparent l'algorithme d'intégration en deux entités distinctes, i.e. le schéma du pas variable et l'intégrateur (le schéma d'intégration). Ces deux entités utilisent les valeurs des dérivées par rapport au temps du système d'équations différentielles. L'idée générale de ces aspects est illustrée sur la figure V.14 :

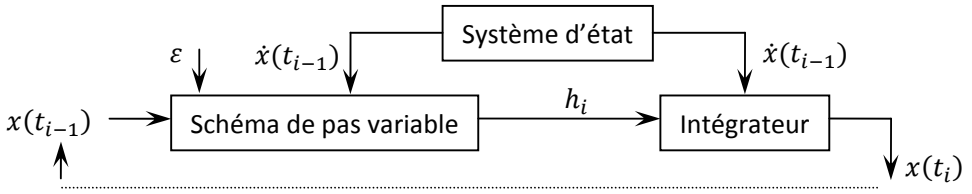


FIGURE V.14 – L'architecture d'un algorithme numérique d'intégration d'équations différentielles

Dans ce paragraphe nous nous appuyons sur cette stratégie d'implémentation de la résolution et de la dérivation des systèmes dynamiques.

V.4.1.1 Le développement en série de Taylor

Le développement en série de Taylor permet de calculer l'évolution de la solution d'un système d'état de $x(t_i)$ à $x(t_{i+1})$ sur un intervalle de temps $[t_i, t_{i+1}]$, en utilisant la somme finie tronquée selon l'équation V.10, à condition que le système soit suffisamment dérivable jusqu'à l'ordre $k \geq 1$:

$$x(t_{i+1}) = x(t_i) + \bar{x}_1 \cdot h_i + \bar{x}_2 \cdot h_i^2 + \cdots + \bar{x}_k \cdot h_i^k + \mathcal{O}(h_i^{k+1}) \tag{V.10}$$

où $\bar{x}_i = \frac{1}{i!} \cdot \frac{d^i x}{dt^i}$ est le i-ème coefficient de Taylor et $\mathcal{O}$ est le terme de l'erreur, qui dans le cas de cette méthode d'intégration est d'ordre $k+1$, k dénotant le dégrée maximal de développement de la série des puissances.

Cette méthode d'intégration est peu utilisée du fait que les dérivées d'ordre supérieur du système d'équations différentielles sont difficile d'évaluer. Une cause plus importante est due au fait que les schémas de pas variable associés à cette méthode ont tendance de réduire excessivement le pas d'intégration dans les régions où le système est raide. Cependant, nous nous y intéressons dans notre étude, étant donné que la Dérivation Automatique permet l'évaluation efficace des coefficients de Taylor, qui dépendent des dérivées d'ordre supérieur des fonctions du système d'état par rapport à la variable indépendante (le temps).

A. L'évaluation des coefficients de Taylor des systèmes d'état autonomes avec ADOL-C

Nous considérons l'outil ADOL-C, en l'utilisant cette fois, non pas pour l'évaluation des dérivées de premier ordre, mais pour l'évaluation automatique des coefficients de Taylor. Celle-ci est une fonctionnalité additionnelle de l'outil ADOL-C, qui justifie à nouveau le choix d'ADOL-C dans nos travaux, en s'appuyant sur les critères évoqués dans le tableau

IV.8. Afin d'appliquer effectivement cet aspect avec ADOL-C, nous supposons que le système, d'équations différentielles, f, dans la variante autonome définie dans le tableau V.11, est entièrement instrumenté avec cet outil. La procédure d'instrumentation, illustrée sur la figure V.15, doit considérer l'état (x) qui est vectoriel, comme variable active indépendante et sa dérivée $(\frac{dx}{dt})$ comme dépendante.

C++:

```
trace_on(ID);
  x <<= ...;//∀x, activation des variables d'état (indépendantes);
  xp  = f(x,P) ; // le système d'état
  xp>>=... ;// activation des dérivées d'état (dépendantes);
trace_off();
```

FIGURE V.15 – Programme C++ du système d'état autonome instrumenté avec l'outil ADOL-C ; variables indépendantes : $x \in \mathbb{R}^n$; variables dépendantes : $\dot{x} \in \mathbb{R}^n$

Grâce à cette stratégie, ADOL-C est capable de propager efficacement les coefficients de Taylor en s'appuyant sur la propriété V.11 des équations différentielles ordinaires :

$$\bar{x}_{i+1} = \frac{1}{1+i} \cdot \bar{f}_i \tag{V.11}$$

où $\bar{f}_i = \frac{1}{i!} \cdot \frac{d^i f}{dt^i}$ est le i-ème coefficient Taylor de la fonction décrivant le système d'état. Le calcul des $\bar{f}_i$ se réalise récursivement en utilisant la propriété de l'équation V.11 et en propageant les dérivées d'ordre supérieur du système autonome instrumenté auparavant. Ceci est présenté dans les équations du tableau V.12, en appliquant les règles de dérivation des fonctions composées à partir d'une variable indépendante x_0 :

TABLE V.12 – Propagation des dérivées d'ordre supérieur en ADOL-C en utilisant l'arithmétique de Taylor, applicable aux systèmes d'état autonomes

$\bar{f}_0$	$=$	$f(x_0)$	$\Rightarrow$	$\bar{x}_1$
$\bar{f}_1$	$=$	$f'(x_0) \cdot \bar{x}_1$	$\Rightarrow$	$\bar{x}_2$
$\bar{f}_2$	$=$	$\frac{1}{2} f''(x_0) \cdot \bar{x}_1 \cdot \bar{x}_1 + f'(x_0) \cdot \bar{x}_2$	$\Rightarrow$	$\bar{x}_3$
$\bar{f}_3$	$=$	$\frac{1}{6} f'''(x_0) \cdot \bar{x}_1 \cdot \bar{x}_1 \cdot \bar{x}_1 + f''(x_0) \cdot \bar{x}_1 \cdot \bar{x}_2 + f'(x_0) \cdot \bar{x}_3$	$\Rightarrow$	$\bar{x}_4$
$\dots$	$\dots$	$\dots$	$\dots$	$\dots$

où $f' = \frac{df}{dx}$ peut s'évaluer directement pour la trace de calcul définie sur la figure V.15 en mode direct ou inverse.

Cette récurrence est propagée automatiquement par ADOL-C en utilisant le pilote `forode`[7]. Ce pilote utilise k dérivations d'ordre supérieur en mode direct scalaire, en utilisant le pilote `hos_forward`[8], où k représente l'ordre maximal des coefficients Taylor. Le mode

7. `forode` est l'acronyme de FORward mode differentiation for Ordinary Differential Equations.
8. `hos_forward` est l'acronyme de High Order Scalar differentiation in FORWARD mode.

direct scalaire de dérivation est souhaitable, car la dérivation du système f se réalise en fonction d'une seule variable indépendante (le temps) pour plusieurs variables dépendantes (les variables d'état). Même si le temps n'apparaît pas explicitement parmi les variables indépendantes de la trace de calcul, cette dérivation est possible en connaissant la tangente de départ $\dot{x} = \bar{f}_0$.

Au moment de notre intervention dans ce travail, le pilote `forode`, responsable avec l'évaluation des coefficients Taylor, supportait seulement les systèmes *autonomes*. Autrement dit, il supportait des traces de calcul où le nombre de variables indépendantes était égal au nombre de variables dépendantes, égaux à la taille du système différentiel.

B. L'évaluation des coefficients des Taylor des systèmes d'état non autonomes avec ADOL-C

Le système d'équations différentielles ordinaires non autonomes représente la formulation la plus générale d'un système d'état non linéaire. La forme autonome n'est qu'une variante particulière. La récursivité du tableau V.12 applicable seulement aux systèmes autonomes est différente dans la variante non autonome :

TABLE V.13 – Propagation des dérivées d'ordre supérieur en ADOL-C en utilisant l'arithmétique de Taylor, applicable aux systèmes d'état non autonomes

$\bar{f}_0$	$=$	$f(x_0)$	$\Rightarrow$	$\bar{x}_1$
$\bar{f}_1$	$=$	$f'(x_0) \cdot \bar{x}_1 + \frac{df(x_0)}{dt}$	$\Rightarrow$	$\bar{x}_2$
$\bar{f}_2$	$=$	$\frac{1}{2} f''(x_0) \cdot \bar{x}_1 \cdot \bar{x}_1 + \frac{d^2 f(x_0)}{dxdt} \cdot \bar{x}_1 + f'(x_0) \cdot \bar{x}_2 + \frac{1}{2} \frac{d^2 f(x_0)}{dt^2}$	$\Rightarrow$	$\bar{x}_3$
$\dots$	$\dots$	$\dots$	$\dots$	$\dots$

Pour le cas des équations non autonomes, étant donné que le temps apparaît explicitement dans le système d'état, il est nécessaire de prendre en compte dans la propagation des coefficients de Taylor, sa dérivée directe par rapport au temps, i.e. $df(x_0)/dt$ en l'ajoutant à sa dérivée composée $f'(x_0) \cdot \bar{x}_1$. C'est ce que nous réalisons en effet le tableau V.13.

Ainsi nous enrichissons le pilote `forode` en ADOL-C. Dans la trace de calcul définie sur la figure V.15 nous introduisons le temps parmi les variables indépendantes, afin d'être en mesure de propager k fois les dérivées d'ordre supérieur de $\dfrac{df}{dt}$ avec le mode de dérivation `hos_forward`. Ainsi, la trace contient n variables dépendantes et $n+1$ variables indépendantes.

C. Les schémas de pas variable

Dans ce paragraphe, nous utilisons les schémas de pas variable proposés en [6], afin d'évaluer un pas maximal h_i avec une tolérance, ϵ, donnée. Ce pas est employé ensuite pour faire évoluer la solution des systèmes dynamiques en utilisant le développement en série de Taylor comme le montre l'équation V.10. Parmi les propositions faites dans [6], nous considérons deux schémas de pas variable, dont un applique la formule simple de l'équation V.12 :

$$h_i = \left(\frac{\epsilon}{\|\bar{x}_k\|_\infty} \right)^{1/k} \tag{V.12}$$

où $\|\bar{x}_k\|_\infty$ représente la norme infinie des toutes les coefficients Taylor de l'ordre k (maximal) du système, ce qui revient au coefficient maximal [9].

Un deuxième schéma, plus élaboré, utilise une équation implicite (voir l'équation V.13) et un critère de rejet d'un pas trop élevé, évalué avec une tolérance relative (ϵ_r) et une tolérance absolue (ϵ_a).

$$h_i^{k-1} \cdot (\|\bar{x}_{k-1}\|_\infty + h_i \cdot k \cdot \|\bar{x}_k\|_\infty) = \epsilon \tag{V.13}$$

où $\epsilon = \mathsf{min}\{\epsilon_r \cdot \mathsf{max}\{\|\bar{x}_0, \bar{x}_1\|_\infty\}, \epsilon_a\}$. Le critère de rejet d'un pas trop grand pour la tolérance ϵ se formule selon V.14 :

$$\mathsf{si} \;\; \|\dot{\bar{x}}_{i+1}, f(t_{i+1}, x_{i+1})\|_\infty > \epsilon \;\; \mathsf{alors} \;\; \tilde{h}_i = facr \cdot h_i \tag{V.14}$$

où $\dot{\bar{x}} = \sum_{k=1}^{n} k \cdot \bar{x}_k \cdot h_i^{k-1}$ est l'expansion en séries Taylor appliquée pour l'évaluation de $\dot{x}$ et $facr$ est un facteur de secours, usuellement égal à 0.8.

V.4.1.2 Dérivation du développement en série de Taylor

Le but de ce paragraphe est de propager les dérivées partielles du développement de Taylor, afin d'obtenir les blocs vectoriels $\left[\frac{\partial x_i(t_r)}{\partial x_0}\right]$ et $\left[\frac{\partial x_i(t_r)}{\partial P}\right]$ des gradients du tableau V.11.

A. Dérivées partielles de la réponse du système par rapport aux valeurs initiales

Dans l'équation V.15, nous proposons une récurrence permettant de propager les dérivées partielles de la réponse (finale) du système par rapport aux valeurs initiales (x_0). Cette récurrence est obtenue en employant à la main les règles de dérivation de fonctions composées, appliqué au développement en série de Taylor :
$x(t_{i+1}) = x(t_i) + \bar{x}_1 \cdot h_i + \bar{x}_2 \cdot h_i^2 + \cdots + \bar{x}_k \cdot h_i^k$
pour l'évaluation des dérivées partielles de la réponse :

$$\frac{\partial x(t_{i+1})}{\partial x_0} = \frac{\partial x(t_i)}{\partial x_0} + \sum_{j=1}^{k} \left(\frac{\partial \bar{x}_j}{\partial \bar{x}_0} \cdot \frac{\partial x(t_i)}{\partial x_0} \cdot h_i^j + \bar{x}_j \cdot j \cdot h_i^{j-1} \cdot \frac{\partial h_i}{\partial \bar{x}_0} \cdot \frac{\partial x(t_i)}{\partial x_0} \right) \tag{V.15}$$

à partir de x_0 jusqu'à $x(t_r)$, sachant que $\bar{x}_0 \equiv x(t_i)$. La difficulté principale dans l'équation V.15 est l'évaluation des dérivées partielles $\frac{\partial \bar{x}_j}{\partial \bar{x}_0}$ et les dérivées correspondantes au pas de calcul $\frac{\partial h_i}{\partial \bar{x}_0}$. Heureusement, il existe sous ADOL-C un pilote permettant d'évaluer les déri-

9. $\|\bar{x}_k\|_\infty = \mathsf{max}\{\bar{x}_{i_k}\}, \; i = 1..n$

vées partielles des coefficients de Taylor, s'appelant `accode` [10]. Cette fonction s'appelle suite à une dérivation d'ordre supérieur en mode inverse, en utilisant la fonction `hov_reverse` [11] de la trace de calcul instrumentée dans la figure V.15. Le pilot `hov_reverse` évalue la famille de matrices carrées $A[n][n][k]$ des dérivées partielles des coefficients Taylor d'ordre k de chaque fonction du système d'état $(\bar{f}_k)$ par rapport au coefficient Taylor d'ordre 0 de chaque état $(\bar{x}_0)$:

$$A_k = U \cdot \frac{\partial \bar{f}_k}{\partial \bar{x}_0} \tag{V.16}$$

où $A_k = A[\cdot][\cdot][k] = \dfrac{\partial \bar{f}_k}{\partial \bar{x}_0} \in \mathbb{R}^{n \times n}$ et U dénote la matrice unité d'ordre n, car il s'agit d'une dérivation vectorielle.

En connaissant la matrice des dérivées partielles A_k, la routine `accode` applique une version spéciale de la règle de dérivation des fonctions composées pour l'évaluation de la famille des matrices carrées $B[n][n][k]$ des dérivées partielles de $\bar{x}_k$ c'est-à-dire des coefficients de Taylor d'ordre k de chaque état, par rapport aux coefficients d'ordre 0, en utilisant la récurrence V.17.

$$B_k = \frac{1}{1+k} \cdot \left(A_k + \sum_{j=1}^{k} (A_{j-1} \cdot B_{k-j}) \right) \tag{V.17}$$

où $B_k = B[\cdot][\cdot][k] = \dfrac{\partial \bar{x}_k}{\partial \bar{x}_0} \in \mathbb{R}^{n \times n}$

Cette approche est applicable pour tout système d'état autonome ou non. Pour l'évaluation des matrices A_k, le mode direct peut s'appliquer avec les mêmes performances que le mode inverse pour des systèmes autonomes (n variables indépendantes et n variables dépendantes). Pour les systèmes non autonomes, le mode inverse est plus efficace, puisque le nombre (n) des variable dépendantes est inférieur au nombre des variables indépendantes $(n+1)$ dans la trace de calcul.

Dans les équations V.12 et V.13, nous avons montré que le pas d'intégration adaptatif dépend des coefficients de Taylor des variables d'état $(\bar{x}_k)$. Normalement les dérivées partielles de ce pas d'intégration par rapport à ces coefficients ont du sens. En réalité, ces dérivées partielles doivent être complètement ignorées, car, en fonction du schéma de pas variable le résultat global de dérivées partielles peut beaucoup varier et de plus il est bruité et incorrect [34]. Ainsi, le gradient ∇h_i est forcé à 0. Nous rappelons que la propagation des dérivées du développement en série de Taylor dans l'équation V.15 est effectuée à la main. Ainsi, la désactivation des dérivées du pas d'intégration se réalise sans aucun problème, car il existe le contrôle manuel total de ces dérivées.

B. Dérivées partielles de la réponse du système par rapport aux paramètres de dimensionnement

Dans ce paragraphe, nous appliquons une version spéciale de la règle de fonctions

10. `accode` est l'acronyme de ACCumulation for Ordinary Differential Equations
11. `hov_reverse` est l'acronyme High Order Vectorial differentiation in REVERSE mode.

composées, afin de propager les dérivées partielles des variables d'état par rapport aux paramètres de dimensionnement, évaluées lors de l'utilisation du développement en série de Taylor :

$$\frac{\partial x(t_{i+1})}{\partial P} = \frac{\partial x(t_i)}{\partial P} + \sum_{j=1}^{k} \left(\frac{\partial \bar{x}_j}{\partial \bar{x}_0} \cdot \frac{\partial x(t_i)}{\partial P} + \frac{\partial \bar{x}_j}{\partial P} \right) \cdot h_i^j \tag{V.18}$$

où nous considérons, dés le début, la désactivation de la dérivation du pas d'intégration.

La difficulté dans cette récurrence est liée à l'évaluation des dérivées partielles $\frac{\partial \bar{x}_k}{\partial P}$. La version antérieure d'utilisation de la fonction `accode` est inapplicable pour cet aspect particulier, car la matrice A_k contient seulement les dérivées partielles des coefficients de Taylor des fonctions du système par rapport au coefficient Taylor des états d'ordre 0. Nous proposons une nouvelle stratégie pour le cas des dérivées partielles par rapport aux paramètres. Cette stratégie consiste à déclarer les paramètres comme variables indépendantes dans la trace de calcul des dérivées de la figure V.15. Ainsi, la trace est élargie à $(n+p)$ variables indépendantes, et n variable dépendantes, pour les systèmes autonomes et $(n+p+1) \times n$, au cas des systèmes non autonomes. En utilisant de nouveau la dérivation en mode inverse vectorielle d'ordre supérieur (`hov_forward`), on obtient la matrice A_k suivante :

$$A_k = U \cdot \left[\frac{\partial \bar{f}_k}{\partial \bar{x}_0}, \frac{\partial \bar{f}_k}{\partial P} \right], \quad \forall k = 1..n \tag{V.19}$$

Ainsi la matrice $A_k = A[n][n+p][k]$, contient aussi les dérivées partielles des coefficients de Taylor des fonctions du système d'état par rapport aux paramètres. La récurrence de l'équation V.17 n'est plus valable pour cette matrice A_k. Nous avons implémenté, lors de nos travaux, une nouvelle version sous ADOL-C de cette récurrence permettant d'évaluer les dérivées $\frac{\partial \bar{x}_k}{\partial P}$, utilisées par l'équation V.18 pour propager les dérivées des variables d'état. Cette nouvelle récurrence est formulée dans l'équation V.20 :

$$\begin{gathered} B_k = \frac{1}{1+k} \cdot \left(A_k + \sum_{j=1}^{k} (A_{j-1} \cdot I_{np} \cdot B_{k-j}) \right) \quad \forall k = 1..n \\ I_{np} = \begin{bmatrix} I_n \\ O_{p \times n} \end{bmatrix} \end{gathered} \tag{V.20}$$

où I_{np} est la matrice colonne formée par la matrice unité de taille $n \times n$ et la matrice nulle $O \in \mathbb{R}^{p \times n}$. A noter que cette récurrence est applicable seulement si la déclaration des paramètres dans la trace de calcul se déroule après la déclaration similaire des variables d'état.

V.4.1.3 Bilan de la Dérivation Automatique du développement en série de Taylor

Nous montrons dans ce paragraphe le processus entier qui permet l'utilisation de l'outil ADOL-C à la fois pour la résolution et la dérivation du développement en série de Taylor appliqué aux systèmes d'équations différentielles. L'idée générale se positionne autour d'une trace de calcul du système d'état, permettant dans un premier temps l'évaluation des coefficients Taylor pour la résolution du système et deuxièmement pour la dérivation de cette résolution. Nous précisons qu'il s'agit d'une dérivation hybride de la résolution du système, combinant la dérivation à la main et la Dérivation Automatique. Plus en détail, la Dérivation Automatique s'applique *par pas d'intégration*, alors que la dérivation à la main *propage* les dérivées à partir de $x(t=0)$ jusqu'à la réponse $x(t=t_r)$. C'est en effet le seul moyen d'évaluation de ces dérivées. Sachant que les coefficients de Taylor sont eux mêmes évalués avec la Dérivation Automatique, l'application de la cette technique pour la propagation des leurs dérivées correspondantes, aurait impliqué la Dérivation Automatique de l'outil ADOL-C, ce qui est absurde. La figure V.4.1.3 explique pas-à-pas comment se déroule ce processus pour le cas général des systèmes d'état non autonomes et en prenant en compte la dérivation des paramètres.

FIGURE V.16 – Le programme final C++ du système d'état non autonome instrumenté avec l'outil ADOL-C ; variables indépendantes (dans cette ordre) : $x \in \mathbb{R}^n,\ P \in \mathbb{R}^p, t \in \mathbb{R}$; variables dépendantes $\dot{x} \in \mathbb{R}^n$

c++ (ADOL-C):

```
trace_on(ID);
  x <<= ...;//∀x, activation des variables d'état (indépendantes);
  P <<= ...;// activation des paramètres (indépendantes);
  t <<= ...;//activation du temps (indépendantes);
  xp  = f(x,P); // le système d'état
  xp>>=... ;// activation des dérivées d'état (dépendantes);
trace_off();
```

Pas no. 1 Avancement de t_i à t_{i+1}	Pas no. 2 Dérivation (Automatique) du pas d'intégration	Pas no. 3 Propagation (à la main)
`forode*(ID, n, n+p+1,...);`	`hov_reverse(ID, n, n+p+1,...,A)`	
⇓	⇓	
$\bar{x}_k(t_i)$	`A[n][n+p][k]`	
⇓	`accode*(n, n+p+1, A, B);`	⇓
$x(t_{i+1})$	⇓	$\partial x(t_{i+1})/\partial x_0$
	`B[n][n+p][k]`$=[\partial\bar{x}_k/\partial\bar{x}_0,\ \partial\bar{x}_k/\partial P]$	$\partial x(t_{i+1})/\partial P$

* - dans la variante implémentée dans nos travaux.

En s'appuyant sur cette approche, nous répondons partiellement à notre objectif énoncé sur le tableau V.11. Seulement les dérivées partielles de la réponse finale par rapport aux valeurs initiales et par rapport aux paramètres sont évaluées.

V.4.1.4 Les algorithmes de type Runge-Kutta

Une autre famille d'intégrateurs est représentée par les algorithmes de type Runge-Kutta. Ces algorithmes avancent la solution du système d'état suivant la récurrence générale de l'équation V.21 :

$$x(t_{i+1}) = x(t_i) + h_i \cdot \dot{\tilde{x}}_i, \quad t_{i+1} = t_i + h_i \tag{V.21}$$

où h_i dénote le pas d'intégration est $\dot{\tilde{x}}_i$ est l'estimation d'une tangente qui varie en fonction du schéma (variante) d'intégration. Par exemple, la formulation avec $\dot{\tilde{x}}_i = \dot{x} = f(x, t_i)$ représente la méthode d'Euler, qui est le schéma d'intégration le plus simple. Cependant, cette méthode, malgré sa simplicité, accumule une erreur importante lors de la propagation de la résolution. Nous l'ignorons complètement dans nos travaux en considérant seulement des variantes de la méthode Runge-Kutta d'ordre supérieur à 4, sachant que l'intégration par la méthode d'Euler est seulement d'ordre 1.

A. Les schémas Runge-Kutta d'intégration d'ordre 4 et 5

En ce qui concerne les méthodes d'ordre supérieur à 4, nous considérons deux schémas d'intégration proposées en [73]. Il s'agit du schéma classique Runge-Kutta d'ordre 4 (RK4) et du schéma Dormand-Prince d'ordre 5 (RK5-DP). La description de ces méthodes d'intégration est illustrée sur le tableau V.14.

TABLE V.14 – Schémas Runge-Kutta d'ordre 4 et 5 d'intégration des systèmes d'équations différentielles ordinaires

RK4	RK5 - Dormand-Prince
$k_1 = h \cdot f(x_n, t_n)$	$k_1 = h \cdot f(x_n, t_n)$
$k_2 = h \cdot f(x_n + \frac{1}{2} \cdot k_1, t_n + \frac{1}{2} \cdot h)$	$k_2 = h \cdot f(x_n + a_{21} \cdot k_1, t_n + c_2 \cdot h)$
$k_3 = h \cdot f(x_n + \frac{1}{2} \cdot k_2, t_n + \frac{1}{2} \cdot h)$	$\cdots$
$k_4 = h \cdot f(x_n + k_3, t_n + h)$	$k_6 = h \cdot f(x_n + a_{61} \cdot k_1 \cdots + a_{65} \cdot k_5, t_n + c_6 \cdot h)$
$x_{n+1} = x_n + \frac{1}{6} \cdot k_1 + \frac{1}{3} \cdot k_2 + \frac{1}{3} \cdot k_3 + + \frac{1}{6} \cdot k_4$	$x_{n+1} = x_n + \sum_{i=1}^{6} b_i \cdot k_i$

Les coefficients $(a_{ij},\ \forall i = 1..6, j = 1..i-1)$, $(b_i, \forall i = 1..6)$, $(c_i, \forall i = 1..6)$ sont des constantes numériques et leurs valeurs peuvent être trouvées en [73].

Nous précisons que les schémas d'intégration d'ordre 4 utilisent 4 évaluations du système d'équations différentielles (f), alors que les schémas d'ordre 5 en utilisent 6. Le nombre d'évaluations n'est pas donc une variation linéaire avec l'ordre du schéma d'intégration. A titre informatif, pour un ordre d'intégration de 8, 11 évaluations du système sont nécessaires.

B. Le schéma de pas variable

Nous utilisons un seul schéma de pas variable pour l'intégration RK. L'idée principale est d'estimer le *prochain* pas d'intégration (h_{i+1}) après l'évaluation de l'état courant ($x(t_i)$). Ce pas est estimé en essayant de garder l'erreur entre la valeur d'état courante, $x(t_i)$, et une valeur *forcée*, $x^*(t_i)$ (évaluée avec un ordre d'intégration inférieur de 1) à un niveau acceptable spécifié par une erreur absolue ϵ_a et une relative ϵ_r. L'estimation de $x^*(t_i)$ se réalise effectivement en fonction du schéma d'intégration. L'idée de ce schéma de pas variable est illustrée dans l'équation V.22 :

$$|\Delta| = |x(t_i) - x^*(t_i)| \leq \mathsf{niveau} \tag{V.22}$$

où le niveau est défini en utilisant la différence entre la pente de variation de l'état antérieur et celle de l'état courant :

$$\mathsf{niveau} = \epsilon_a + \mathsf{max}\{|\dot{x}(t_{i-1})| - |\dot{x}(t_i)|\} \cdot \epsilon_r \tag{V.23}$$

Pour un système d'état de taille n, ces formules se généralisent dans l'équation V.24 :

$$\epsilon = \sqrt{\frac{1}{n} \cdot \sum_{i=1}^{n} \left(\frac{\Delta_i}{\mathsf{niveau}_i} \right)^2} \tag{V.24}$$

et si $\epsilon \leq 1$, le pas h_{i+1} est acceptable, sinon il faut le diminuer avec un facteur de secours, facr, en général égal à 0.8. Ainsi, la formule générale du pas prochain est :

$$h_{i+1} = \mathsf{facr} \cdot h_i \cdot \left(\frac{1}{\epsilon} \right)^{1/k} \tag{V.25}$$

où k est l'ordre courant du schéma d'intégration RK.

V.4.1.5 Dérivation Automatique de l'algorithme de Runge-Kutta

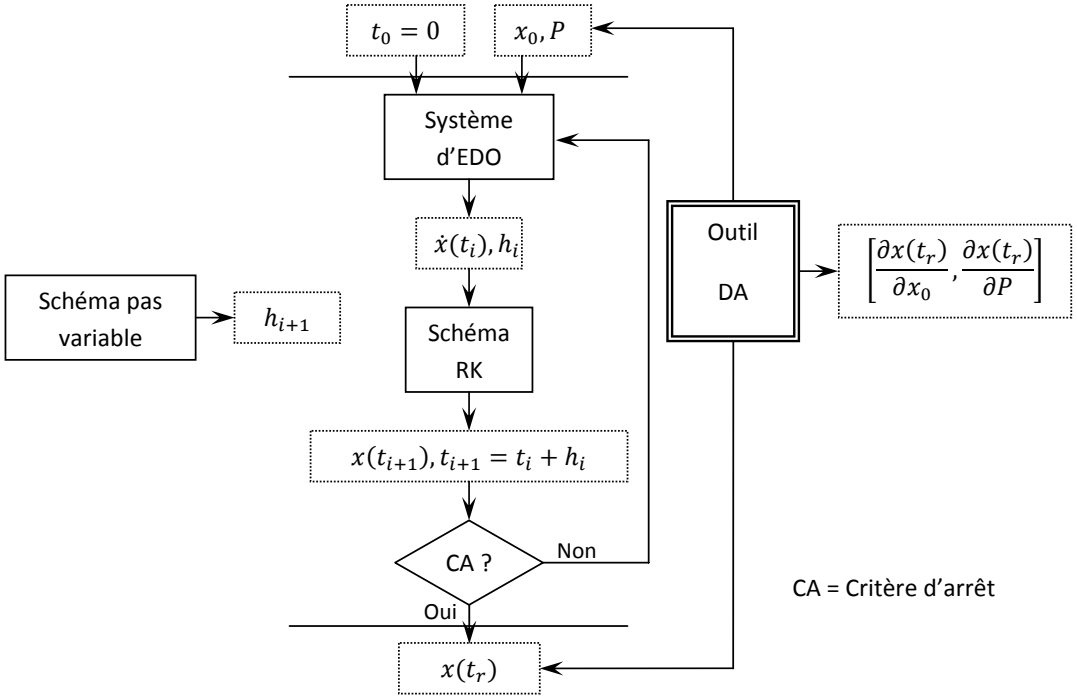
Dans ce paragraphe, nous proposons deux solutions pour employer la Dérivation Automatique à l'algorithme d'intégration RK d'équations différentielles. Dès le début, nous considérons que la dérivation du schéma du pas variable est complètement désactivée, sachant que le pas d'intégration dépend en général de l'état. De manière générale, en fonction de l'outil de Dérivation Automatique utilisé, cet aspect peut se réaliser de différentes façons. Cependant, la désactivation de la dérivation de ce schéma peut se réaliser avec un minimum des connaissances sur la stratégie de propagation des dérivées de l'outil en question.

A. Dériver globalement en une phase

Une approche très simple à mettre en œuvre, est de considérer le système d'équations différentielles, ainsi que le schéma d'intégration dans un seul programme, en entrée d'un outil de Dérivation Automatique. Ces aspects sont proposés en [89], où les auteurs utilisent le mode direct et inverse de dérivation pour évaluer les dérivées partielles des algorithmes RK par rapport au contrôle, en s'appuyant seulement sur le critère d'arrêt

temps final. L'avantage de cette approche est que l'effort supplémentaire de programmation [12] est considérablement réduit. L'idée générale de cette approche est illustrée sur la figure V.17 :

FIGURE V.17 – La stratégie *Dériver globalement* d'un schéma d'intégration Runge-Kutta



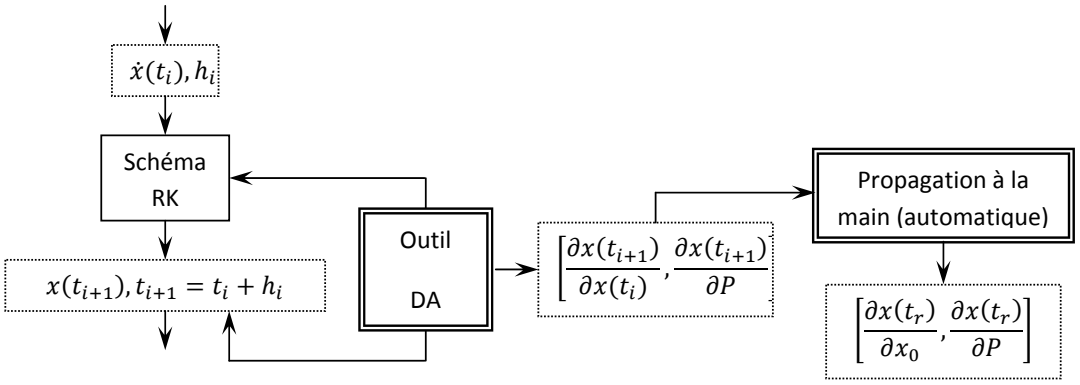
Vu que cette stratégie fait l'objet de l'application de la Dérivation Automatique d'un algorithme itératif, il est plus efficace de considérer le mode direct de dérivation, même si le nombre des variables de sortie (n) est toujours supérieur au nombre des entrées ($n+p$).

B. Dériver par pas d'intégration

Une approche plus efficace permettant d'évaluer les mêmes dérivées que dans la figure V.17, est de ne pas dériver de manière globale, mais seulement à chaque pas i d'intégration et de propager les dérivées à la main à partir de x_0 jusqu'à la réponse finale x_f. Cette approche s'avère plus efficace pour le calcul des dérivées, car elle donne la possibilité d'utiliser efficacement le mode inverse de dérivation. De plus la propagation des dérivées à la main permet certaines optimisations dans le programme, comme par exemple une gestion plus efficace de la mémoire en utilisant les pointeurs s'il s'agit du langage C/C++. De plus, le schéma de pas variable peut s'appeler à la fin de chaque itération et ainsi être facilement dissocié du programme actif (c'est à dire qu'on peut facilement le désactiver pour la dérivation).

12. Nous appelons *effort supplémentaire de programmation*, tout effort qui n'est pas lié à l'implémentation de l'algorithme RK, au système d'équations différentielles, ou à l'utilisation de l'outil de Dérivation Automatique.

FIGURE V.18 – La stratégie *Dériver par pas d'intégration* d'un schéma d'intégration Runge-Kutta



Le seul inconvénient de cette stratégie est l'effort supplémentaire d'implémentation par rapport à la stratégie *Dériver globalement*. Dans notre travail, nous considérons cette dernière stratégie (*Dériver par pas d'intégration*) pour deux raisons. Premièrement, elle plus rapide et moins coûteuse en mémoire, aspects qui deviennent bénéfiques en optimisation, surtout quand il s'agit d'un cahier des charges fortement contraint. Deuxièmement, cette approche de dérivation est similaire avec l'expansion en série de Taylor. Il sera donc plus facile de formaliser les deux approches dans une librairie ou outil.

V.4.2 Les critères d'arrêt

La procédure de dérivation des algorithmes d'intégration, proposée au paragraphe V.4.1, permet seulement d'évaluer les gradients de la réponse finale par rapport aux valeurs initiales et aux paramètres. Les critères d'arrêt permettent l'évaluation du bloc suivant de ces gradients, i.e. les dérivées partielles par rapport à la butée de l'état ($\tilde{x}_f$) et par rapport au temps final et ($\tilde{t}_f$), ainsi que le gradient complet du temps de réponse (∇t_r). Le but de ce paragraphe est d'illustrer ces aspects, par rapport au critère d'arrêt, qui peut se formuler en termes *d'état final imposé* ou *temps final imposé*. Nous ajoutons à ceci la règle suivante :

Une intégration peut être arrêtée par un seul critère d'arrêt et il est imposé qu'uniquement ce critère soit spécifié

V.4.2.1 Le critère *d'état final imposé*

Le premier critère se formule en terme *d'état final*, ce qui signifie que l'intégration s'arrête au moment où la variable d'état x_i atteint une valeur $\tilde{x}_{i_f}$ spécifiée. Cet aspect se formule avec l'équation implicite :

$$x_i(t_r) = \tilde{x}_{i_f} \tag{V.26}$$

où t_r est le temps de réponse.

En pratique, ce critère d'arrêt peut être utile, par exemple, lors de la simulation du mouvement d'une pièce mobile d'un dispositif qui atteint une butée formulée par les contraintes

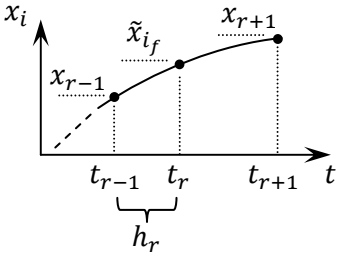
d'encombrement, ou bien, lors de la simulation transitoire d'un circuit électrique afin de déterminer le temps de réponse correspondant à une valeur d'état maximale, minimale, ou correspondante à la constante de temps du système.

Afin de déterminer le dernier pas d'intégration, une stratégie est de permettre l'intégration se dérouler jusqu'à un instant de temps, t_{r+1}, correspondant à une valeur d'état $x_i(t_{r+1})$ qui *dépasse* légèrement la valeur imposée $\tilde{x}_{i_f}$. La condition nécessaire et suffisante pour déterminer cet événement, est celle de l'équation V.27 et cet aspect est illustré graphiquement sur la figure V.19 :

$$(x_i(t_{r+1}) - \tilde{x}_{i_f}) \cdot \dot{x}_i(t_{r+1}) > 0 \tag{V.27}$$

Cette condition doit être posée après chaque pas d'intégration. Si elle est satisfaite à un moment donné, le schéma de pas variable de l'algorithme d'intégration doit être désactivé et il est nécessaire de forcer un nouveau pas h_r, calculé avec une méthode de résolution de l'implicite V.26, sachant que $t_r = t_{r-1} + h_r$.

FIGURE V.19 – Le critère d'arrêt *état final*



La méthode numérique de la sécante donne normalement un résultat en une itération, maximum deux, vu que l'état varie presque linéairement dans une région temporelle correspondante à un pas d'intégration. Ceci est acceptable, sachant que pour chaque itération, il est nécessaire de faire une résolution du système d'état.

L'évaluation du dernier pas d'intégration, implique la détermination du temps de réponse du système, défini implicitement dans V.26.

V.4.2.2 Les dérivées du critère *d'état final imposé*

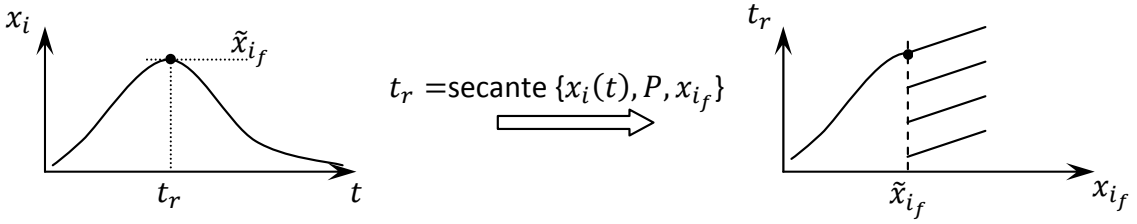
Grâce à ce critère d'arrêt, il est possible de déterminer les dérivées de la réponse finale par rapport à l'état final et le gradient complet du temps de réponse, défini implicitement par l'équation V.26.

Ainsi, en appliquant le théorème des fonctions implicites V.2 du paragraphe V.3.4.3, on est capable d'évaluer le gradient complet du temps de réponse :

$$\nabla^{\top} t_r = \begin{bmatrix} \dfrac{\partial t_r}{\partial x_0} = -\dfrac{1}{\frac{\partial x_i(t_r)}{\partial t_r}} \cdot \left[\dfrac{\partial x_i(t_r)}{\partial x_0}\right] = -\dfrac{1}{\dot{x}_i(t_r)} \cdot \left[\dfrac{\partial x_i(t_r)}{\partial x_0}\right] \\ \dfrac{\partial t_r}{\partial P} = -\dfrac{1}{\frac{\partial x_i(t_r)}{\partial t_r}} \cdot \left[\dfrac{\partial x_i(t_r)}{\partial P}\right] = -\dfrac{1}{\dot{x}_i(t_r)} \cdot \left[\dfrac{\partial x_i(t_r)}{\partial P}\right] \\ \dfrac{\partial t_r}{\partial \tilde{x}_{i_f}} = \dfrac{1}{\frac{\partial x_i(t_r)}{\partial t_r}} = \dfrac{1}{\dot{x}_i(t_r)} \end{bmatrix} \tag{V.28}$$

Ces dérivées partielles ne sont pas définies pour $\dot{x}_i = 0$. Autrement dit, le gradient du temps de réponse n'a pas de sens au cas où l'état final, $x_i(t_r)$, atteint une valeur spécifiée, $\tilde{x}_{i_f}$ dans un point d'extremum. La fonctionnelle qui définit le temps de réponse, i.e. l'algorithme de résolution de l'implicite (méthode de la sécante), n'est pas continue dans un tel point. Ces aspects sont illustrés graphiquement sur la figure V.20 :

FIGURE V.20 – La discontinuité du temps de réponse au critère d'arrêt *état final*



La fonction temps réponse, n'est pas définie dans ce cas spécial dans l'intervalle $[\tilde{x}_{i_f},\ \infty]$ pour un point d'intersection avec le maximum de x_i, ou dans $[-\infty,\ \tilde{x}_{i_f}]$ pour une intersection dans un point minimal. Ainsi, au point $\tilde{x}_{i_f}$, la fonction temps de réponse n'est pas continue, elle n'est donc pas dérivable et en conséquence son gradient n'a pas de sens. Ces aspects justifient dans l'équation V.27, le choix de l'inégalité stricte par rapport à zéro.

En nous appuyant sur le gradient du temps de réponse, il est possible aussi d'évaluer les dérivées partielles de la réponse du système par rapport à l'état final spécifié :

$$\frac{\partial x_j(t_r)}{\partial \tilde{x}_{i_f}} = \begin{cases} \dfrac{\partial x_j(t_r)}{\partial t_r} \cdot \dfrac{\partial t_r}{\tilde{x}_{i_f}} \equiv \dfrac{\dot{x}_j(t_r)}{\dot{x}_i(t_r)}, & \forall j = 1..n \\ 1, \text{si } j = i \end{cases} \tag{V.29}$$

V.4.2.3 Le critère *temps final imposé*

Le critère d'arrêt le plus simple à gérer est le *temps final*, qui notifie le schéma de pas variable qu'une valeur spécifiée pour le temps d'intégration ($\tilde{t}_f$) à été dépassée. Le dernier pas d'intégration, h_r, se calcule simplement en effectuant la différence $h_r = t_{r-1} - \tilde{t}_f$. Ceci implique que $t_r = \tilde{t}_f$. Ainsi, toutes les dérivées partielles du temps de réponse sont nulles, sauf $\dfrac{\partial t_r}{\partial \tilde{t}_f} = 1$.

Les dérivées partielles de la réponse du système par rapport au temps final spécifié sont simplement calculées en évaluant une seule fois le système :

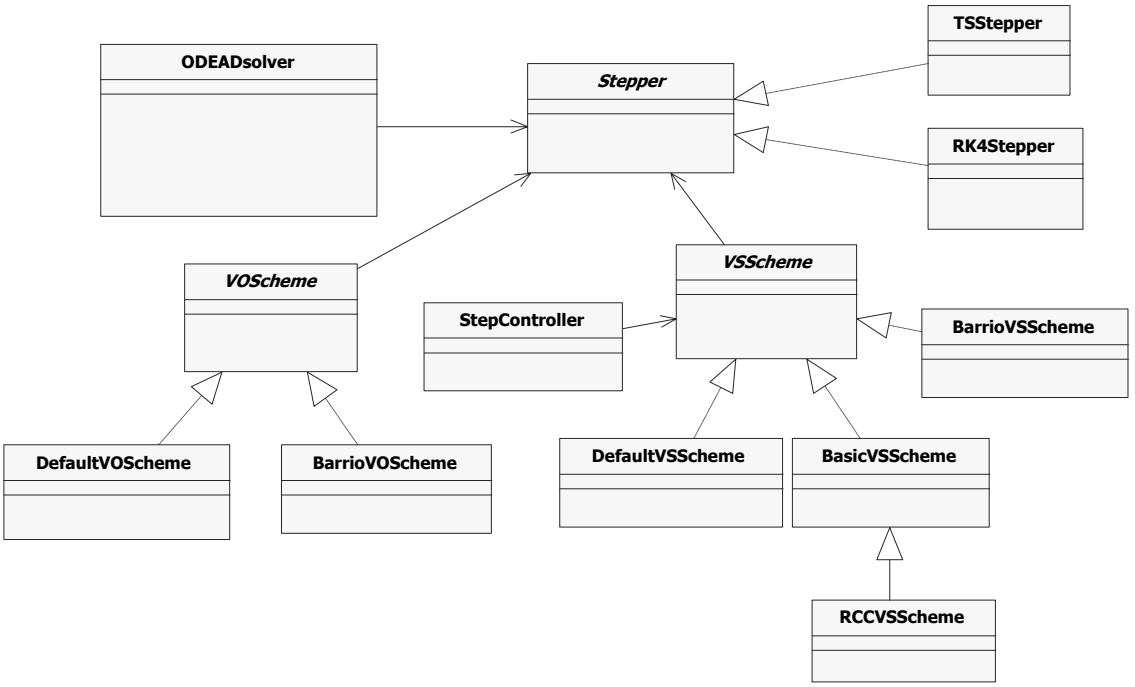
$$\frac{\partial x_j(t_r)}{\partial \tilde{t}_f} = \dot{x}_j(t_r) \tag{V.30}$$

Nous mentionnons que les dérivées partielles de l'état final par rapport à $\tilde{x}_{i_f}$ sont nulles, sachant que nous avons imposé la règle d'un seul critère d'arrêt spécifié pour une intégration.

V.4.3 Implémentation d'un outil de simulation et dérivation des systèmes dynamiques

Dans ce paragraphe, nous proposons une méthodologie logicielle permettant d'englober les aspects théoriques de dérivation des algorithmes d'intégration des systèmes d'équations différentielles dans un outil logiciel utilisable facilement. Lors d'un projet plus élaboré, le but principal est d'intégrer cet outil dans Reluctool [71], qui est un logiciel, utilisé par l'entreprise *Schneider Electric*. Celui-ci est dédie à la modélisation et la simulation des actionneurs mécatroniques par des réseaux des réluctances, comme l'application illustrée au paragraphe V.3.4.2. Actuellement, la modélisation de cet actionneur a seulement été proposée en régime statique.

FIGURE V.21 – Architecture logicielle de l'outil *ODEADSolver* incorporant des intégrateurs d'équations différentielles



Des développements sont en cours pour modéliser de différentes façons sous Reluctool des scénarii de mouvement des pièces mobiles, en partant de l'hypothèse que le système ait un seul dégrée de liberté. Un intégrateur pas à pas résout un système d'équations différentielles électriques, magnétiques et mécaniques. Le couplage entre le système électrique

et magnétique se fait par l'intermédiaire des variables d'état issues du flux de mailles ($\dot{\phi}$). Notre objectif à moyen terme est de remplacer dans un premier temps cet intégrateur, qui est un schéma d'ordre 5 d'intégration Runge-Kutta, par les schémas d'intégration proposés au paragraphe V.4.1, de décrire le système d'état dans une manière convenable pour la Dérivation Automatique et d'encapsuler tout dans un composant de calcul *IACr*, décrit au paragraphe IV.2.1. Ce composant doit être utilisable dans les services de CADES, notamment dans le service d'optimisation, afin de réaliser la conception des actionneurs mécatroniques dynamiques.

Afin de préparer la mise en œuvre de ces aspects, nous proposons l'architecture logiciel des intégrateurs présentée sur la figure V.21, qui donne la possibilité de choisir, soit une intégration s'appuyant sur le développement en série de Taylor, soit une intégration de type Runge-Kutta. Elle met aussi à disposition une liste des schémas de pas variable et donne la possibilité de choisir facilement son critère d'arrêt. Cet outil d'intégration est complètement instrumenté avec ADOL-C, dans la logique illustré au paragraphe V.4.1. L'implémentation de cet outil a été réalisée dans le langage C++, langage supporté par l'outil ADOL-C, en s'appuyant sur les aspects de programmation orienté-objet.

L'outil baptisé *ODEADSolver* développé dans notre travail, permet, grâce à son système d'interfaces [13], l'ajout de nouvelles méthodes numériques d'intégration et de divers schémas de pas variable.

Nous avons intégré l'outil *ODEADSolver* dans une phase opérationnelle exploitable seulement pour l'environnement logiciel CADES. La figure V.22 montre comment on peut définir un modèle de dimensionnement dynamique.

a.

```
sml:
n = constante; //taille du système d'état;
dx1_dt(x1,...,xn, t)=...;
...              //système d'état(scalaire);
dxn_dt(x1,...,xn, t)=...;
```

b.

```
sml:
import "ODEADSolver.jar";//l'import de ODEADSolver(fonction de fonctions);
dx_dt(x[n]) = [dx1_dt(x[0],...,x[n-1]),..., dxn_dt(x[0],...,x[n-1])];
declare x0[n];//les valeurs initiales;
declare xf[n];//valeurs finales (critère d'arrêt état final);
[x1,...xn, tr] = solve(dx_dt(_),x0, xf, schema,... ); //résolution(vectorielle);
```

FIGURE V.22 – Modélisation sous *sml* des systèmes d'équations différentielles et leur résolution ; a. Modélisation analytique du système d'équations différentielles ; b. Appel à la résolution numérique vectorielle.

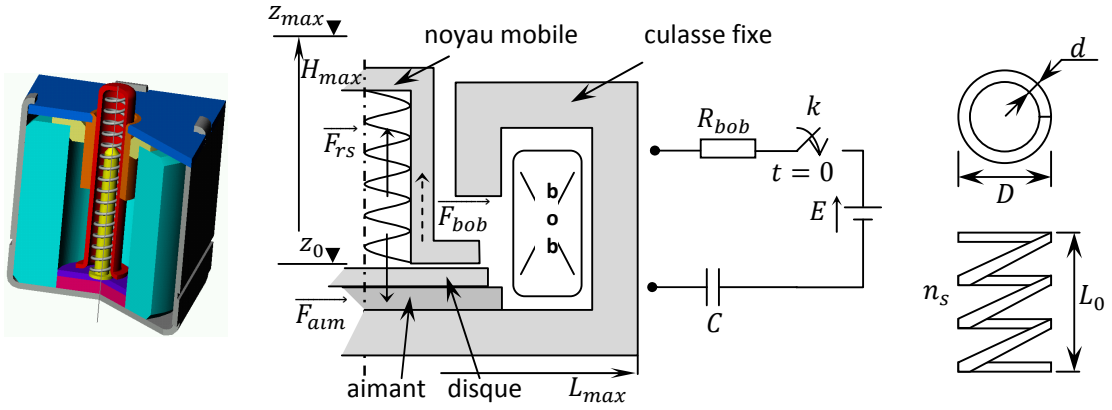
13. L'outil *ODEADSolver* contient l'interface *Stepper* permettant d'ajouter facilement de nouveaux schémas d'intégration.

Son intégration sous Reluctool est en cours de spécification au travers du stage de Thang Dang-Quoc.

V.4.4 Application pour le dimensionnement des dispositifs mécatroniques

Dans ce paragraphe, nous proposons le dimensionnement d'un dispositif électromagnétique avec des pièces en mouvement. Ce dispositif est un déclencheur dynamique, fabriqué par l'entreprise Schneider Electric dans les années '70. Son dimensionnement a été proposé par M. Perrault en [4]. Les auteurs proposent une stratégie d'optimisation basée sur une optimisation SQP du modèle statique et une optimisation manuelle du modèle dynamique, en s'appuyant sur les résultats de l'état du système fournis par une intégration Runge-Kutta d'ordre 4. Cette optimisation manuelle consomme, évidement, une grande quantité de temps et en plus, elle ne peut être réalisée qu'en simplifiant le modèle dynamique. Nous proposons dans ce paragraphe une technique complètement automatique qui demande un effort de réalisation beaucoup moins élevé, se résumant seulement à la définition du système d'état et à la modélisation statique du dispositif. Nous l'utilisons comme un banc de test pour les approches de Dérivation Automatique des intégrateurs des systèmes d'équations différentielles. Le but est de s'appuyer sur son modèle dynamique, d'établir un cahier des charges et de réaliser son dimensionnement en s'appuyant sur des algorithmes SQP. Le modèle dynamique est résolu et dérivé avec l'outil *ODEADSolver* présenté au paragraphe V.4.3.

Ce dispositif, illustré dans une section demi-transversale sur la figure V.23, est composé d'une bobine de résistance électrique R, alimentée par un circuit électrique contenant une source de tension continue, un interrupteur et une capacité. La bobine crée un flux qui circule dans un circuit magnétique composé d'une culasse fixe, d'un aimant et d'un noyau mobile.



a. b. c.

FIGURE V.23 – Le déclencheur dynamique ; a. Vue 3D du dispositif fabriqué ; b. Demi-section transversale du dispositif avec les éléments constitutifs de sa structure ; c. Le ressort est ses paramètres

Afin d'avoir un système du 2-ième ordre, un condensateur a été introduit dans le circuit,

sans utilité réelle. Dans la phase non opérationnelle, le noyau mobile est équilibré par la résultante entre la force de l'aimant et la force du ressort. Dans la phase opérationnelle, à partir du moment de la fermeture de l'interrupteur, la force électromagnétique créée par la bobine s'oppose à la force du ressort et le noyau décolle vers le haut entre z_0 et z_{max}.

V.4.4.1 Modélisation du déclencheur dynamique

Le déclencheur est modélisé statiquement par des réseaux de réluctances. Le système dynamique comporte quatre équations différentielles ordinaires, dont l'état est représenté par :

- i - le courant dans la bobine.
- $\frac{di}{dt}$ - la dérivée temporelle du courant, introduite pour réduire l'équation différentielle d'ordre second à deux équations ordinaires.
- z - la position du noyau mobile.
- v - la vitesse du noyau mobile.

Ces aspects de modélisation, ainsi que le calcul de la force totale actionnant sur le noyau, sont détaillés dans l'Annexe C.3.

V.4.4.2 Cahier des charges d'optimisation

Le but de l'optimisation est de minimiser la masse totale du dispositif pour un temps de réponse inférieur à $3.5ms$ et une force imposée en position haute de $15N$. En parallèle, il existe aussi le besoin du dimensionnement du ressort. Le tableau V.15, centralise certaines spécifications du cahier des charges du dispositif.

TABLE V.15 – Le cahier des charges d'optimisation du déclencheur dynamique

Paramètre	Déscription	Contraint
W_per	Energie de percussion en position haute	$[0.12; 100]$ J
t_r	Temps de réponse	$[0; 3.5]$ ms
F	Force résiduelle en position haute	15 N
t_choq	Tenue au choc en position basse	$[2000; 10000]$ m/s^{-2}
$masse$	masse totale	minimiser
c_viab	Contrainte de viabilité du ressort	$[0.25; 1.2]$ m^{-1}
c_fab	Contrainte de fabrication du ressort	$[0.08; 0.2]$

Ce tableau illustre seulement les contraintes sur les sorties.

La contrainte sur l'énergie de percussion en position haute (voir l'Annexe C.3.4) assure la promptitude du résultat de déclenchement. Elle est imposée à une valeur inférieure de 0.12 J. La contrainte de la force résiduelle en position haute, assure la stabilité du noyau mobile après le déclenchement. Elle est imposée à une valeur fixe. La tenue au

choc en position basse dénote la stabilité du noyau mobile aux vibrations dans la phase non-opérationnelle (voir l'Annexe C.3.5). Sa contrainte associée est utile par exemple si le dispositif est supposé fonctionner dans un milieu embarqué, en contact avec d'autre pièces ou dispositifs en vibration.

A ces contraintes s'ajoutent 17 dégrées des libertés, donc un ensemble de 17 paramètres d'entrée, dénotant les dimensions du dispositif et les paramètres du ressort. Parmi ces variables des dimensionnement, il existe la contrainte, $0.3mm \geq z_0 \leq 0.8mm$, où z_0 représente la position initiale du noyau. Le paramètre H_{max}, représentant la butée du noyau (le critère d'arrêt) est aussi contraint dans l'intervalle $[24.0mm; 40.0mm]$.

V.4.4.3 Résultats

Dans ce paragraphe, nous montrons une série des résultats concernant la résolution et la dérivation du système d'état utilisé dans la modélisation du déclencheur, ainsi que les résultats d'optimisation.

A. La résolution du système d'état

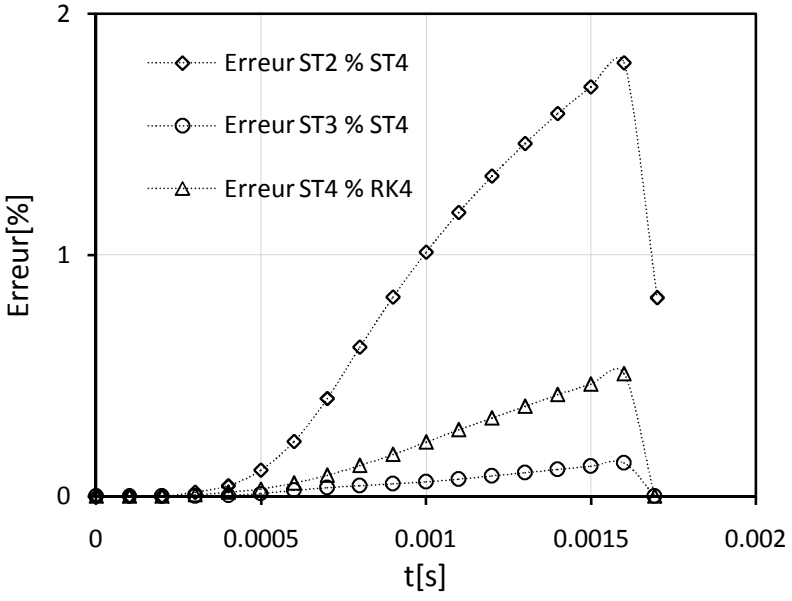


FIGURE V.24 – Les erreurs relatives entre les valeurs discrètes de la position, intégrée entre $z_0 = 0.03mm$ et $z_{max} = 4.55mm$ ce qui implique un temps de réponse de $1.69ms$ en un nombre de 17 pas ; la valeur de z_{max} est déduite à partir de la configuration initiale du déclencheur ; l'erreur est calculée avec la formule $\epsilon_r[\%] = \left| \frac{x - x_{ref}}{\max\{|x_{ref}|\}} \right| \times 100$

Dans ce paragraphe, nous utilisons les valeurs initiales de paramètres utilisées en optimisation, pour comparer les résultats entre les intégrations du système d'état de (voir l'Annexe C.3.2). Nous appliquons la méthode du développement en série de Taylor (ST) d'ordre 2 jusqu'à 4 et le schéma d'intégration Runge-Kutta d'ordre 4, afin de réaliser une comparaison en terme de précision des résultats d'intégration. La variable d'état concernée

est la position du noyau mobile, z. Toutes ces intégrations sont réalisées avec un pas fixe, $h = 0.1ms$.

La figure V.24 montre que l'erreur accumulée par le développement en série de Taylor d'ordre 2, par rapport à la résolution d'ordre 4, monte jusqu'à 1.8%. A partir de l'ordre 3 cette erreur reste inférieure à 1%. On peut admettre que l'ordre 3 d'intégration est suffisamment précis pour ce problème, avec un pas de $h = 0.1ms$.

La figure V.25 offre une visualisation de l'évolution en temps de la position, intégrée avec les schémas ST et RK d'ordre 5, en s'appuyant sur les schémas de pas variable de l'équation V.13 et V.22.

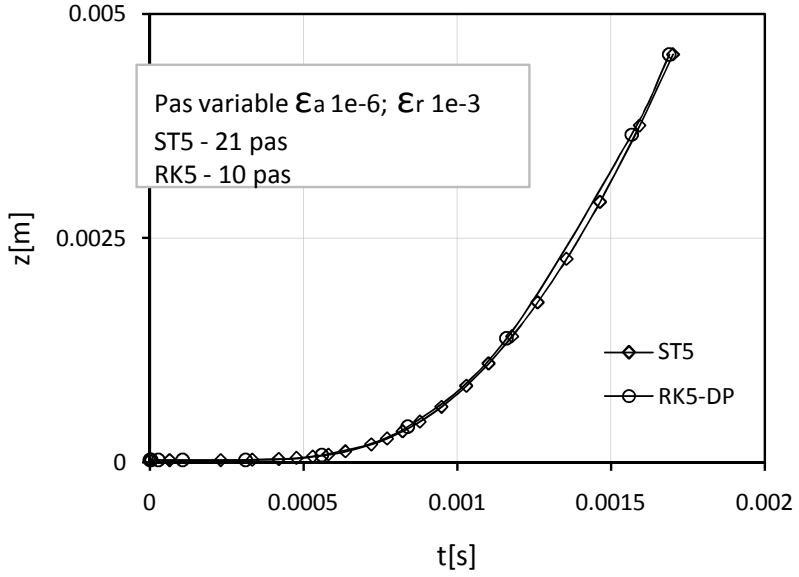


FIGURE V.25 – La réponse du système d'état calculé avec le développement en série de Taylor et la méthode Runge-Kutta en utilisant des schémas de pas variable ; variable concernée : la position du noyau mobile, z ; pour la configuration initiale du dispositif, le noyau mobile décolle à l'instant $t \approxeq 0.033ms$, moment où la force résiduelle devient positive

L'intégration RK est jusqu'à deux fois moins coûteuse en termes de quantité de pas par rapport au développement en série de Taylor. Ceci est compréhensible, puisque les schémas de pas variables pour l'intégration ST ont tendance à diminuer le pas dans les régions où la raideur de l'état devient importante.

B. Résultats de dérivées partielles

Dans ce paragraphe, nous nous intéressons au calcul des gradients de la force résiduelle en position haute et à ceux du temps de réponse, obtenus lors de l'application de Dérivation Automatique pour les schémas d'intégration ST5 et RK5-DP en pas variable. Ces gradients sont composés de plusieurs blocs, illustrés sur le tableau V.11. Afin de tester les valeurs des dérivées par rapport aux paramètres initiaux d'intégration, nous reprenons la configuration initiale du dispositif, sur laquelle nous faisons varier z_0 entre $0.05mm$ et $0.08mm$ avec un pas $\Delta z_0 = 0.00003mm$. Ceci peut se réaliser en modifiant, par exemple, l'épaisseur du

disque. Pour chaque valeur discrète de z_0, nous réalisons une intégration et un calcul de force résiduelle en position haute. La butée du noyau mobile reste constante à la valeur $z_f = z_{max}$. La variation de la force illustrée sur la figure V.26a est due seulement à la variation du courant en état final, sachant que la position finale reste fixe. Cette force décroit avec l'augmentation de la valeur initiale de la position, c'est-à-dire que la force crée par la bobine s'affaiblisse lors de l'augmentation de l'épaisseur du disque pour des paramètres constants du ressort. Cela se passe de la même façon pour le temps de réponse, qui diminue lorsque le point de départ de simulation est plus proche de l'état final.

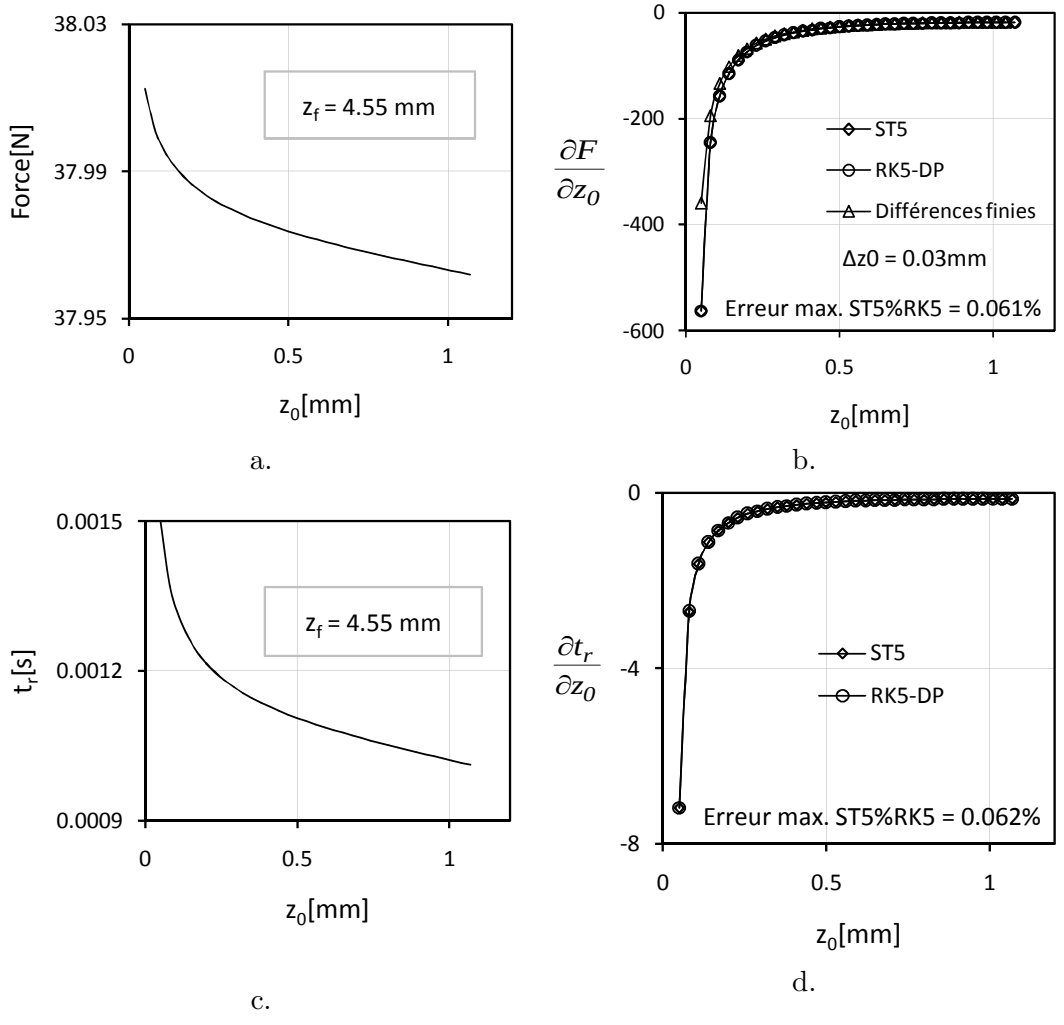


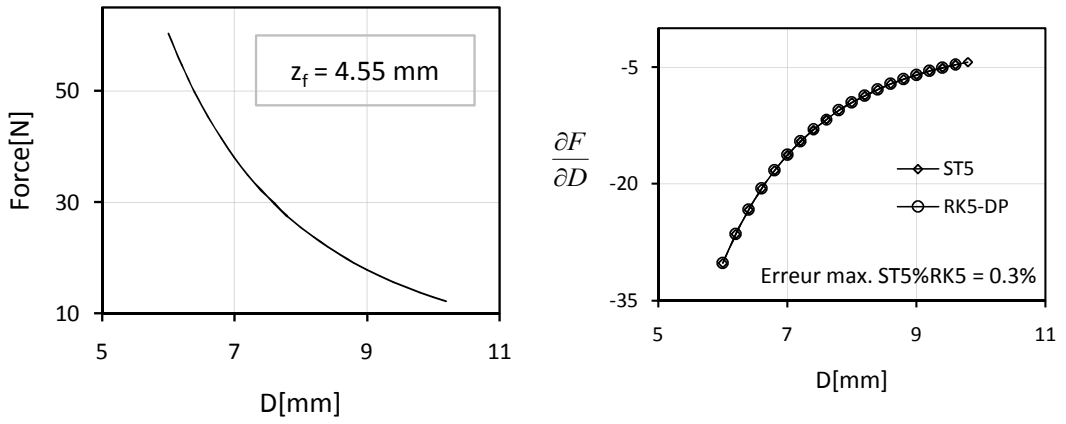
FIGURE V.26 – La variation et les dérivées partielles de la force résiduelle et du temps de réponse en position haute par rapport à la variable initiale z_0 de la position ; schémas d'intégration considérés : ST5 et RK5-DP en pas variable avec les tolérances $\epsilon_a = 10^{-6}$ et $\epsilon_r = 10^{-3}$

Sur la figure V.26b, les courbes de variation des dérivées partielles de la force par rapport à z_0, sont tracées dans les mêmes conditions. Leur allure est comparée à la courbe obtenue en appliquant la méthode de différences finies dans les mêmes points. Nous ajoutons la mention que la méthode des différences finies n'est utilisée ici comme une référence

pour le calcul des dérivées.

Les dérivées obtenues pour les deux schémas d'intégration sont quasi-identiques, l'erreur maximale étant d'environ 0.062%.

Pour le deuxième bloc de gradients, celui concernant les dérivées partielles par rapport aux paramètres, nous réalisons la même démarche, afin d'obtenir une variation de la force résiduelle en position haute. Le diamètre de la spire du ressort varie sur la figure V.27 dans la plage des valeurs $[6mm; 10.4mm]$ avec un pas $\Delta D = 0.2mm$.



a. b.

FIGURE V.27 – La variation et les dérivées partielles de la force résiduelle en position haute par rapport au diamètre du ressort D ; schémas d'intégration considérés : ST5 et RK5-DP en pas variable avec les tolérances $\epsilon_a = 10^{-6}$ et $\epsilon_r = 10^{-3}$; cette variation prend en compte la configuration initiale du dispositif

De même que pour les dérivées partielles de la force résiduelle par rapport à la position initiale, les dérivées par rapport au diamètre de la spire du ressort restent quasi-identiques, pour les deux schémas d'intégration considérés.

C. Performances des schémas d'intégration

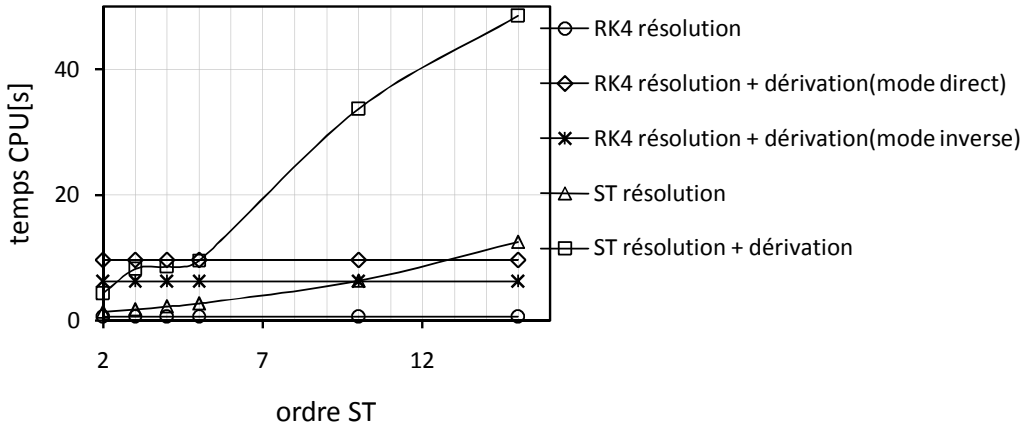
Dans ce paragraphe, nous illustrons les performances d'exécution en termes de temps CPU et de mémoire utilisée pour les schémas d'intégration ST, en faisant varier l'ordre d'intégration entre 2 et 15. Ces performances sont comparées avec le schéma d'intégration RK4. Nous prenons en compte aussi la configuration initiale du dispositif, en utilisant des intégrations en pas fixe. Nous n'évaluons que les temps CPU dédiées effectivement à l'intégration et à la dérivation. Les temps morts, comme par exemple les manipulations des données avant ou après les pas d'intégration, sont éliminés complètement de cette étude.

Pour les temps CPU d'exécution, nous pouvons remarquer les aspects suivants, dans le cas particulier de notre application :

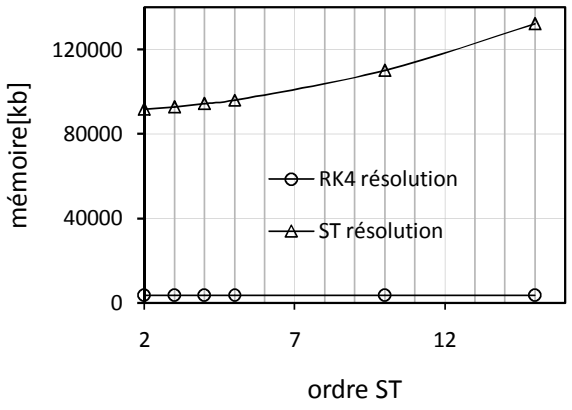
- le temps CPU de résolution du schéma d'intégration ST4 est jusqu'à 3.5 plus important que la résolution RK4 et ce temps augmente vite jusqu'à une valeur de 31 fois plus importante pour une intégration d'ordre 15.
- les temps CPU de dérivation, y compris la résolution, sont très proches pour les deux schémas d'intégration, RK4 et ST4. Pour une intégration ST15, le temps CPU

augmente jusqu'à une valeur de 5 fois plus importante.

Nous mentionnons que la mémoire utilisée par le schéma d'intégration ST4 est jusqu'à 26 fois plus importante que la résolution RK4 et elle augmente jusqu'à une valeur de 36 fois plus importante pour une intégration d'ordre 15.



a.



b.

FIGURE V.28 – Les performances des intégrateurs ; a. les temps CPU consommés par la résolution et la dérivation par les schémas d'intégration ST d'ordres 2, 3, 4, 5, 10, 15, comparés avec le temps CPU consommé par RK4 ; la valeur du pas fixe : $h = 10^{-7}s$; nombre des pas 16900 ; b. La quantité mémoire dynamique consommée par les schéma ST comparée avec la mémoire de résolution utilisée par le schéma RK4 ; la valeur du pas fixe : $h = 10^{-5}s$; nombre des pas 169 ; Ces valeurs ont été obtenues en utilisant l'outil *mpatrol* de profilage des programmes C/C++ ; voir `http://mpatrol.sourceforge.net/doc/`

D. Les résultats d'optimisation

Dans ce paragraphe, nous illustrons l'évolution de certains paramètres du déclencheur dans l'optimisation. Les grandeurs de sortie analysées sont la fonction objectif (la masse du dispositif), la force résiduelle en position haute et le temps de réponse du système dynamique. Parmi les entrées, nous avons choisi la position initiale du noyau mobile. L'optimisation converge en 29 itérations, avec une précision de 10^{-10}. Le schéma de résolution du système dynamique utilisé est la ST5 en pas variable.

Nous considérons acceptables ces variations, ainsi que le nombre d'itérations. Ceci

dénote de nouveau que les gradients de la réponse du système d'état sont évalués avec une bonne précision pour les algorithmes d'optimisation SQP.

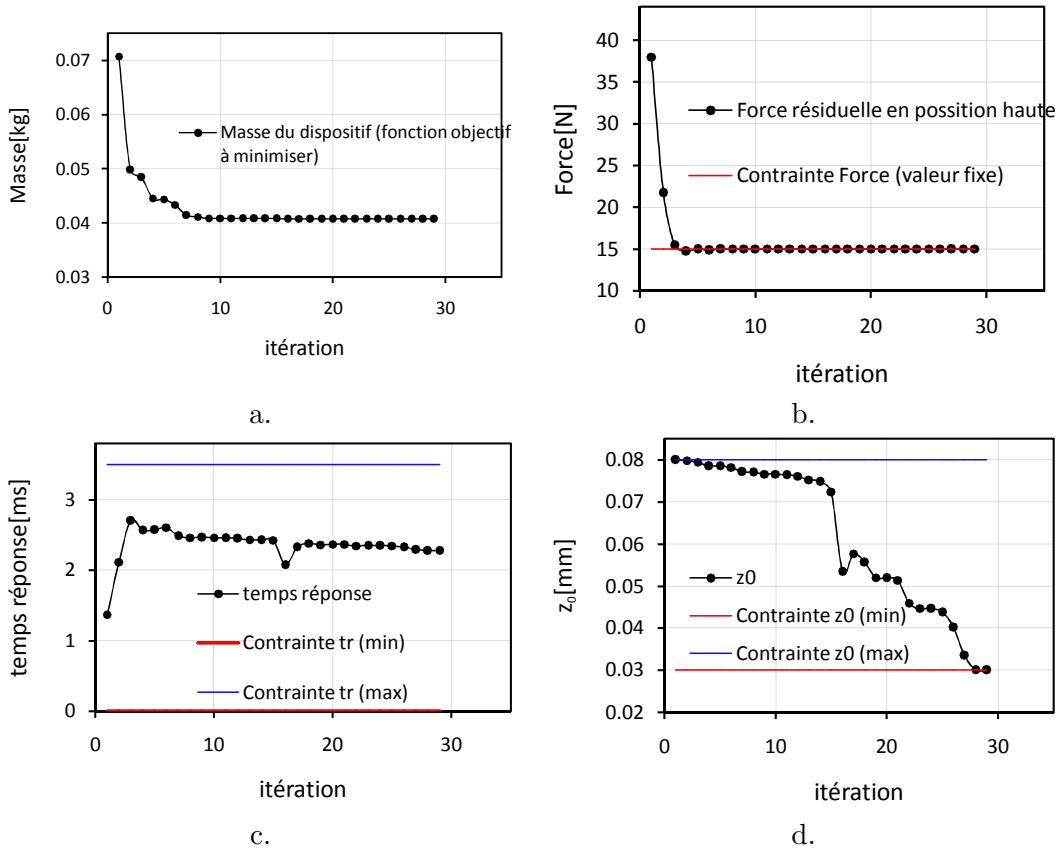


FIGURE V.29 – Résultats d'optimisation SQP appliquée au déclencher dynamique

V.5 Conclusion

Nous proposons dans la première partie de ce chapitre de dériver automatiquement un algorithme d'intégration des fonctions, pour lequel il n'a été pas nécessaire de connaître les aspects mathématiques implémentés ou la structure de l'algorithme utilisé. Ceci est contraire au cas de dérivation de l'algorithme de résolution des systèmes d'équations implicites ou d'intégration des systèmes d'équations différentielles, où des interventions supplémentaires ont été nécessaires afin d'appliquer correctement la Dérivation Automatique. L'intention de l'intégrale était de bénéficier d'un des avantages majeurs de la Dérivation Automatique : elle s'applique sans qu'il soit nécessaire de connaître la structure interne de la fonction à dériver. C'est un des avantages qui rendent possible la dérivation de tout programme avec peu ou pas d'effort de la part de l'utilisateur. Les résultats fournis par ADOL-C sont comparés à une référence obtenue en appliquant la dérivation à la main, en s'appuyant sur les propriétés mathématiques de l'intégrale, dans le cadre du dimensionnement d'un micro-actionneur électromagnétique intégré dans la structure d'un miroir déformable.

Dans sa deuxième partie, ce chapitre propose d'utiliser l'outil ADOL-C pour la méthode numérique de type Newton de résolution des systèmes d'équations implicites non-linéaires, afin d'évaluer les dérivées partielles de la solution par rapport aux paramètres. Le mode direct est préférable, sachant qu'il s'agit d'une méthode numérique itérative. Des solutions mathématiques sont évoquées, afin d'exploiter d'une manière efficace la propagation des ces dérivées, comme par exemple le fait d'ignorer la dérivée d'ordre seconde qui peut apparaître dans le calcul des dérivées. Les résultats fournis par la Dérivation Automatique sont ensuite comparés au théorème des fonctions implicites. Ce théorème peut s'appliquer sous divers formes, i.e en utilisant le calcul formel, en utilisant de nouveau la Dérivation Automatique et plus simple, en exploitant de nouveau les fonctionnalités additionnelles de l'outil ADOL-C, plus précisément le pilote `inverse_tensor_eval`. La conclusion est que la dérivation de l'algorithme de Newton-Raphson est très précise, mais avec un coût plus élevé de temps d'exécution et mémoire. Ces dérivées partielles sont ensuite exploitées dans l'environnement logiciel CADES pour le dimensionnement d'un actionneur linéaire statique.

Ce chapitre propose dans sa dernière partie d'utiliser toujours ADOL-C pour la Dérivation Automatique des algorithmes numériques de résolution des systèmes d'équations différentielles ordinaires. Deux schémas d'intégration sont proposés, i.e. le développement en série de Taylor et les méthodes Runge-Kutta. Pour le développement en série de Taylor nous utilisons de nouveau des fonctionnalités additionnelles existantes en ADOL-C, i.e des pilotes pour l'évaluation et la dérivation des coefficients de Taylor. Dans la phase initiale de nos travaux, ces pilotes étaient applicables seulement pour des systèmes autonomes. Pour les non autonomes, nous proposons des variantes spéciales implémentées au cours de nos travaux. Pour les deux schémas d'intégration, RK et ST, nous proposons la même stratégie de dérivation : employer la Dérivation Automatique par pas de résolution et propager à la main les dérivées partielle à partir de l'état initial jusqu'à l'état final. Cette stratégie donne le contrôle total sur un pas d'intégration, permettant de désactiver les dérivées du pas d'intégration ou de gérer à l'avenir des événements discrets. Les performances de l'évaluation des dérivées de l'état final et celles du temps de réponse du système dynamique sont analysées pour ces deux schémas de résolution dans le cadre du dimensionnement d'un déclencheur dynamique. Le schéma RK est plus performant en résolution ainsi qu'en dérivation, sachant que le schéma de développement en série de Taylor paye un coût élevé pour l'évaluation des dérivées d'ordre supérieur.

Toutes les méthodes numériques étudiées dans ce chapitre sont applicables pour la modélisation des dispositifs électriques dans le langage de CADES en utilisant le concept de fonctions de fonctions externes. Grâce à ce concept, des modèles de dimensionnement utilisant des intégrales, des résolutions de systèmes implicites non linéaires et des résolution de systèmes d'état peuvent être optimisés sous CADES par des algorithmes basés sur le calcul des gradients.

Conclusion générale

Au cours de ces travaux de thèse, nous avons proposé d'utiliser la technique de Dérivation Automatique, afin d'évaluer les gradients des modèles de dimensionnement, exploitables par les algorithmes d'optimisation tels que SQP (Minimisation Séquentielle Quadratique). La Dérivation Automatique est peu utilisée aujourd'hui en Génie Electrique. L'originalité de nos travaux repose sur sa utilisation pour le dimensionnement des dispositifs électriques.

Cette technique puissante a été intégrée dans l'environnement logiciel CADES, développé au cours de ces dernières années, au sein de l'équipe Mage du G2Elab. Grâce à notre intervention sous CADES, l'ingénieur concepteur a désormais la possibilité de décrire ses modèles de dimensionnement par des algorithmes, quelle que soit leur complexité et la dérivation est prise en charge automatiquement pour lui. Cette contrainte était à l'origine de beaucoup de complications et de limitations avant notre intervention. Le concepteur avait la responsabilité d'implémenter lui même la dérivation de tout programme informatique qu'il souhaitait utiliser en modélisation. Pour des algorithmes complexes, cette tâche demandait beaucoup d'effort humain, le concepteur étant dans la posture d'exploiter dans la mesure du possible toutes les propriétés mathématiques de la méthode numérique utilisée. Souvent, la solution la plus simple, mais pas la plus performante, était l'utilisation des différences finies.

Nous avons enrichi le langage de modélisation disponible sous CADES en introduisant la notion de variables vectorielles. Ce type de données multiples est souhaitable là où une fonction est supposée retourner non seulement une valeur, mais plusieurs. Cela évite d'implémenter la notion de persistance. Les variables vectorielles sont utiles, en général pour tout compactage des donnés ayant la même signification. Les aspects de vectorisation ont été réalisables d'une manière très efficace grâce à la Dérivation Automatique.

Les avancées réalisées au niveau du langage de modélisation existant sous CADES, ont été possibles sans complication majeure concernant la tâche de dérivation. Les difficultés principales provenaient de l'utilisation d'un outil de Dérivation Automatique supportant un langage de programmation autre que le langage cible du composant implémentant la norme *ICAr*. Nous avons dû coupler ces deux langages (Java et C/C++) par des architectures qui

ont demandé un effort de réalisation considérable. L'interface JNI, utilisée avec modération en respectant les règles proposées par ses développeurs, [63] permet d'être très efficace en rapidité et fiable pour le calcul, par rapport aux autres alternatives existantes d'interfaçage entre ces deux langages.

Dans nos travaux nous nous sommes aussi intéressés à la Dérivation Automatiques des méthodes numériques couramment utilisées dans la modélisation des dispositifs électriques. Une méthode numérique d'intégration des fonctions est dérivée avec ADOL-C. Cette méthode est implémentée dans un programme complexe, développé sur un millier des lignes de code. La Dérivation Automatique a été appliquée avec succès par une mise en œuvre de seulement quelques heures. La méthode de Newton-Raphson de résolution des systèmes d'équations implicites est dérivée dans la même philosophie que l'intégrale. La particularité ici est que nous exploitons les propriétés de l'algorithme Newton-Raphson, afin de rendre cette méthode efficacement dérivable, notamment en ignorant les dérivées d'ordre second qui apparaissent dans la propagation des valeurs des dérivées. La résolution des systèmes d'équations différentielles ordinaires a aussi été dérivée. Ici, il est nécessaire de dériver tout l'historique des variables d'état, à partir des valeurs initiales jusqu'à l'état final. Grâce à ces aspects, le concepteur aura la possibilité de dimensionner le temps de réponse d'un système dynamique et les états initiaux et finaux. La dérivation des systèmes d'équations différentielles entre dans un projet plus élaboré. Le but final est d'intégrer ces développements dans des outils comme Reluctool développé au sein du laboratoire et utilisé par Schneider Electric pour le dimensionnement des actionneurs électromécaniques.

Perspectives

Différentes perspectives s'offrent suite à ces travaux, pour l'amélioration du langage de modélisation *sml*, pour l'amélioration des architectures de l'environnement logiciel CADES, pour les méthodes numériques utilisables dans la modélisation des dispositifs et pour l'amélioration des outils de Dérivation Automatique.

Le besoin le plus important est relatif à l'architecture de l'environnement CADES intégrant les aspects de Dérivation Automatique avec l'outil ADOL-C. Sachant que le programme cible d'un composant de calcul implémentant la norme *ICAr* est aujourd'hui décrit en Java, un outil de Dérivation Automatique supportant ce langage de programmation simplifierait beaucoup l'architecture du composant de calcul, en éliminant les passages par l'interface JNI de communication Java/C++. Un outil performant de Dérivation Automatique pour des programmes en Java est apparu en 2008 et il s'appelle *ADiJaC* [82]. Cet outil implémente les deux modes de dérivation en utilisant la technique de transformation du code source. Les auteurs précisent que les performances d'*ADiJaC* sont comparables avec les références dans le domaine de la transformation du code source (i.e. TAPENADE [56][70], ADIFOR [13], etc...). Cet outil est d'un intérêt majeur pour la suite de nos travaux (*ADiJaC* étant en cours de développement au du début de cette thèse).

Cette perspective ne se limite pas seulement à l'utilisation des outils de Dérivation Automatique pour des programmes en Java, mais aussi à d'autres outils puissants, implémentant la transformation source-to-source, comme TAPENADE. Cela serait notamment envisageable pour dériver directement un langage comme *sml*.

Sachant que l'architecture que nous proposons dans ces travaux fait l'objet de l'utilisation de l'interface JNI de communication Java/C, une perspective intéressante pourra être la prise en compte d'autres solutions pour réaliser cet interfaçage.

Des travaux intéressants restent à faire afin d'assurer la portabilité du composant de calcul vers d'autres plateformes, notamment sous Unix/Linux, sachant que pour l'instant, tous nos développements ont été réalisés sous Microsoft Windows ™. Nous mentionnons que dans nos travaux, le composant *ICAr* perd la notion de portabilité en raison de l'existence du programme natif C++ de calcul des dérivées, mais dans le principe, rien n'empêche de le rendre portable, en lui faisant transporter les ressources nécessaires

pour chaque environnement cible. Actuellement, la modélisation sous *sml* est possible en s'appuyant sur des aspects statiques. Nos développements concernant l'intégration des systèmes d'équations différentielles ordinaires peuvent amener facilement le langage *sml* à supporter directement la modélisation des dispositifs dynamiques, par l'écriture directe des équations sous la forme différentielle (sachant que dans nos développements ceci se réalise en utilisant le concept de fonctions de fonctions). Il sera donc intéressant d'enrichir ce langage afin qu'il puisse permettre la description de systèmes différentiels.

Le langage *sml* pourrait devenir plus puissant si les aspects vectoriels tels qu'ils ont été présentés dans ce travail étaient étendus aux variables complexes. Normalement cette tâche ne devrait pas présenter de difficulté majeure que ce soit pour la dérivation ou pour l'implémentation effective. Les matrices, ou les structures des données à plusieurs dimensions restent à être implémentées dans le futur. Il serait aussi très utile pour le concepteur, et pour la modélisation en général, de disposer des opérations vectorielles telles que l'addition, la multiplication, l'inversion/transposition des matrices. Comme montré dans nos travaux, ces opérations pourraient facilement être implémentés grâce à la possibilité de surcharge les opérateurs pour les objets représentant les tableaux en C++.

Les aspects implémentés dans nos travaux concernant la Dérivation Automatique des méthodes numériques de résolution des systèmes d'équations différentielles est applicable seulement aux systèmes dynamiques continus. Une idée intéressante pourra être la prise en compte de la dynamique continue par morceaux et avec des événements discrets. Ceci est en effet un besoin très fort pour la simulation des actionneurs électromécaniques implémentés sous Reluctool [71].

L'évaluation des coefficients de Taylor représente un savoir-faire qui pourra être exploité non seulement pour la résolution d'un système d'équations différentielles, mais aussi pour un modèle de dimensionnement. Les coefficients Taylor peuvent être employés pour approximer par un polynôme la variation d'une variable de sortie autour d'un point d'entrée. Cela peut conduire à réaliser une analyse de sensibilités par des simulations de Monte Carlo beaucoup plus rapidement, car l'évaluation d'un polynôme pour plusieurs milliers de points de calcul est plus rapide que l'évaluation du modèle lui même pour une précision légèrement dégradée, si la distribution de points n'est pas trop étalée.

Un autre aspect concernant les méthodes numériques pourra être l'enrichissement de la bibliothèque de CADES avec d'autres méthodes numériques utiles pour la conception des dispositifs électriques. Des méthodes comme l'interpolation, l'identification de courbes ou des solveurs d'équations aux dérivées partielles pourront constituer des travaux intéressants à l'avenir. Par exemple la Dérivation Automatique des solveurs d'équations aux dérivées partielles est un défi et ceci pourra apporter beaucoup des bénéfices au niveau du dimensionnement avec SQP des dispositifs électromagnétiques modélisés par les équations de Maxwell.

La possibilité d'utiliser des fonctions dérivées en externe existant exclusivement sous ADOL-C représente un sujet controversé dans la communauté de Dérivation Automatique. Cet aspect s'est rendu indispensable pour nos travaux, lors de l'appel de fonction externes

Java dans les programmes des modèles de dimensionnement instrumentés avec cet outil. Nous encourageons donc tout développement de cette approche dans les autres outils de Dérivation Automatique.

A cette occasion nous introduisons les applications métier, développées au cours des dernières années au sein de notre équipe de recherche. Il s'agit notamment de MacMems [75] et Reluctool, qui gravitent autour de l'environnement CADES. Il est peut être intéressant d'intégrer dans ces logiciels directement la technique de Dérivation Automatique, dans la même façon que nous l'avons implémentée sous CADES.

Enfin, nous pourrons étendre nos travaux aux applications de modélisation métier, développées au cours des dernières années au sein de notre équipe de recherche. Il s'agit notamment de MacMems développé dans la thèse de Lalao Rakatoarion [75] et Reluctool dévéloppé dans la thèse de Bertrand du Pelloux [71] et de Phoung Thai Do, qui gravitent autour de l'environnement CADES. Il est peut être intéressant d'intégrer dans ces logiciels directement la technique de Dérivation Automatique, dans la même façon que nous l'avons implémentée sous CADES.

Glossaire

algorithme	Méthode de résolution d'un problème énoncée sous la forme d'une série d'opérations à effectuer, 13
algorithme déterministe	Un algorithme d'optimisation est déterministe, lorsque toute exécution conduit au même résultat, dans les mêmes conditions, 11
algorithme stochastique	Algorithme d'optimisation basé sur la technique mathématique appliquée à ce qui relève de l'aléatoire, à ce qui est statistique, 11
CADES	Component Architecture for Design of Engineering Systems - Logiciel développé au laboratoire G2ELab pour le dimensionnement des dispositifs électriques, 19
cahier des charges	L'ensemble des contraintes formulées pour le besoin (les variables de sortie), ainsi que l'ensemble de dégrées de liberté formulés pour les paramètres (variables d'entrée), 8
calcul formel	Procédé de transformation d'expressions mathématiques s'appuyant sur la manipulation de ces expressions sous leur forme symbolique. Applicable au calcul des dérivées, 16
CAO	Conception Assistée par Ordinateur, 10
compilateur	Programme informatique qui traduit un langage, le langage source, en un autre, appelé le langage cible, en préservant la signification du texte source, 19

composant logiciel — Terme général qui désigne un sous-ensemble logiciel complet, interfacé et documenté pour être appelé dans des méthodologies de développement comme DCOM, Corba, .Net ou JavaBean, 19

DA — La technique de Dérivation Automatique, 17

DF — La méthode des différences finies permettant d'évaluer les dérivées d'une fonction par une approximation de la limite mathématique de leur définition, 15

différentielle — Le produit de la dérivée d'une fonction f(x) par un accroissement arbitraire h de sa variable. Appliqué par le mode direct de dérivation, 29

dérivation à la main — Technique de dérivation nécessitant l'effort humain pour l'implémentation effective du programme de calcul des dérivées d'une fonction donnée, 17

facette — Fonctionnalité existante dans un composant logiciel de calcul déployable par un service, 19

fonction de fonction — Cas particulier d'une fonction définie non seulement avec des arguments représentant des variables, mais aussi avec des arguments représentant d'autre fonctions, 56

fonction externe — Fonction définie (au sens informatique) dans un langage autre que celui de base. Cet aspect est utilisé dans le contexte de la modélisation en utilisant un langage de description de modèles. Par exemple, nous proposons d'utiliser le langage *sml* de base. Ensuite, les langages C/C++ ou Java peuvent être utilisés pour la définition des fonctions externes. Dans le contexte de la DA, une fonction (appelée dans un programme) est externe si l'outil de DA ne dérive pas cette fonction suivant les mode de dérivation, mais appelle plutôt une routine externe responsable à fournir les dérivées dans le point où la fonction à été appelée initialement, 14

fonction interne — Fonction définie (au sens informatique) dans le même langage que celui de base d'un programme informatique. Notion existante dans le langage *sml*, 53

fonction objectif — Contrainte de minimisation/maximisation associée à une variable de sortie, 10

formule explicite — Formule qui permet l'évaluation directe du terme à gauche du signe =. Exemple : a=b*c, 13

formule implicite — Formule qui ne permet pas l'évaluation directe du terme à gauche du signe =. Exemple : a=b*c/a, 13

ICAr — Interfaces for Component Architecture - Le nom de la norme du composant logiciel de calcul dans le contexte de l'environnement logiciel CADES, 70

instrumentation — Procédé (dans le contexte de la DA s'appuyant sur la suchearge d'opérateurs) qui consiste à modifier le programme initial à dériver afin de le rendre dérivable par l'outil de DA. Ceci comprend en général la modification du type de toutes les variables actives et si c'est le cas, l'appel du programme entre le fonctions de début et fin de la trace de calcul, 36

JNI — Java Native Interface. Mécanisme permettant d'executer du code natif en Java, 84

machine virtuelle — Environnement d'exécution servant d'interface entre le système d'exploitation et des instructions en bytecode, 19

modèle de dimensionnement — Modèle qui exprime les variables de sortie en fonction d'entrées dans le but de l'exploiter ultérieurement dans l'optimisation, 8

mémoire dynamique — La mémoire qui est allouée au cours de l'exécution d'un programme informatique, 36

norme — Norme d'un composant logiciel. Standard qui assure la compatibilité d'un composant logiciel avec son environnement d'exécution, 19

paramètre de dimensionnement — Variable (d'entrée) soumise à une contrainte représentant un degré de liberté au dimensionnement d'un dispositif, 8

parseur — Entité d'analyse syntaxique d'un texte, réalisé par une séquence d'indicateurs (des mots), dans le but de déterminer sa structure grammaticale selon un ensemble des règles, 52

pilote — Fonction spéciale (dans le contexte de la DA utilisant une trace de calcul) permettant d'exploiter une trace de calcul dans différents buts comme : le calcul des dérivées du premier ordre ou d'ordre supérieur, calcul des coefficients de Taylor, calcul des dérivées d'un système implicite, etc, 79

pointeur	Variable contenant l'adresse d'une autre variable d'un type donné, 36
programme natif	Programme informatique en code natif (ou langage machine) composé d'instructions directement reconnues par un processeur (exemple de langages natives : C, C++, FORTRAN, Pascal, etc), 84
semi-analytique	Qui combine les formules analytiques avec les algorithmes, 13
service	Fonctionnalité implémentée dans un environnement logiciel permettant le déploiement d'une facette, 19
sml	System Modeling Language - Le langage de modélisation utilisé dans ce travail, 14
SQP	Sequential Quadratic Programming - La technique d'optimisation utilisée dans ce travail, 12
temps CPU	La quantité totale du temps qu'un programme informatique dépense à l'exécution des instructions du processeur, 29
trace de calcul	L'emplacement mémoire tampon utilisé par les outils de DA pour sauvegarder les valeurs des variables intermédiaires pour l'application ultérieure du mode inverse de dérivation, 38
variable active	Variable d'entrée, intermédiaire ou de sortie (dans le contexte de la DA) intervenant au calcul d'une sortie, 159
variable d'entrée	Variable qui n'est jamais calculée (dans le contexte de la modélisation) par une formule, fonction ou algorithme. Elle intérvient uniquement au calcul de variables de sortie, 8
variable de dimensionnement	Variable d'entrée ou sortie soumise à une contrainte en dimensionnement, 8
variable de sortie	Variable calculée (dans le contexte de la modélisation) par une formule, fonction ou algorithme à partir de variables d'entrées ou d'autres sorties, 8

variable dépendante	Variable qui dépend au moins d'une autre variable différente (dans le contexte de dérivation), 15
variable indépendante	Variable qui ne dépend que d'elle même (dans le contexte de dérivation) et sa propre dérivée est égale à l'unité, 15
variable intermédiaire	Variable intervenant (dans le contexte du dimensionnement) dans le calcul d'une variable de dimensionnement, n'étant soumise à aucune contrainte. Variable intervenant (dans le contexte de dérivation) dans le calcul des gradients, n'étant pas présente dans la matrice jacobienne, 27
variable interne	Variable (dans le contexte de la modélisation) qui notifie le compilateur qu'il s'agit d'une variable intermédiaire (voir glossaire). Notion existante dans le langage *sml*, 54

Bibliographie

[1] L. Allain. *Capitalisation et traitement des modèles pour la conception en génie électrique*. Mémoire de Thèse - Laboratoire d'Électrotechnique - Grenoble, Institut National Polytechnique de Grenoble, France, 2003.

[2] Cayuga Research Associates. Admat : Automatic differentiation toolbox for use with matlab. User Guide version 2.0, Cayuga Research Associates, LLC.

[3] E. Atienza. *Méthodologie et outil pour le dimensionnement*. Mémoire de Thèse - Laboratoire d'Électrotechnique - Grenoble, Institut National Polytechnique de Grenoble, France, 2003.

[4] E. Atienza, M. Parrault, F. Wurtz, V. Mazauric, and J. Bigeon. A methodology for the sizing and the optimization of an electromagnetic release. *IEEE Transactions on Magnetics*, Volume 36, No.4, July 2000.

[5] P. Aubert, N. Di Césaré, and O. Pironneau. Automatic differentiation in C++ using expression templates and application to a flow control problem. *Computing and Visualization in Science*, 3 :197–208, 2001.

[6] R. Barrio. Performance of taylor series method for odes/daes. *Applied Mathematics and Computation*, Elsevier, No. 163, 2005.

[7] F. L. Bauer. Computational graphs and rounding errors. *Siam, Numerical Analysis Journal*, Volume 11, 1974.

[8] M. Bücker and P. Hovland. Automatic differentiation. `http://www.autodiff.org/`, 2000, (consulté en 2009).

[9] B. M. Bell and J. V. Burke. Algorithmic differentiation of implicit functions and optimal values. In Christian H. Bischof, H. Martin Bücker, Paul D. Hovland, Uwe Naumann, and J. Utke, editors, *Advances in Automatic Differentiation*, pages 67–77. Springer, 2008.

[10] P. C. Benjamin. *Types and Programming Languages*. MIT Press., 2002.

[11] C. H. Bischof, B. Lang, and A. Vehreschild. Automatic differentiation for MATLAB programs. *Proceedings in Applied Mathematics and Mechanics*, 2(1) :50–53, 2003.

[12] C. H. Bischof, L. Roh, and A. Mauer. ADIC — An extensible automatic differentiation tool for ANSI-C. *Software–Practice and Experience*, 27(12) :1427–1456, 1997.

[13] C. Bischoff, A. Carle, P. Khademi, and A. Mauer. Adifor 2.0 : Automatic differentiation of fortran 77 programs. *IEEE Computational Science and Engineering*, Volume 3, No.3, 1996.

[14] H. M. Bücker, B. Lang, A. Rasch, and C. H. Bischof. From analytic to automated derivatives : a case study of the electrostatic potential. In *Algorithms and Software for Mobile Communications, Proceedings of the 10th Aachen Symposium on Signal Theory, Aachen, Germany, September 20–21, 2001*, pages 255–260, Berlin, 2001. VDE Verlag.

[15] H. M. Bücker, B. Lang, A. Rasch, and C. H. Bischof. Computing sensitivities of the electrostatic potential by automatic differentiation. *Computer Physics Communications*, 147(1–2) :720–723, 2002.

[16] H. M. Bücker and A. Vehreschild. Automatically differentiating a two-dimensional finite-difference time-domain program. In *Proceedings of the 16th Conference on the Computation of Electromagnetic Fields COMPUMAG 2007, Aachen, Germany, June 24–28, 2007*, pages 83–84, 2007.

[17] C. Chillet and J. Y. Voyant. Design-oriented analytical study of a linear electromagnetic actuator by means of a reluctance network. *IEEE Transactions On Magnetics*, Volume 37, No. 4, July 2001.

[18] S. B. Cooper. *Introduction to the Theory of Computation*. Chapman and Hall/CRC, 2004.

[19] G. F. Corliss and A. Griewank. Operator overloading as an enabling technology for automatic differentiation. Technical report, Center for Research on Parallel Computation, Rice University, Huston, U.S.

[20] C. Coutel. *Contribution méthodologique à la conception sous contraintes de dispositifs électromagnétiques*. Mémoire de Thèse - Grenoble Génie Electrique Laboratoire, Institut National Polytechnique de Grenoble, France, 1999.

[21] C. Coutel, F.Wurtz, and J. Bigeon. A comparative study of two methods for constrained optimization with analytical models dealing with implicit parameters. *IEEE Transactions On Magnetics*, Volume 35, No. 3, May 1999.

[22] J. A. de Vasconcelos. *Optimisation de forme de structures électromagnétiques*. Mémoire de Thèse, Ecole Centrale de Lyon, France, 1994.

[23] B. Delinchant, D. Duret, L. Estrabaut, L. Gerbaud, N. H. Hieu, B. Du Peloux, H. L. Rakotoarison, F. Verdiere, and F. Wurtz. An optimizer using the software component paradigm for the optimization of engineering systems. *COMPEL*, Volume 26, No. 2, 2007.

[24] B. Delinchant, G. Gruosso, and F. Wurtz. Two levels modeling for the optimization of electromagnetic actuators. *IEEE Transactions On Magnetics*, Volume 45, No. 3, 2009.

[25] B. Delinchant, F. Wurtz, and E. Atienza. Reducing sensitivity analysis time-cost of compound model. *IEEE, Transactions on Magnetics*, Volume 40, No. 2, March 2004.

[26] B. Delinchnat. *Un Environnement à base des Composants, Intégrant le Concepteur et ses Outils, pour des Nouvelles Méthodes de CAO.* Mémoire de Thèse - Laboratoire d'Électrotechnique - Grenoble, Institut National Polytechnique de Grenoble, France, 2003.

[27] Design processing technologies. `http://designprocessing.free.fr/`, 2006.

[28] E. Dezille. Utilisation de la différentiation de code dans le cadre su dimensionnement sous contraintes. Master's thesis, Institut National Polytechnique, Grenoble, France, 29 juin 2004.

[29] C. Divoux, O. Cugat, G. S. Reyne, J. Boussey-Said, and S. Basrour. Deformable mirror using magnet membranes : Application to adaptative optics in astrophysics. *IEEE Transactions On Magnetics*, Volume 35, No. 5, 1998.

[30] A. Dürrbaum, W. Klier, and H. Hahn. Comparison of automatic and symbolic differentiation in mathematical modeling and computer simulation of rigid-body systems. *Multibody System Dynamics*, Volume 7 No 4, May 2002.

[31] B. du Peloux, L. Gerbaud, F. Wurtz, V. Leconte, and F. Dorschner. Automatic generation of sizing static models based on reluctance networks for the optimization of electromagnetic devices. *IEEE Transactions On Magnetics*, Volume 42, No. 4, April 2006.

[32] E. Durand. *Magnétostatique.* Masson et Cie, 1968.

[33] D. Duret, L. Gerbaud, B. Du Peloux, F. Verdière, and H. L. Rakatoarison. A generator of software components dedicated to optimization of engineering systems. In *6th Workshop on Optimization and Inverse Problems in Electromagnetism, Sorrento, Italy*, 2006.

[34] P. Eberhard and C. Bischoff. Automatic differentiation of numerical integration algorithms. *Mathematics of Computation*, Volume 68, No. 226, April 1999.

[35] A. Elsheich, S. Noack, and W. Wiechert. Sensitivity analysis of modelica applications via automatic differentiation. In *6th International Modelica Conference*, volume Volume : 2, On page(s) : P669-P675, 2008.

[36] A. Elsheikh and W. Wiechert. Automatic sensitivity analysis of dae-systems generated from equation-based modeling languages. In Christian H. Bischof, H. Martin Bücker, Paul D. Hovland, Uwe Naumann, and J. Utke, editors, *Advances in Automatic Differentiation*, pages 235–246. Springer, 2008.

[37] P. Enciu, L. Gerbaud, and F. Wurtz. Automatic differentiation for sensitivity calculation in electromagnetism : Application on algorithms. In *13th Biennial IEEE Conference on Electromagnetic Field Computation (CEFC)*, May 2008.

[38] P. Enciu, H. L. Rakotoarison, B. Delinchant, F. Wurtz, L. Gerbaud, and L. Estrabaut. Cades : A software for analytical modeling and numerical algorithms interaction for

gradient based optimization. In *10th International Workshop on Optimization and Inverse Problems in Electromagnetism*, September 2008.

[39] P. Enciu, F. Wurtz, L. Gerbaud, and B. Delinchant. Automatic differentiation for electromagnetic models used in optimization. *COMPEL : The International Journal for Computation and Mathematics in Electrical and Electronic Engineering*, Volume 28, No.5 (à apparaître), 2009.

[40] V. Fischer. *Composants Logiciels pour le Dimensionnement en Génie Electrique. Application à la Résolution d'Equations Différentielles.* Mémoire de Thèse - Laboratoire d'Électrotechnique - Grenoble, Institut National Polytechnique de Grenoble, France, 2004.

[41] V. Fischer and L. Gerbaud. Corelab : a component-based integrated sizing environment. *COMPEL*, Volume 24, 2005.

[42] V. Fischer, L. Gerbaud, and F. Wurtz. Using automatic code differentiation for optimization. *IEEE, Transactions on Magnetics*, Volume 41, No. 5, May 2005.

[43] F. François. *Contribution de la modélisation floue à la conception en génie électrique.* Mémoire de Thèse - Institut National Polytechnique de Grenoble, France, 1994.

[44] A. Gentilhomme. *C.O.C.A.S.E. un système d'aide à la conception d'appareillage électriques.* Mémoire de Thèse - Institut National Polytechnique de Grenoble, France, 1991.

[45] L. Gerbaud. *GENTIANE : une plateforme pour la conception des ensembles machine-convertisseur-commande.* Rapport d'Habilitation à Diriger des Recherches, Institut National Polytechnique de Grenoble, France, 2000.

[46] L. Gerbaud and J. Bigeon. Elaboration d'un modèle mixant des approches symboliques et numériques pour le dimensionnement des convertisseurs statiques : problème particulier des intégrales semi-numériques des formules efficaces et moyennes. *Revue Internationale de Génie Electrique, version longue de l'article présenté à Numelec 2000*, Volume 4/2001, 1997.

[47] R. Giering. Tangent linear and Adjoint Model Compiler, users manual. Cambridge, MA, December 1997. Unpublished.

[48] J. C. Gilbert, G. Le Vey, and J. Masse. La différentiation automatique de fonctions représentées par des programmes informatiques. Rapport de Recherche, Institut de Recherche en Informatique et en Automatique, Unité de Recherche INRIA- Rocquencourt, Le Chesnay Cedex, France.

[49] N. I. M. Gloud and D. P. Robinson. A second derivative sqp method : local convergence. Rapport Technique, SFTC Rutherford Appleton Laboratory, U.K.

[50] A. Griewank, C. Bischof, G. Corliss, A. Carle, and K. Williamson. Derivative convergence of iterative equation solvers. *Optimization Methods and Software*, Volume 2, No. 3-4, 1993.

[51] A. Griewank, D. Juedes, and J. Utke. Algorithm 755 : Adol-c : A package for the automatic differentiation of algorithms written in c/c++. *ACM Transactions on Mathematical Software*, Volume 22, No. 2, 1996.

[52] A. Griewank and A. Walther. *Evaluating Derivatives, Principles and Techniques of Algorithmic Differentiation*. SIAM, Society for Industrial and Applied Mathematics, Philadephia, 2008.

[53] T. Hahn. Cuba : a library for multidimensional numerical integration. Max-Planck-Institut fur Physik, Munich, Germany.

[54] P. H. Hartel and L. A. V. Moreau. Formalizing the safety of java, the java virtual machine and java card. *ACM Computer Surveys*, Volume 33, No. 4, Dec 2001.

[55] L. Hascoet. Tapenade. on-line automatic differentiation engine. `http://tapenade.inria.fr:8080/tapenade/form.jsp`, Janv. 2009.

[56] L. Hascoet and V. Pascual. Tapenade 2.1 user's guide. Rapport Technique, Institut National de la Recherche en Informatique et en Automatique - INRIA, France.

[57] N. H. Hieu. *Méthodes et outils pour la conception de composants intégrés dans un réseau électrique embarqué*. Mémoire de Thèse - Grenoble Génie Electrique Laboratoire, Université Joseph Fourier, Grenoble, France, 2008.

[58] A. Ibsais and V. Ajjarapu. The role of automatic differentiation in power system analysis. *IEEE Transactions On Power Systems*, Volume 12, No. 2, 1997.

[59] M. Jerosolimski and M. Levacher. A new method for fast calculation of jacobian matrices : Automatic differentiation for power system simulation. *IEEE Transactions On Power Systems*, Volume 9, No. 2, 1994.

[60] W. Klein, A. Griewank, and A. Walther. Differentiation methods for industrial strength problems. In George Corliss, Christèle Faure, Andreas Griewank, Laurent Hascoët, and Uwe Naumann, editors, *Automatic Differentiation of Algorithms : From Simulation to Optimization*, Computer and Information Science, chapter 1, pages 3–23. Springer, New York, NY, 2002.

[61] M. Kofler. Maple : An introduction and references. Addison Wesley.

[62] G. Lewis, S. Barber, and E. Siegel. *Programming with Java IDL. Developing Web Applications with Java and CORBA*. John Wiley and Sons, 1999.

[63] S. Liang. *The Java Native Interface Programmer's Guide and Specification*. Addison-Wesley, 1999.

[64] D. Magot. *Méthodes et Outils Logiciels d'aide au Dimensionnement. Application aux composants Magnétiques et aux filtres passifs*. Mémoire de Thèse - Laboratoire d'Électrotechnique - Grenoble, Institut National Polytechnique de Grenoble, France, 2004.

[65] Mathworks. Matlab 7, programming fundamentals. The MathWorks, Inc. 3 Apple Hill Drive Natick, MA 01760-2098.

[66] L. H. De Medeiros, G. Reyne, and G. Meunier. Comparison of global force calculations on permanent magnets. *IEEE Transactions On Magnetics*, Volume 34, No. 5, 1998.

[67] T. Orfanogianni and R. Bacher. Using automatic code differentiation in power flow algorithms. *IEEE Transactions On Power Systems*, Volume 14, No. 1, 1999.

[68] T. Orfanogianni and R. Bacher. Steady-state optimization in power systems with series facts devices. *IEEE Transactions On Power Systems*, Volume 18, No. 1, 2003.

[69] V. Pascual and L. Hascoët. TAPENADE for C. In Christian H. Bischof, H. Martin Bücker, Paul D. Hovland, Uwe Naumann, and J. Utke, editors, *Advances in Automatic Differentiation*, pages 199–209. Springer, 2008.

[70] V. Pascual and L. Hascoet. Extension of tapenade towards fortran95. *in Bucker H.M., Corliss G., Hovland P., Naumann U., Norris B. (éditeurs) : Automatic Differentiation : Applications, Theory, and Tools, Lectures Notes in Computational Science and Engineering*, Dec Springer, 2005.

[71] B. Du Peloux. *Modélisation des actionneurs électromagnétiques par réseaux des réluctances. Création d'un outil métier dédié au prédimensionnement par optimisation.* Mémoire de Thèse - Laboratoire d'Électrotechnique - Grenoble, Université Joseph Fourier, France, 2006.

[72] R. J. Petti. Introduction to macsyma. Publishers, Inc.

[73] W. H. Press, S. A. Teukolski, W. T. Vetterling, and B. P. Flanery. Integration of ordinary differential equations. In *Numerical Recipes - The art of scientific Computing 3rd edition*, pages 899–946. Cambridge University Press, 2007.

[74] W. H. Press, S. A. Teukolski, W. T. Vetterling, and B. P. Flanery. Root finding and nonlinear sets of equations. newton-raphson method using derivative. In *Numerical Recipes - The art of scientific Computing 3rd edition*, pages 456–462. Cambridge University Press, 2007.

[75] H. L. Rakotoarison. *Méthode et Outil de Génération Automatique de modèle pour l'optimisation Fortement Contrainte des Microsystèmes Magnétiques.* Mémoire de Thèse - Laboratoire d'Électrotechnique - Grenoble, Université Joseph Fourier, France, 2007.

[76] J. Regnier. *Conception de systèmes hétérogènes en Génie Electrique par optimisation évolutionnaire multicritère.* Mémoire de Thèse, Institut National Polytechnique de Toulouse, France, 2003.

[77] U.S. Rice University, Houston. Open65. `http://www.hipersoft.rice.edu/open64`, 2003.

[78] B. Sareni. *Conception simultanée par optimisation des systèmes d'énergie électrique.* Rapport d'Habilitation à Diriger des Recherches, Institut National Polytechnique de Toulouse, France, 2006.

[79] D. Shiriaev, A. Griewank, and J. Utke. A user guide to ADOL–F : Automatic differentiation of Fortran codes, 1996.

[80] M. Sipser. *Introduction to the Theory of Computation 2nd edition*. PWS Publishing, 2006.

[81] R. Skjelvic. Automatic differentiation in java. Master's thesis, Department of Informatics, University of Bergen, Norway, 2001.

[82] E. I. Slusanschi. *Algorithmic Differentiation of Java Programs*. Mémoire de Thèse - Fakultat fur Mathematik, Informatik und Naturwissenschaften der Rheinisch-Westfalischen Technischen Hochschule - Aachen, Germany, 2008.

[83] Vacuumchmelze, soft magnetic cobalt-iron-alloys vacoflux 48, vacoflux 50, vacodur50, vacoflux 17. `http://www.vacuumschmelze.de`, 2009.

[84] Hsl, formerly the harwell subroutine library. `http://www.cse.clrc.ac.uk/nag/hsl/hsl.shtml`, 2009.

[85] Sun microsystems. `http://fr.sun.com/`, 2009.

[86] G. Reese (traduction en français de Hervé Soulard). *JDBC et Java, Guide du programmeur*. O'Reilly, 2001.

[87] J. Utke. Openad : Algorithm implémentation user guide. Technical memorandum, AML-MCS-TM-274, Mathematics ans Computer Science Division, Argonne National Laboratory, Argonne.

[88] T. T. Vu. *Problèmes Combinatoires et Modèles Multi-Niveaux pour la Conception Optimale des Machines Électriques*. Mémoire de Thèse, Ecole Centrale de Lille, France, 2009.

[89] A. Walther. Automatic differentiation of explicit runge-kutta methods for optimal control. *Computational Optimization and Applications*, Springer, vol. 36, July 2007.

[90] A. Walther and A. Griewank. Adol-c : A package for the automatic differentiation of algorithms written in c/c++. Technical report, Institute of Scientific Computing, Technische Universitat Dresden, Dresden, Germany.

[91] M. A. Weisfeld. *The Object-Oriented Thought Process*. Addison-Wesley Educational Publishers Inc., 2008.

[92] S. Wolfram. The mathematica book. Cambridge University Press, 4th Edition.

[93] F. Wurtz. *Une nouvelle approche pour la conception sous contraintes de machines électriques*. Mémoire de Thèse - Laboratoire d'Électrotechnique - Grenoble, Institut National Polytechnique de Grenoble, France, 1996.

[94] F. Wurtz. Statut et nature des processus de conception que nous utilisons en électrotechnique et possible rationalisation et automatisation. *L'Electrotechnique du futur*, No. 6/7, Juin/Juillet 2006.

[95] F. Wurtz. *Conceptions de la conception pour le génie électrique*. Rapport d'Habilitation à Diriger des Recherches, Institut National Polytechnique de Grenoble, France, 2008.

[96] F. Zaoui. Large electrical power systems optimization using automatic differentiation. In Christian H. Bischof, H. Martin Bücker, Paul D. Hovland, Uwe Naumann, and J. Utke, editors, *Advances in Automatic Differentiation*, pages 293–302. Springer, 2008.

Publications

Articles de revues internationales avec comité de relecture

Non paru

COMPEL — *Automatic Differentiation for Electromagnetic Models used in Optimization*
P. Enciu, F. Wurtz, L. Gerbaud, B. Delinchant - Volume 28 No. 5. **2009**.

Abstract : *Purpose : This paper deals with Automatic Differentiation, for the device sizing in electromagnetism by using gradient constrained optimization. CADES framework (Component Architecture for the Design of Engineering Systems), previously described, is presented here with extended features.*

Design/Methodology/Approach : The paper is subject to further usage for optimization of Automatic Differentiation (also named Algorithmic Differentiation) which is a powerful technique that computes derivatives of functions described as computer programs in a programming language like C/C++, FORTRAN.

Findings : Indeed, analytical modeling is well suited regarding optimization procedure, but the modeling of complex devices needs sometimes numerical formulations. This paper then reviews the concepts implemented in CADES which aim to manage the interactions of analytical and numerical modeling inside of gradient based optimization procedure. Finally, the paper shows that Automatic Differentiation has no limit for the input program complexity, or gradients accuracy, in the context of constrained optimization of an electromagnetic actuator.

Originality/value : Automatic Differentiated is employed for a large and complex numerical code computing multidimensional integrals of functions. Thus, the paper intends to prove the Automatic Differentiation capabilities in the context of electromagnetic device sizing by means of gradient optimization. The code complexity as also as the implications of Automatic Differentiation usage may stand as a good reference for the researchers in this field area.

Articles de conférences internationales avec comité de relecture

Paru

IEEE CEFC 2008 - Athènes. Grèce

Automatic Differentiation for Sensitivity Calculation in Electromagnetism : Application on Algorithms
P. Enciu, L. Gerbaud, F. Wurtz.

OIPE 2008 - Ilmenau Allemagne

Practical use of Automatic Differentiation for Optimization of Electromagnetic Devices
P. Enciu, L. Gerbaud, F. Wurtz.

OIPE 2008 - Ilmenau Allemagne

CADES : A software for analytical modelling and numerical algorithms interaction for gradient based optimization
P. Enciu, L. Gerbaud, F. Wurtz.

Soumis

IEEE COMPUMAG 2009 - Florianopolis. Brésil

Automatic Differentiation Applied for Optimization of Dynamical Systems
P. Enciu, L. Gerbaud, F. Wurtz.

Annexe A

Exemples d'application de la Dérivation Automatique

A.1 Utilisation de l'outil TAPENADE pour générer les programmes de calcul des dérivées

Nous considérons le programme FORTRAN corréspondant à la fonction de test décrite au chapitre II sur la figure II.2 :

```
C       FORTRAN:
        subroutine f(x, y)
           real x(2), y(2), a

        if(x(1).gt.2.0) then
            a = x(1) + x(2)
        else
            a = x(1) * x(2)
        endif

        do i=1,2
            a = a*x(i)
        enddo

        y(1) = a/x(2)
        y(2) = sin(a)

        end
```

FIGURE A.1 – Le programme FORTRAN de la fonction de test

L'outil de Dérivation Automatique TAPENADE, basé sur la technique source-to-source, génère le programme suivant qui calcule les dérivées en mode direct :

```
C         FORTRAN:
C         Generated by TAPENADE     (INRIA, Tropics team)
C         Tapenade 3.1 (r2754) - 01/12/2009 09:44C
C         Differentiation of f in forward (tangent) mode:
C         variations  of output variables: y
C         with respect to input variables: x
          SUBROUTINE F_D(x, xd, y, yd)
          IMPLICIT NONE
          REAL x(2), y(2), a
          REAL xd(2), yd(2), ad
          INTEGER i
          INTRINSIC SIN
          IF (x(1) .GT. 2) THEN
             ad = xd(1) + xd(2)
             a = x(1) + x(2)
          ELSE
             ad = xd(1)*x(2) + x(1)*xd(2)
             a = x(1)*x(2)
          END IF
          DO i=1,2
             ad = ad*x(i) + a*xd(i)
             a = a*x(i)
          ENDDO
          yd(1) = (ad*x(2)-a*xd(2))/x(2)**2
          y(1) = a/x(2)
          yd(2) = ad*COS(a)
          y(2) = SIN(x(2))
          END
C
```

FIGURE A.2 – Le programme FORTRAN de calcul de dérivées en mode direct de la fonction de test généré par l'outil TAPENADE

Le programme correspondant au mode inverse, aussi généré par TAPENADE est donné sur la figure A.3 :

```
C         FORTRAN:
C         Generated by TAPENADE     (INRIA, Tropics team)
C        Tapenade 3.1 (r2754)
C         Differentiation of f in reverse (adjoint) mode:
C        gradient, with respect to input variables: x y
C        of linear combination of output variables: x y
C
          SUBROUTINE F_B(x, xb, y, yb)
          IMPLICIT NONE
          REAL x(2), y(2), a
          REAL xb(2), yb(2), ab
          INTEGER i
          INTEGER branch
          INTRINSIC SIN
          REAL tempb
          IF (x(1) .GT. 2) THEN
              a = x(1) + x(2)
              CALL PUSHINTEGER4(0)
          ELSE
              a = x(1)*x(2)
              CALL PUSHINTEGER4(1)
          END IF
          DO i=1,2
              CALL PUSHREAL4(a)
              a = a*x(i)
          ENDDO
         ab = COS(a)*yb(2)
         yb(2) = 0.0
         tempb = yb(1)/x(2)
         ab = ab + tempb
         xb(2) = xb(2) - a*tempb/x(2)
         yb(1) = 0.0
         DO i=2,1,-1
              CALL POPREAL4(a)
              xb(i) = xb(i) + a*ab
              ab = x(i)*ab
         ENDDO
              CALL POPINTEGER4(branch)
         IF (branch .LT. 1) THEN
              xb(1) = xb(1) + ab
              xb(2) = xb(2) + ab
         ELSE
              xb(1) = xb(1) + x(2)*ab
              xb(2) = xb(2) + x(1)*ab
         END IF
         END
```

FIGURE A.3 – Le programme FORTRAN de calcul de dérivées en mode inverse de la fonction de test généré par l'outil TAPENADE

A.2 Utilisation de l'outil ADOL-C pour la dérivation des programmes

Nous détaillons dans ce paragraphe l'approche nécessaire pour dériver la fonction de test, illustrée sur la figure II.2, avec l'outil ADOL-C [51] implémentant la technique de

surcharge d'opérateurs utilisant une trace de calcul. L'approche se déroule en deux phases. La première phase représente l'instrumentation de la fonction. Dans la deuxième phase, les modes de dérivation souhaités s'appliquent sur la trace de calcul générée lors de l'instrumentation.

A.2.1 Instrumentation avec ADOL-C

En connaissant la liste complète d'entrées/sorties, les éléments suivants doivent être écrits (en respectant l'ordre) dans un programme C++ afin de préparer la dérivation de la fonction avec ADOL-C :

```
C++:
#include <adolc.h>    /*l'import de toutes les fonc. ADOL-C  */

int f(adouble *x_, adouble *y_){
adouble a;            /* variables indépendantes             */
adouble x[2];         /* activation des entrées              */
adouble y[2];         /* activation des sorties              */

trace_on(0);          /* début de la trace de calcul         */
    x[0] <<= x_[0];
    x[1] <<= x_[1];   /* spécification des indépendantes     */

    if(x[0] > 2) a = x[0] + x[1]; /* le code de la fonction */

    else          a = x[0] * x[1];


    for(int i=0; i<2; i++)
           a = a* x[i];

    y[0] = a/x[2];
    y[2] = sin(a);

     y[0] >>= y_[0]; /* spécification des dépendantes        */
     y[1] >>= y_[1];
trace_off();          /* fin de la trace de calcul           */
}
```

FIGURE A.4 – Instrumentation avec ADOL-C de la fonction de test

Les étapes du processus d'instrumentation sont :

1. Déclarer toutes les variables étant du type `adouble`. C'est le type de données pour lequel tous les opérateurs et les fonctions mathématiques de base sont surchargés. En s'appuyant sur la description de la classe qui surcharge les opérations mathématiques, donnée sur la figure II.10, *adouble* est le nom de cette classe choisi par les développeurs d'ADOL-C. En déclarant une variable `adouble`, elle devient *active* pour la Dérivation Automatique. Déclarer toutes les variables du programme comme *actives* assure la continuité du graphe de calcul. D'ici vient la certitude que la règle des fonctions composées employée au calcul ultérieur des dérivées partielles s'appliquera correctement.

2. Marquer le début de la trace de calcul ; ceci représente le début du code qu'on désire dériver ultérieurement. Cela se réalise en appelant la fonction spéciale `trace_on(ID)`, où ID représente l'identificateur de la trace de calcul (ici $ID = 0$). Sachant qu'un programme peut contenir plusieurs traces, il est recommandé que cet identificateur soit unique pour chaque trace. S'il existe au moins deux traces avec le même identificateur il est certain qu'une va écraser l'autre. Le début de la trace crée une nouvelle mémoire-tampon destinée à stocker tous les opérateurs et les opérandes associés.

3. Déclarer les variables indépendantes. Les variables indépendantes sont les variables en fonction desquelles on réalise la dérivation ultérieurement. Cela se réalise en utilisant l'opérateur spécial "<<=" où l'opérande du gauche est une variable *active*, donc une variable du type `adouble` et l'opérande du droit est une variable *inactive* du type `double`. En même temps cet opérateur initialise la variable active. Plus précisément c'est la variable membre `valuer` de la variable active qui est initialisée (à voir figure II.10).

4. Ecrire le programme dans la syntaxe C. Dans cette phase nous ne notifions aucune complication. Le programme est donné tel qu'il existe initialement.

5. Déclarer les variables dépendantes. Les variables dépendantes sont les variables pour lesquelles on réalise la dérivation ultérieurement. Il s'agit de toutes les variables de sortie de la fonction. Cela se réalise en utilisant l'opérateur spécial ">>=" où l'opérande de gauche est une variable *active*, donc une variable du type `adouble` et l'opérande droite est une variable *inactive* du type `double`. En même temps, cet opérateur initialise la variable inactive associée. Plus précisément, c'est la variable membre `valuer` de la variable active qui initialise la variable inactive.

6. Marquer la fin de la trace de calcul. Ceci représente la fin du code qu'on désire dériver ultérieurement. Cela se réalise en appelant la fonction spéciale `trace_off()`. Cette fonction correspond à la fermeture de la mémoire-tampon créée auparavant.

A.2.2 Réutilisation de la trace de calcul

L'instrumentation, telle que nous l'avons présentée dans le paragraphe A.2.1, ne calcule pas les dérivées partielles. Elle a pour but juste de créer la mémoire tampon de la trace de calcul. L'utilisation d'ADOL-C pour le calcul effectif des dérivées exige donc des appels à des fonctions spéciales supplémentaires (*des pilotes*) qui font possible la réutilisation de la trace. La réutilisation peut se faire dans le but de calcul de la fonction (dérivées d'ordre 0), de calcul des dérivées en mode direct ou inverse et bien une liste vaste d'autres fonctionnalités implémentées sous ADOL-C que le lecteur peut la trouver en [51][90]. Nous présentons seulement dans les paragraphes qui suivent la réutilisation en vue de calcul, de mode directe et de mode inverse de dérivation.

A.2.2.1 La réutilisation de la trace de calcul pour l'évaluation de la fonction

Cette approche peut être utile pour des appels successifs à la fonction. Il est plus convenable en termes de mémoire et de rapidité de réutiliser une trace que de régénérer une à chaque appel. Dans la figure A.5, nous illustrons le pilote implémenté sous ADOL-C qui permet ceci :

```
C/C++:
zos_forward(        //Zero Order Forward Mode
         id,        //identificateur de la trace
         m ,         //nombre des variables dépendantes
         n ,         //nombre des variables indépendantes
         k ,         //préparation pour le mode inverse
         *x,         //les valeurs des variables indépendantes
         *y,         //les valeurs des variables dépendantes
);
```

FIGURE A.5 – Réutilisation de la trace de calcul pour l'évaluation de la fonction

Les paramètres de cette fonction sont peut être évidentes à la premier vue. Cependant, nous les détaillons ici pour effacer toute trace d'ambiguïté. La liste de ces arguments est donnée dans l'énumération suivante :

- ID - à remplacer par l'identificateur de la trace. Pour le programme de la fonction de test $ID = 0$.
- m - le nombre de variables dépendantes. Ici $m = 2$. Si par une erreur quelconque cet argument n'est pas conforme avec le nombre des variables dépendantes effectivement stockées dans la trace, ADOL-C renvoie évidemment un message d'erreur.
- n - le nombre des variables indépendantes. Ici $n = 2$.
- k - constante positive qui prépare le mode inverse. L'utilisateur doit donc connaitre en avance le mode de dérivation utilisé ultérieurement.
- $*x$ - l'adresse d'un ensemble de valeurs d'entrée.
- $*y$ - l'adresse où ADOL-C sortira les valeurs de sortie.

A.2.2.2 La réutilisation de la trace de calcul pour l'évaluation des dérivées en mode direct

De la même façon, nous détaillons ici la réutilisation de trace pour évaluer les dérivées de premier ordre en mode direct. Nous avons présenté au paragraphe II.2.2.1 que mode direct évalue efficacement les dérivées partielles de toutes les sorties en fonction d'une entrée. C'est ce qu'on appelle le mode direct scalaire (une seule entrée). La fonction donnée sur la figure A.6 appliquée sur la trace $ID = 0$ dérive notre programme en mode direct.

```
C/C++:
fos_forward(        //First Order Forward Mode
         id,        //identificateur de la trace
         m ,        //nombre des variables dépendantes
         n ,        //nombre des variables indépendantes
         k ,        //préparation pour le mode inverse
         x[n],      //les valeurs des variables indépendantes
         xt[n],     //le vecteur tangent
         y[m],      //les valeurs des variables dépendantes
         yt[m],     //les valeurs des dérivées
)
```

FIGURE A.6 – Réutilisation de la trace de calcul pour l'évaluation des dérivées de premier ordre en mode direct

Les premiers cinq paramètres de cette fonction ont la même signification que ceux évoqués pour la fonction `zos_forward`. Nous détaillons les derniers trois dans l'énumération suivante :

- $xt[n]$ - est vecteur tangent des entrées. Il s'agit du vecteur $\dot{x}$ de l'équation II.5 de la règle des fonctions composées. En général, ce vecteur est la i^{eme} base Cartésienne où i est l'indice de la variable d'entrée par rapport à laquelle on veut dériver. Par exemple pour calculer $\frac{\partial y_{0,1}}{\partial x_0}$ [1] le vecteur tangent est `xt = {1,0}`.
- $y[n]$ - l'adresse où ADOL-C sortira les valeurs de sortie. Nous rappelons que le mode direct de dérivation calcule les dérivées et les valeurs de la fonction en parallèle.
- $yt[m]$ - l'adresse où ADOL-C sortira les valeurs des dérivées de toutes les sorties par rapport à l'entrée spécifié en xt.

A noter que l'outil ADOL-C donne aussi la possibilité de calcul la matrice jacobienne complète en mode direct. Pour ceci l'utilisateur doit utiliser le mode direct vectoriel de dérivation, `fov_forward`, documenté en [51].

A.2.2.3 La réutilisation de la trace de calcul pour l'évaluation des dérivées en mode inverse

Nous détaillons ici la réutilisation de trace pour évaluer les dérivées de premier ordre en mode inverse. Nous avons présenté au paragraphe II.2.2.2 que mode inverse évalue efficacement les dérivées partielles d'une sortie en fonction de toutes les entrées (un gradient). C'est ce qu'on appelle le mode inverse scalaire (une seule sortie). Nous rappelons que le mode inverse nécessite un calcul de la fonction. La figure A.7 illustre la routine responsable avec le mode inverse. Pour que la fonction `fos_reverse` puisse fonctionner il faut donc d'abord appeler la fonction `zos_forward` de la figure A.5 avec l'argument $k = 1$.

1. Les système d'indexation est celui utilisé dans le langage C/C++. Il commence à partir de l'indice 0.

```
C/C++:
fos_reverse(        //First Order Reverse Mode
         id,        //identificateur de la trace
         m ,        //nombre des variables dépendantes
         n ,        //nombre des variables indépendantes
         u[m],      //les valeurs adjointes des sorties
         z[n],      //les valeurs adjointes des entrées
)
```

FIGURE A.7 – Réutilisation de la trace de calcul pour l'évaluation des dérivées de premier ordre en mode inverse

L'ensemble de premiers trois paramètres de la fonction `fos_reverse` a la même signification que pour la fonction `zos_forward`. Nous détaillons dans l'énumération suivante la signification de deux derniers paramètres.

1. $u[m]$ - est vecteur adjoint des sorties. Il s'agit du vecteur $\bar{y}$ de l'équation II.9 de la règle des fonctions composées. En général, ce vecteur est la i^{eme} base Cartésienne où i est l'indice de la variable de sortie pour laquelle on veut évaluer les dérivées partielles de toutes les entrées. Par exemple pour calculer $\frac{\partial y_1}{\partial x_{0,1}}$ le vecteur adjoint est `u = {0,1}`.
2. $yt[m]$ - l'adresse où ADOL-C sortira les valeurs des dérivées de la sortie spécifié en u par rapport à toutes les entrées.

Pour calculer la matrice jacobienne complète, l'outil ADOL-C implémente une version vectorielle du mode inverse appelée `fov_reverse`.

A.3 Le concept de fonction dérivée en externe implémenté sous ADOL-C

Une fonction externe ADOL-C [2] peut être appelée dans une trace de calcul grâce à une technique récemment implémentée (en cours de développement en 2007). L'utilisateur a la responsabilité de spécifier sa fonction de calcul et de calcul des dérivées. Cette fonction n'est pas dérivée par ADOL-C. Les initialisations préliminaires sont également de la responsabilité de l'utilisateur. La figure suivante illustre la liste complète des instructions préalables permettant faire fonctionner un code de dérivation en mode direct appelant une fonction dérivée en externe.

2. Une fonction externe *ADOL-C* est une fonction appelée dans une trace de calcul des dérivées ADOL-C qui doit fournir sa valeur et ses dérivés partielles dans tous le points de son domaine de définition. Ces fonctions ont une signature spécifique et elles sont d'un réel intérêt au cas où son code n'est pas disponible en C/C++

```
C++:
int ext_fct(int n, double *xin, int m, double *yout); /* la fonction externe*/
01
int dext_fct(int n, double *x, double **X, int m, double *y, double **Y); /*la
fonction qui évalue les dérivées de la fonction externe en mode direct*/

ext_diff_fct *edf;   /* la structure caractérisant la fonction externe        */
02
reg_ext_fct(ext_fct);/* initialisation avec le pointeur de la fonction externe*/

edf->dp_x = new double[n]; /*allocation des variables du domaine de définition*/
03
edf->dp_y = new double[m]; /*allocation des variables du domaine de valeurs   */
...
04
edf->fov_forward = dext_fct; /*Spécification de la fonction externe ADOL-C*/
...
trace_on(ID);          /* début de la trace de calcul       */
     ... <<= ...;      /* spécification des indépendantes   */
          ...
05
 call-ext_fct(edf, n, x, xa, m, y, ya); /*l'appel de la fonction dans la trace*/
         ...
     ... >>= ...;      /* spécification des dépendantes     */
trace_off();           /* fin de la trace de calcul         */
...
```

FIGURE A.8 – L'appel d'une fonction externe ADOL-C dans une trace de calcul

A.3.1 Spécification des éléments d'une fonction dérivée en externe ADOL-C

ADOL-C demande une fonction externe en deux fonctions. La première réalise le calcul (`ext_fct` dans la figure A.8), la seconde réalise le calcul de sérivées (`dext_fct` dans la figure A.8).

A noter que le mode de dérivation doit être connu à l'avance afin d'initialiser la structure de la fonction externe (`edf` sur la figure A.8). Nous proposons sur la figure A.8 une fonction externe qui réalise la dérivation en mode direct vectoriel (`fov_forward`[3]). Pour une dérivation en mode inverse la fonction pointeur change sa signature.

Le rôle d'ADOL-C est d'appliquer la règle des fonctions composées à partir du bas du graphe de calcul (à partir des variables indépendantes) jusque au sommet (jusque aux variables dépendantes) afin d'évaluer les dérivés partielles du programme. Ce graphe inclut aussi la fonction dérivées en externe.

La liste de significations de tous les paramètres permettant de construire une fonction externe ADOL-C, donnée ci-dessous, pourra améliorer la compréhension de ces aspects.

1. Au niveau de la fonction de calcul ($ext\_fct : \mathbb{R}^n \rightarrow \mathbb{R}^m$:
   - $param$: `int n` - la taille du domaine de définition de la fonction ;
   - $param_{in}$[4] : `double *x` $\in \mathbb{R}^n$ - le vecteur de valeurs du domaine de définition de

3. fov_forward est l'acronyme de *F*irst *O*rder *V*ectorial differentiation in *forward* mode qui peut se traduire par *dérivation vectorielle de premier ordre en mode direct*

4. Un $paramtre_{in}$ représente une adresse (valide) passée en paramètre à une fonction quelconque qui contient des valeurs qui peuvent être utilisées dans la définition de la fonction. Pour un comportement normal, il n'est pas forcement nécessaire de modifier les valeurs à l'adresse en question. Cette notion a du sens dans les langages permettant le passage des variables par référence ou par adresse (e.g. C/C++, Java, etc...).

la fonction ;
- $param$: `int m` - la taille du domaine de valeurs de la fonction ;
- $param_{out}$:[5] : `double *y` $\in \mathbb{R}^m$ - les valeurs de la fonction correspondantes au point x ;

2. Au niveau de la fonction de calcul des dérivées ($dext\_fct : \mathbb{R}^n \rightarrow \mathbb{R}^{m \times n}$, supposée être appelée dans une trace de calcul contenant $n^* \geq 1$ variables indépendantes :
   - $param$: `int n` - la taille du domaine de définition de la fonction ;
   - $param_{in}$: `double *x` $\in \mathbb{R}^n$ - le vecteur de valeurs du domaine de définition de la fonction ;
   - $param_{in}$: `double **X` $\in \mathbb{R}^{n \times n^*}$ - la matrice tangente ;
   - $param$: `int m` - la taille du domaine des valeurs de la fonction ;
   - $param_{out}$:`double *y` $\in \mathbb{R}^m$ - les valeurs de la fonction correspondantes au point x ;
   - $param_{out}$: `double **Y` $\in \mathbb{R}^{m \times n^*}$ - la matrice jacobienne calculée au point x ;

A.3.2 Exemple d'utilisation d'une fonction dérivée en externe

Nous allons détailler cette approche autour de l'exemple A.1. Après le retour de la fonction externe ADOL-C, le reste du programme continue à être dérivé en appliquant normalement la règle générale des fonctions composées afin d'avancer jusqu'au sommet du graphe de calcul.

Exemple A.1
Supposons la fonction externe ADOL-C $f : \mathbb{R}^2 \rightarrow \mathbb{R}^2$, définie comme :

$$f(x, y) = [f_0 = 2 \cdot x^2 + 1,\ f_1 = 2 \cdot y^2 + 1]$$

La matrice jacobienne de cette fonction, $JAC_f : \mathbb{R}^2 \rightarrow \mathbb{R}^{2 \times 2}$, peut se formuler facilement, comme suit :

$$JAC_f(x, y) = \begin{pmatrix} \dfrac{\partial f_0}{\partial x} = 4 \cdot x & \dfrac{\partial f_0}{\partial y} = 0 \\ \dfrac{\partial f_1}{\partial x} = 0 & \dfrac{\partial f_1}{\partial y} = 4 \cdot y \end{pmatrix}$$

Supposons une trace de calcul des dérivées ADOL-C contenant trois variables indépendantes, $a,\ b,\ c$ et deux variables dépendantes $v,\ w$ qui se calculent comme :

$$[v, w] = f(a^2 + c, b^2 + c).$$

Supposons que la trace appelle seulement la fonction externe f. La figure suivante montre le graphe de calcul des dérivées partielles de v et w en mode direct.

5. Un $paramtre_{out}$ représente une adresse (valide) passée en paramètre à une fonction quelconque qui contient des valeurs sans aucun sens qui ne sont pas utilisées dans la définition de la fonction. Pour un comportement normal, il est forcement nécessaire de modifier le(s) valeur(s) à l'adresse en question.

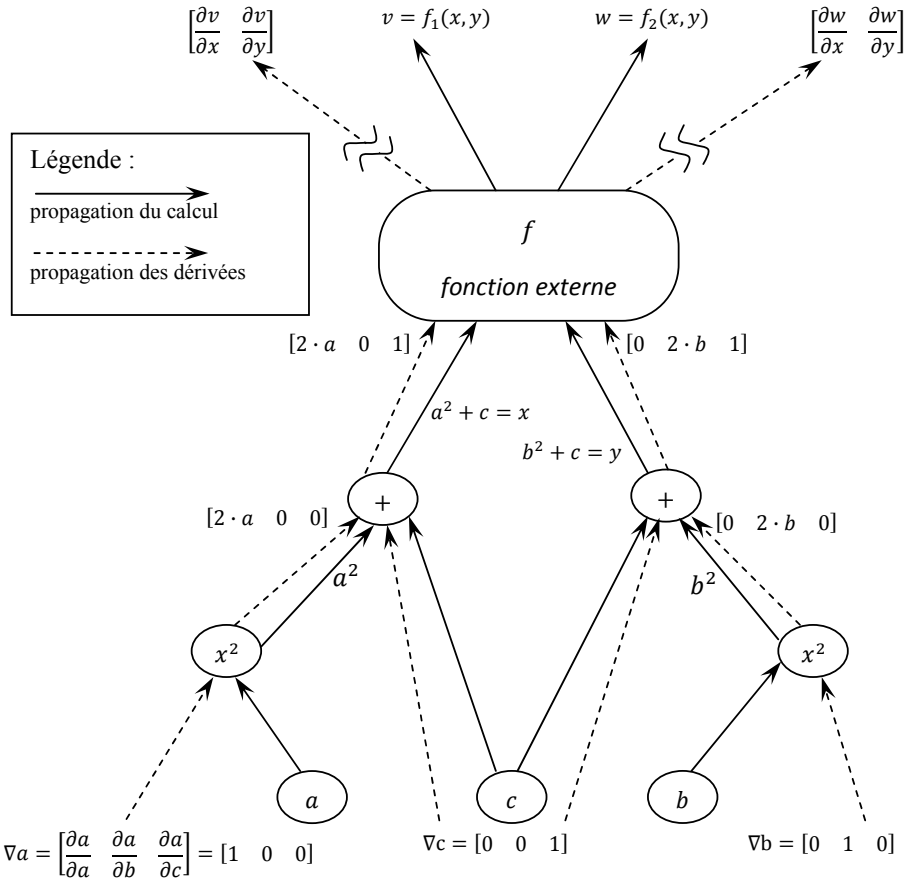


FIGURE A.9 – Le graphe de propagation du calcul et des dérivés pour un exemple d'utilisation d'une fonction externe

Après l'appel de la fonction qui renvoie la matrice jacobienne, JAC_f, le graphe de calcul de dérivés est localement interrompu, le lien n'étant pas réalisé entre les dérivées partielles de la variable de retour par rapport aux arguments de la fonction externe et les dérivées partielles par rapport aux variables indépendantes de la trace de calcul :

$$\begin{pmatrix} \frac{\partial v}{\partial a} & \frac{\partial v}{\partial b} & \frac{\partial v}{\partial c} \end{pmatrix} \neq \begin{pmatrix} \frac{\partial v}{\partial x} & \frac{\partial v}{\partial y} \end{pmatrix} \tag{A.1}$$

Afin de régler cet inconvénient, ADOL-C fournit automatiquement la matrice des dérivées partielles des arguments de la fonction externe par rapport aux variables indépendantes de la trace de calcul. Cette matrice s'appelle la matrice tangente et elle est présente dans la liste d'arguments de la fonction de calcul de dérivées (l'argument X de la fonction `dext_fct` dans la figure A.8) :

$$X = \begin{pmatrix} \frac{\partial x}{\partial a} & \frac{\partial x}{\partial b} & \frac{\partial x}{\partial c} \\ \frac{\partial y}{\partial a} & \frac{\partial y}{\partial b} & \frac{\partial y}{\partial c} \end{pmatrix} = \begin{pmatrix} 2 \cdot a & 0 & 1 \\ 0 & 2 \cdot b & 1 \end{pmatrix}$$

Cependant, l'inégalité A.1 devient égalité en multipliant son membre droit avec la matrice tangente. Nous rappelons que cette opération reste à la charge de l'utilisateur :

$$\begin{pmatrix} \frac{\partial v}{\partial a} & \frac{\partial v}{\partial b} & \frac{\partial v}{\partial c} \end{pmatrix} = \begin{pmatrix} \frac{\partial v}{\partial x} & \frac{\partial v}{\partial y} \end{pmatrix} \cdot \begin{pmatrix} \frac{\partial x}{\partial a} & \frac{\partial x}{\partial b} & \frac{\partial x}{\partial c} \\ \frac{\partial y}{\partial a} & \frac{\partial y}{\partial b} & \frac{\partial y}{\partial c} \end{pmatrix}$$

Annexe B

Elements architecturaux de l'environnement logiciel CADES

B.1 Facettes disponibles dans le composant de calcul implémentant la version 2.0 de la norme *ICAr*

B.1.1 La facette de calcul de gradients sélectifs

En optimisation, il se peut que certaines variables de sortie ne soient pas soumises sous contraintes. Les valeurs des dérivées partielles de ces sorties n'interviennent donc dans les itérations d'optimisation des algorithmes basés sur le calcul des gradients. Le calcul de la matrice jacobienne complète, donnée par la facette de calcul des gradients, introduite dans le paragraphe IV.2.2.2, dévient inefficace du moment où il n'existe pas de moyen pour faire le tri entre les dérivées partielles utiles ou non. Afin d'éliminer cet inconvénient, une facette, appelée `SelectiveJacobianSolver`, créée suite des travaux de thèse de H. L. Rakotoarison [75], donne la possibilité d'accéder à la valeur d'une certaine dérivée partielle comme le montre la figure B.1. Ceci n'est pas suffisant, puisque la séquence du calcul d'une certaine dérivée partielle peut impliquer de recalculer ses dépendances. Grâce à une stratégie de sélectivité (tri entre ce qui est utile ou non) cette facette utilise des informations supplémentaires concernant les sorties déjà calculées. Ceci augmente énormément l'efficacité du calcul des gradients.

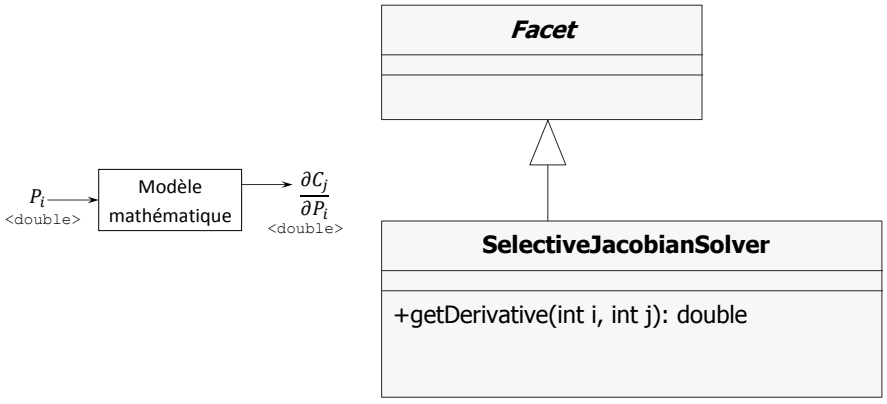


FIGURE B.1 – La facette de calcul des gradients sélectifs

B.1.2 La facette d'unités

La facette d'unités est capable d'affecter les unités de mesure, définies par le concepteur au niveau du modèle mathématique, aux certaines variables spécifiées. Les entrées de cette facette sont seulement celles variables spécifiées par l'utilisateur à porter des unités de mesure. Les sorties sont les chaines de caractères dénotant le nom de l'unité. Cet aspect à été implémenté lors des travaux de master de A. Vital.

Cette facette n'analyse pas sémantiquement les unités afin d'être capable de réaliser des opérations mathématiques et finalement de simplification. Sa tâche est seulement d'affectation.

B.1.3 La facette d'initialisation

Le modèle mathématique peut contenir des fonctions implémentant des algorithmes numériques qui exigent en entrée des valeurs initiales afin de converger vers des valeurs finales. C'est notamment le cas des fonctions implémentant des algorithmes Newton pour la résolution des systèmes des équations implicites ou les algorithmes d'intégration d'équations différentielles. Il est très efficace, du point de vue de l'expressivité d'un modèle de dimensionnement, d'utiliser la valeur initiale et finale sous un seul nom pour la variable en question (qui est spécifié explicitement par l'utilisateur). De plus, grâce à cette facette, il est possible de connaitre en avance que certains paramètres, i.e. les paramètres dénotant les valeurs initiales, ne sont pas dimensionnables et donc de les exclure du calcul des dérivées partielles effectué par la facette de calcul des gradients. Cette approche est illustrée sur la figure B.2, où une variable d'initialisation est du type `IntializingParameter`, une structure qui pressente des attributs dénotant les valeurs initiales et finales.

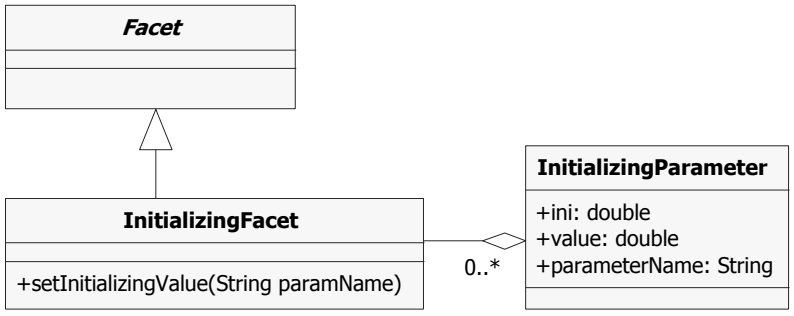


FIGURE B.2 – La facette d'initialisation

La gestion des variables d'initialisation est réalisée par la facette d'initialisation qui prend en entrée la valeur initiale d'une variable. Cette facette n'effectue pas le calcul du modèle mathématique puisque cette tâche est exclusivement attribuée aux facettes de calcul ou de calcul des gradients. La récupération des valeurs finales des variables d'initialisation peut se faire à posteriori d'un calcul complet du modèle de dimensionnement.

B.1.4 La facette de configuration

Dans la même logique que pour les paramètres d'initialisation, il se peut exister des situations quand les modèles mathématiques exigent en entrée des paramètres permettant de configurer certains algorithmes numériques. Il s'agit notamment des constantes telles qu'une précision de convergence, d'un pas d'intégration minimal ou maximal ou un nombre maximal d'itérations. Normalement ces paramètres sont dans la plupart des cas non dimensionnables. Leurs dérivées partielles ne sont pas donc évaluées par la facette de calcul des gradients spécifiée au paragraphe IV.2.2.2. Ces constantes peuvent prendre des valeurs par défaut ou elles peuvent bien se transformer dans des variables dépendantes. L'idée de la facette de configuration est d'affectations les paramètres de configuration par des valeurs par défaut ou des valeurs définies ou calculées, tout cela à la demande de l'utilisateur d'un service s'appuyant sur cette facette.

B.1.5 La facette de visualisation

Dans la modélisation, le concepteur utilise un jeu des paramètres géométriques et/ou physiques pour définir un jeu des performances. Ces performances peuvent être bien exprimées en terme de dimensions (distances ou angles) du dispositif, évaluées à partir des paramètres géométriques. Ainsi, une convergence efficace d'un algorithme d'optimisation peut s'atteindre à condition que toutes les dimensions géométriques, explicitées ou non dans le modèle mathématique, soient positives au cours d'itérations. Vue que le concepteur explicite dans la plupart des cas seulement une petite partie de toutes les distances possibles intervenant dans la géométrie du dispositif, il lui revient très difficile à trouver les incohérences de nature géométrique, sans une représentation visuelle de la forme du dispositif. CADES propose une stratégie pour solutionner cet aspect reposant sur la facette de visualisation. Muni d'un jeu minimal des paramètres géométriques d'un composant de calcul, le concepteur peut définir une vue ou dessin du dispositif ou d'une partie considérée comme substantielle. La facette de visualisation s'en charge avec l'actualisation de la vue géométrique en fonction des valeurs courantes des toutes les paramètres qui la définissent. La figure B.3 illustre le modèle orienté-objet de cette facette.

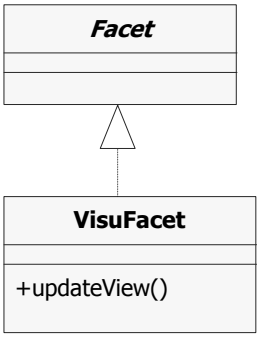


FIGURE B.3 – La facette de visualisation

L'expérience montre que cette facette se rend utile pour la vérification d'un cahier des charges pour les paramètres géométriques, ou pour la validation d'une solution d'optimisation. Son utilité principale reste dans l'exploration de la forme des dispositifs au cas des optimisations défectueuses ou sans succès.

B.2 L'interface graphique du *Component Optimizer*

B.2.1 Le menu principal

Le menu principal du *Component Optimizer* permet de :
- charger un composant de calcul implémentant la norme *ICAr*.
- choisir un algorithme d'optimisation.
- charger un cahier des charges.

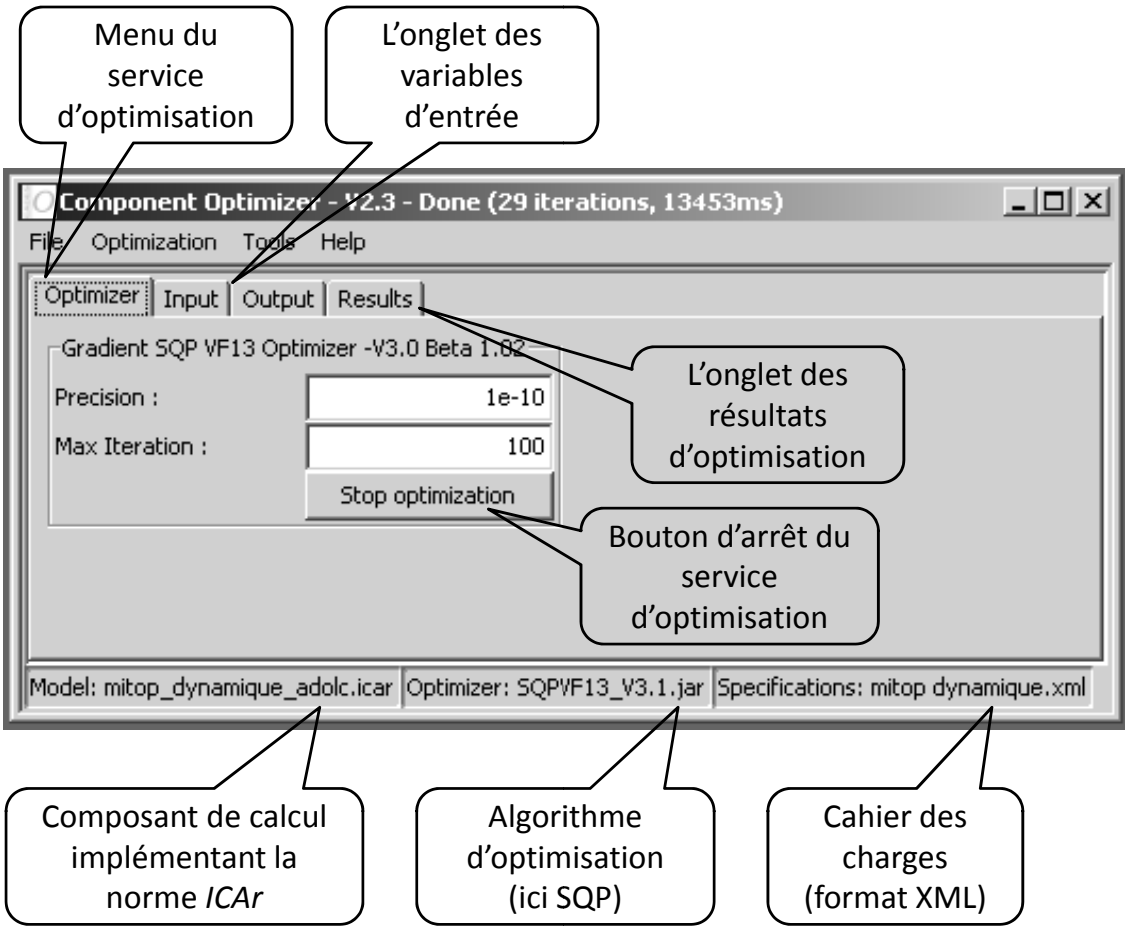


FIGURE B.4 – L'interface graphique du *Component Optimizer*

Lors du chargement de ces composants, le menu met à disposition les interfaces suivantes :
- un onglet pour le paramétrage du service d'optimisation ; cet onglet est spécifique à chaque algorithme d'optimisation. Par exemple, pour un algorithme SQP, il propose

le paramétrage du nombre maximal d'itérations (en général une centaine suffit), et la précision de résolution du cahier des charges. Pour un algorithme génétique, en général ici on retrouve le nombre maximal de populations à générer, ou le nombre maximal d'évaluations du modèle de dimensionnement.

– un onglet pour la gestion de contraintes sur les variables d'entrée/sortie.
– un onglet pour l'affichage des résultats d'optimisation.

B.2.2 Les onglets de contraintes de résultats d'optimisation

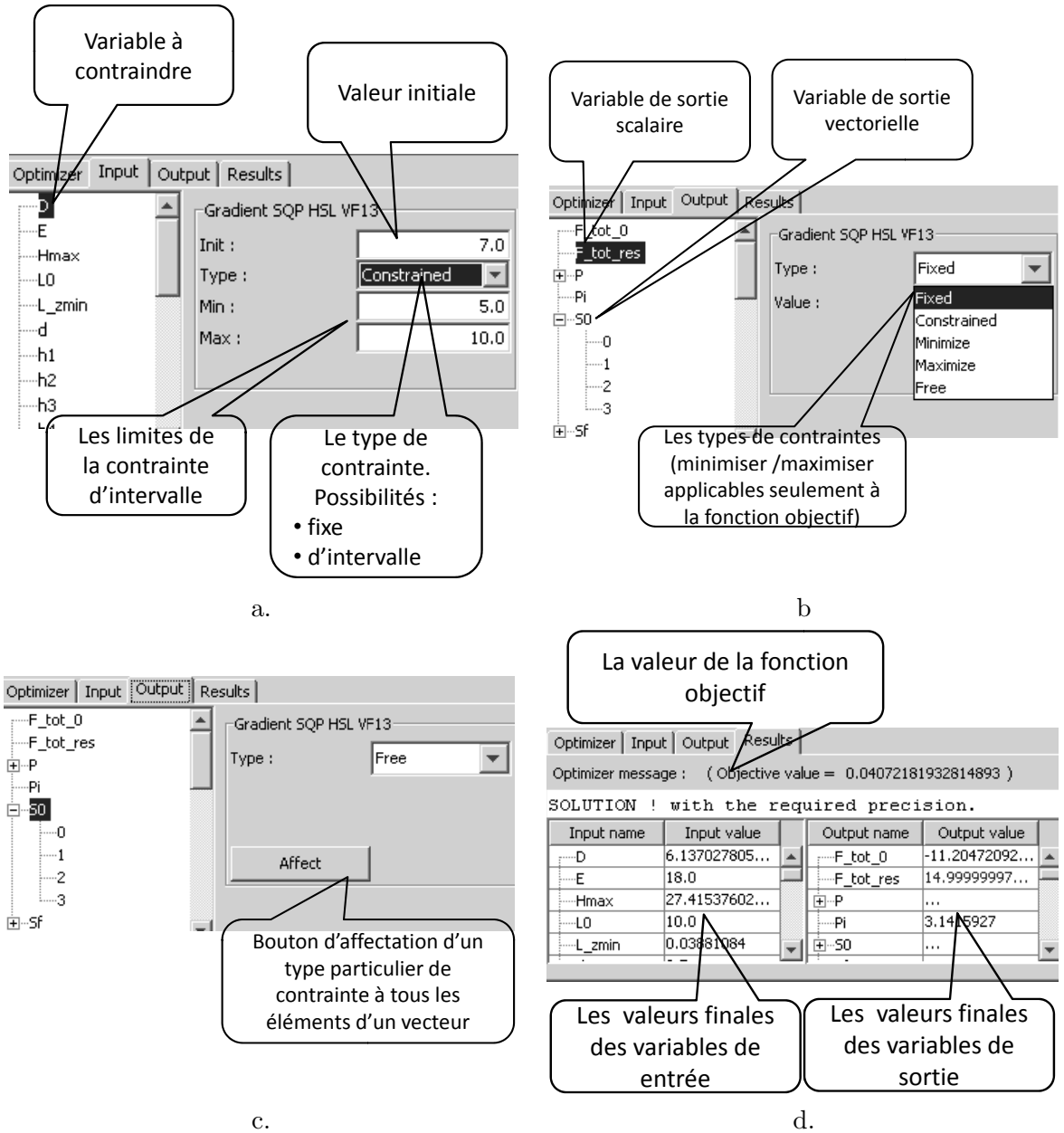


FIGURE B.5 – Les onglets du *Component Optimizer* pour la gestion des contraintes ; a. l'onglet d'entrées ; b. l'onglet de sorties avec l'illustration d'une variable scalaire ; c. l'onglet de sorties avec l'illustration d'une variable vectorielle ; d. l'onglet de résultats d'optimisation

Les onglets de contraintes des variables d'entrées/sorties permettent la formulation des spécifications d'optimisation. L'onglet de résultats dévient active après la convergence de l'algorithme d'optimisation et il illustre les valeurs finales des variables d'entrées/sorties.

Le *Component Optimizer* assure aussi l'écriture dans un fichier des valeurs des variables des modèles de dimensionnement à chaque itération d'optimisation. Ce fichier est ensuite exploitable dans un *PostProcesseur* d'optimisation afin d'analyser ou vérifier les variations des variables.

B.3 Les services implémentées dans le *Component Calculator*

B.3.1 Les services d'affichage et traçage

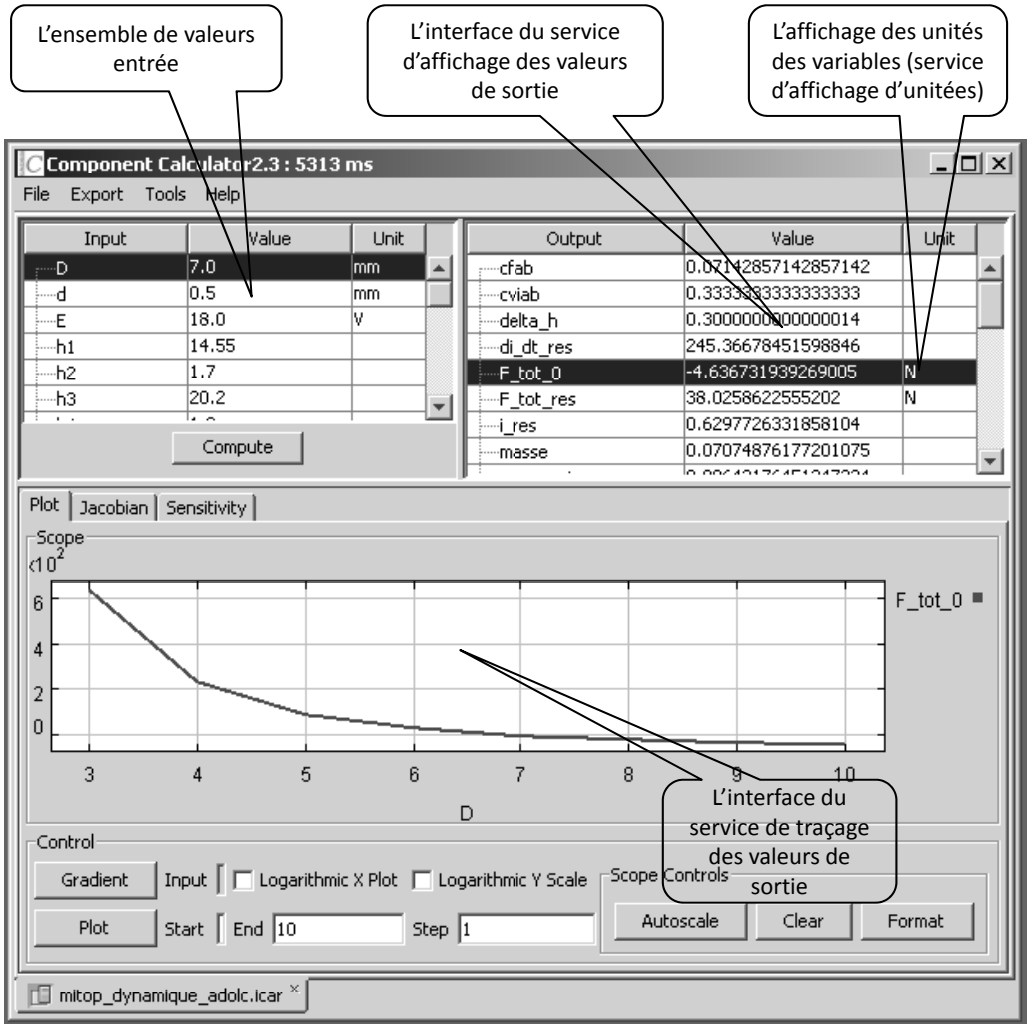


FIGURE B.6 – L'interface graphique du *Component Calculator* avec l'illustration des services d'affichage et traçage des variables de sorties

La figure B.6 illustre les interfaces graphiques des services d'affichage et de traçage des

valeurs de sortie implémentées dans le *Component Calculator*.

A la demande de l'utilisateur, le service d'affichage invoque la méthode `compute()` de la facette de calcul du composant *ICAr*, afin d'actualiser dans le tableau de droite les valeurs des variables de sortie. Le service de traçage autorise la variation d'une une seule entrée dans un intervalle spécifié par l'utilisateur et fixe les autres aux valeurs du tableau de gauche, afin d'afficher les courbe de variation correspondantes à plusieurs sorties (spécifiées aussi par l'utilisateur). Par exemple, sur la figure B.6, on trace la sortie `F_tot_0`, en faisant varier l'entrée `D` dans la plage de valeurs $[3, 10]$ avec un pas de 1. A noter que ce service utilise, lui aussi, la facette de calcul.

Le service d'affichage et celui de traçage des dérivées partielles fonctionnent sur le même principe. La figure B.7 illustre seulement l'interface graphique du service d'affichage des dérivées partielles.

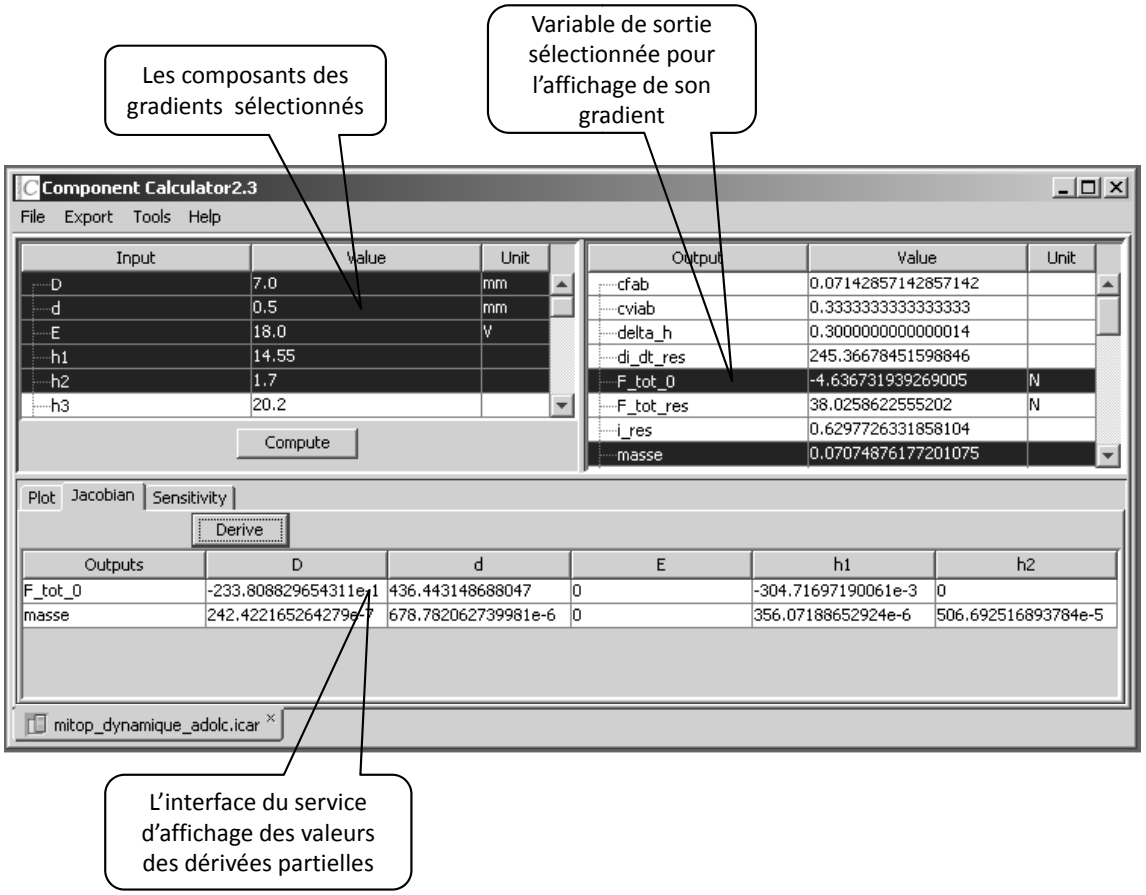


FIGURE B.7 – L'interface graphique du services d'affichage des valeurs des dérivées partielles

Ces services utilisent la facette de calcul des gradients ou des gradients sélectifs (voir paragraphe B.1.1) si cette dernière est disponible dans le composant *ICAr*.

B.3.2 Le service de calcul de sensibilités

L'interface graphique du service de calcul de sensibilités des variables de sortie d'un composant de calcul *ICAr* est illustrée sur la figure B.7. Ce service utilise en principe la facette de calcul des gradients et la facette de calcul.

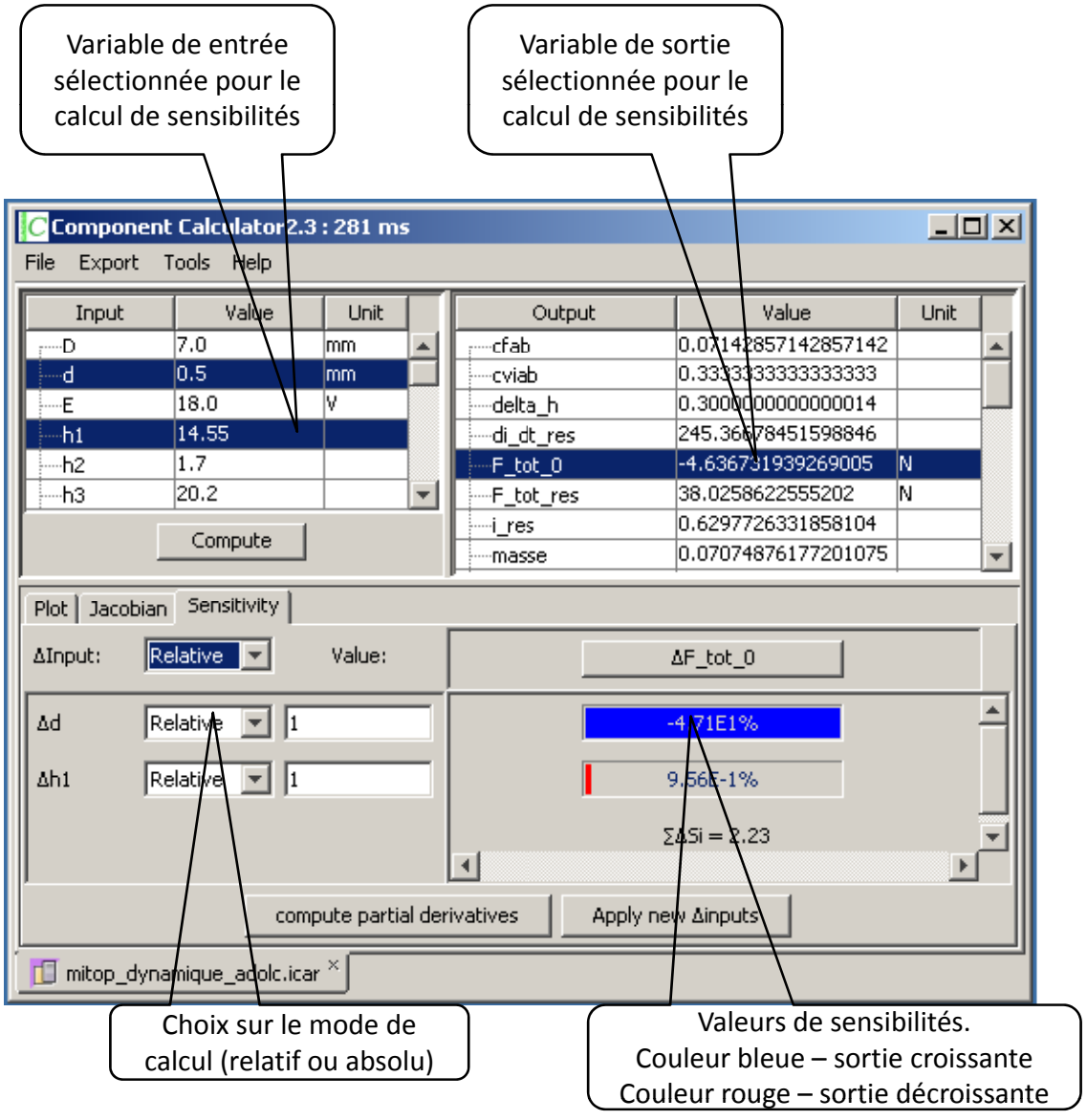


FIGURE B.8 – L'interface graphique du service de calcul des sensibilités

Ces sensibilités sont calculées, en linéarisant autour d'un point (voir figure B.9) la variation de la sortie concernée. Cette linéarisation est obtenue en approximant la courbe de variation de la sortie avec sa dérivée partielle de premier ordre.

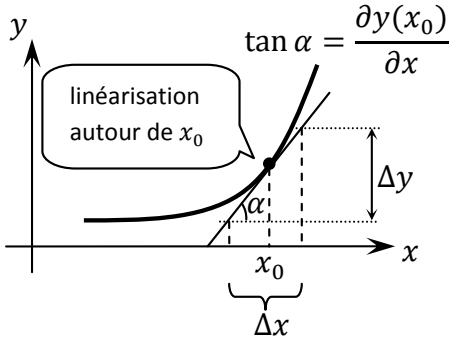


FIGURE B.9 – Linéarisation autour d'un point en vue de calcul des sensibilités

La formule de calcul de la variation de la de la sortie est :

$$\Delta y = \frac{\partial y(x_0)}{\partial x} \cdot \Delta x \tag{B.1}$$

où y est la variable de sortie, x_0 représente le point où on calcule la sensibilité et Δx représente sa variation. La sensibilité (relative) de la sortie y par rapport à la variable d'entrée x se calcule avec la formule :

$$\Delta s[\%] = \frac{\Delta y}{y_0} \times 100 \tag{B.2}$$

où y_0 représente la valeur initiale de la sortie, calculée en x_0.

Cette interface graphique permet à l'utilisateur d'effectuer facilement une analyse de sensibilités de toutes les sorties de son modèle de dimensionnement. Par exemple, il peut identifier facilement quelle est la variable d'entrée qui influence le plus une variable de sortie sélectionnée. Ceci est visible sur l'interface graphique, en identifiant la région la plus colorée aux endroits d'affichage des sensibilités correspondants à ces entrées. Pour notre exemple de la figure B.8, pour la sortie `F_tot_0`, la variable d'entrée `d` a une influence plus élevée que l'entrée `h_1`.

B.3.3 Le service de visualisation

Pour les composants de calcul pour lesquels on dispose d'un dessin du dispositif associé, on peut utiliser le service de visualisation disponible dans *Component Calculator* afin d'actualiser la vue du dispositif selon les valeurs courantes des paramètres géométriques.

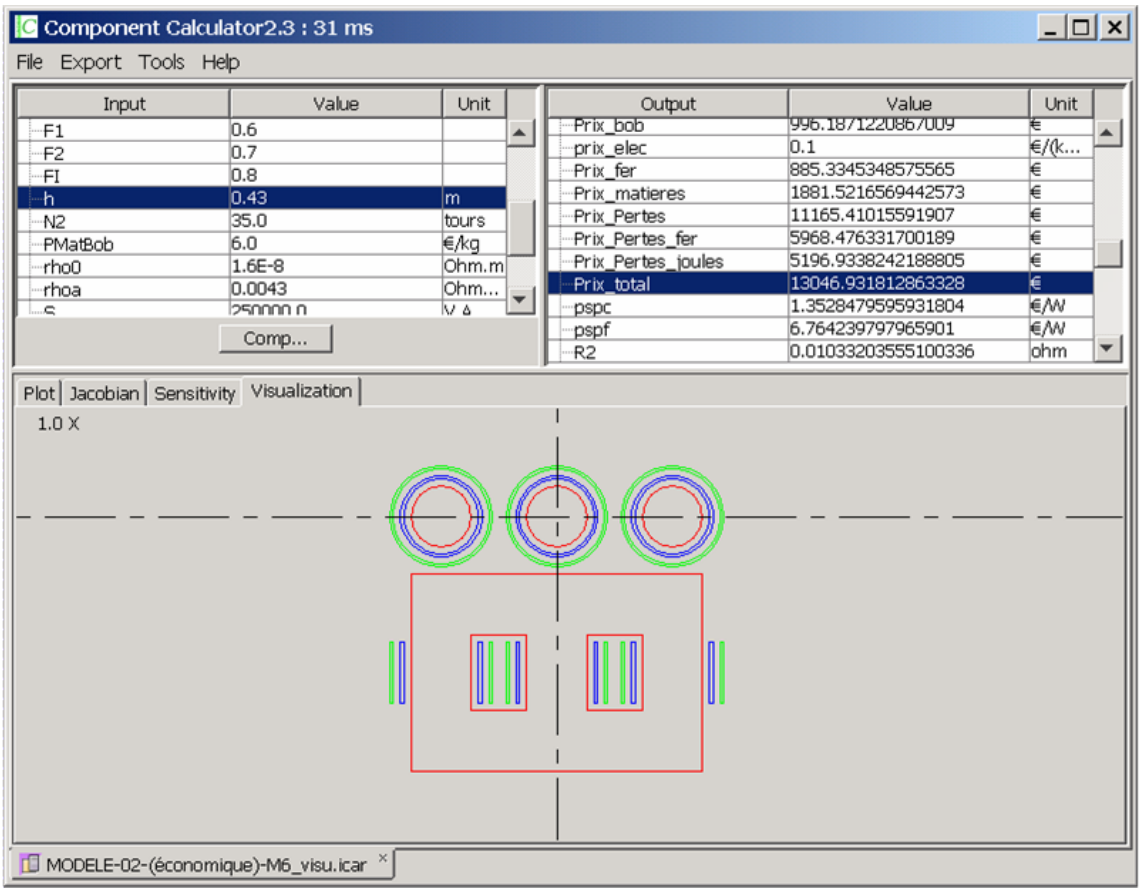


FIGURE B.10 – L'interface graphique associée au service de visualisation

Ce service utilise la facette de visualisation qui fait appel au dessin ou à l'image du dispositif.

Annexe C

La modélisation des dispositifs utilisés comme bancs de tests

C.1 Calcul de force pour le micro-actionneur intégré dans le miroir déformable

Dans ce paragraphe, nous proposons le calcul de la force crée par la bobine actionnant sur un aimant, utilisé pour la modélisation du micro-actionneur intégré dans le miroir déformable.

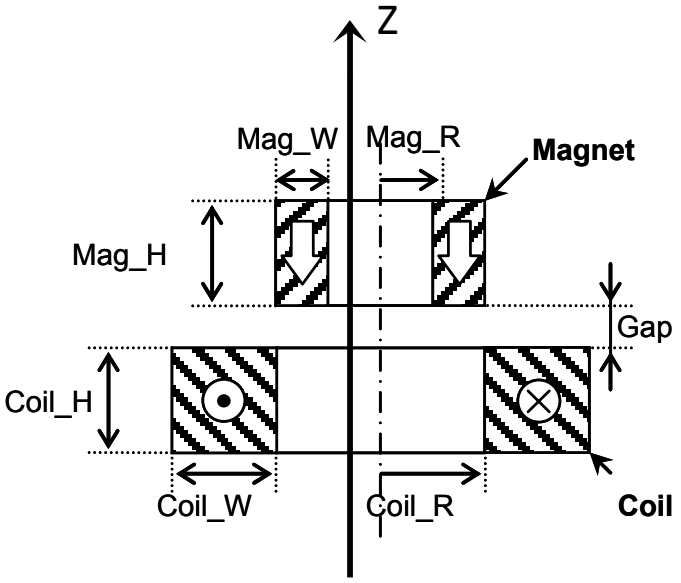


FIGURE C.1 – Le micro-actionneur électromagnétique utilisé dans la structure du miroir déformable

C.1.1 Calcul de force

Nous adoptons pour le calcul de la force appliquée à l'aimant, la méthode des charges magnétiques surfaciques proposée en [66] et [32]. Le principe de cette méthode est de modéliser l'aimantation M d'un aimant permanent par des charges magnétiques fictives surfaciques et/ou volumiques ; c'est la modélisation colombienne appliquée en magnétisme par analogie avec l'électrostatique :

$$\begin{aligned} \sigma_S &= \vec{M} \cdot \vec{n} \\ \sigma_V &= -\nabla \cdot \vec{M} \end{aligned} \tag{C.1}$$

La force applique à un aimant placé dans un champ H_{ext} se formule dans l'équation C.2 :

$$\vec{F} = \iint_S \sigma_S \cdot \vec{H}_{ext} dS + \iiint_V \sigma_V \cdot \vec{H}_{ext} dV \tag{C.2}$$

En adoptant l'hypothèse d'uniformité de l'aimantation de l'aimant ($\nabla \cdot \vec{M} = 0$), dans l'équation C.2, le calcul de force se réduit seulement à l'intégrale surfacique. Nous considérons seulement la composante H_z du champ magnétique externe ($H_x = 0$, $H_y = 0$). Pour la géométrie particulière de l'aimant, donnée sur la figure C.1 et en adoptant ces hypothèses, le calcul de la composante z de la force peut se réaliser avec l'intégrale double donnée dans l'équation C.3 :

$$F_z = \int_{Mag\_R}^{Mag\_R+Mag\_W} \int_0^{2\pi} \sigma_S \cdot H_{ext}^z d\rho d\theta \tag{C.3}$$

écrite en coordonnées cylindriques.

L'inconnue de cette équation est le champ magnétique externe. Pour le dispositif de la figure C.1, il s'agit du champ créé par la bobine. Dans le paragraphe suivant nous montrons l'évaluation de cette variable.

C.1.2 Calcul du champ

La loi de Biot-Savard permet de calculer le champ magnétique crée par un conducteur dans le vide en absence de matière aimantée en tout point de l'espace. Sa formule est donnée dans l'équation suivante :

$$\vec{H}(\vec{r}) = \frac{1}{4\pi} \left[\iiint_V \frac{\vec{j} \times (\vec{r} - \vec{r'})}{\left| \vec{r} - \vec{r'} \right|^3} dV \right] \tag{C.4}$$

où j représente la densité du courant traversant le conducteur, r représente la position du point d'observation et le vecteur r' définit la position de l'élément dV en coordonnées cartésiennes. En formulant l'intégrande de l'équation C.4 en coordonnées cylindriques ($\vec{r_c}$, $\vec{r_c'}$) et en considérant :

$$g_\theta = \int_{Mag\_R}^{Mag\_R+Mag\_W} \int_{-Mag\_H}^{Mag\_H} \frac{\vec{r_c} - \vec{r_c'}}{\left| \vec{r_c} - \vec{r_c'} \right|^3} d\rho dz \tag{C.5}$$

la composante H_z du champ émis par la bobine s'écrit (en appliquant la loi de Biot-Savard) :

$$H_{bobine}^z = \frac{j_\theta}{4\pi} \int_0^{2\pi} g_\theta d\theta \tag{C.6}$$

La formule de g_θ est analytique dépendant de la géométrie du dispositif. Sa formule de

calcul (générée par Maple 11 [61]) est la suivante :

$$\begin{aligned} g_\theta \;=\;& g\_0(Mag\_R\,cos(\theta), Mag\_R + Mag\_W, sin(\theta), \tfrac{Coil\_H+Mag\_H}{2} + Gap, \\ & Coil\_R + Coil\_W, \theta, \tfrac{Coil\_H}{2}) - \\ & - g\_0(Mag\_R\,cos(\theta), Mag\_R + Mag\_W, sin(\theta), \tfrac{Coil\_H+Mag\_H}{2} + Gap, \\ & Coil\_R + Coil\_W, \theta, -\tfrac{Coil\_H}{2}) - \\ & - g\_0(Mag\_R\,cos(\theta), Mag\_R + Mag\_W, sin(\theta), \tfrac{Coil\_H+Mag\_H}{2} + Gap, \\ & Coil\_R, \theta, \tfrac{Coil\_H}{2}) + \\ & + g\_0(Mag\_R\,cos(\theta), Mag\_R + Mag\_W, sin(\theta), \tfrac{Coil\_H+Mag\_H}{2} + Gap, \\ & Coil\_R, \theta, -\tfrac{Coil\_H}{2}) \end{aligned} \tag{C.7}$$

où

$$\begin{aligned} g\_0(x, y, z, \rho_s, \theta_s, z_s) \;=\;& (z_s - z) \ln \left(\frac{(\rho_\mathrm{s} - \mathrm{x}\cos(\theta_\mathrm{s}) - \mathrm{y}\sin(\theta_\mathrm{s}))}{\sqrt{(z_s - z)^2 + \mathrm{u}^2 + 0.1\,10^{-17}}} + \right. \\ & \left. + \sqrt{\frac{(\rho_\mathrm{s} - \mathrm{x}\cos(\theta_\mathrm{s}) - \mathrm{y}\sin(\theta_\mathrm{s}))^2}{(z_s - z)^2 + \mathrm{u}^2 + 0.1\,10^{-17}} + 1} \right) - \\ & - (x\cos(\theta_s) + y\sin(\theta_s)) \cdot \\ & \cdot \ln \left(\frac{z_s - z}{\sqrt{(\rho_s\cos(\theta_s) - x)^2 + (\rho_s\sin(\theta_s) - y)^2}} + \right. \\ & \left. + \sqrt{\frac{(z_s - z)^2}{v} + 1} \right) - \mathrm{u} \cdot \\ & \cdot \arctan \left(\frac{(z_s - z)\,(\rho_\mathrm{s} - \mathrm{x}\cos(\theta_\mathrm{s}) - \mathrm{y}\sin(\theta_\mathrm{s}))}{\mathrm{u}\sqrt{(z_s - z)^2 + v} + 0.1\,10^{-17}} \right) \\ u \;=\;& x\sin(\theta_s) - y\cos(\theta_s) \\ v \;=\;& (\rho_s\cos(\theta_s) - x)^2 + (\rho_s\sin(\theta_s) - y)^2 \end{aligned}$$

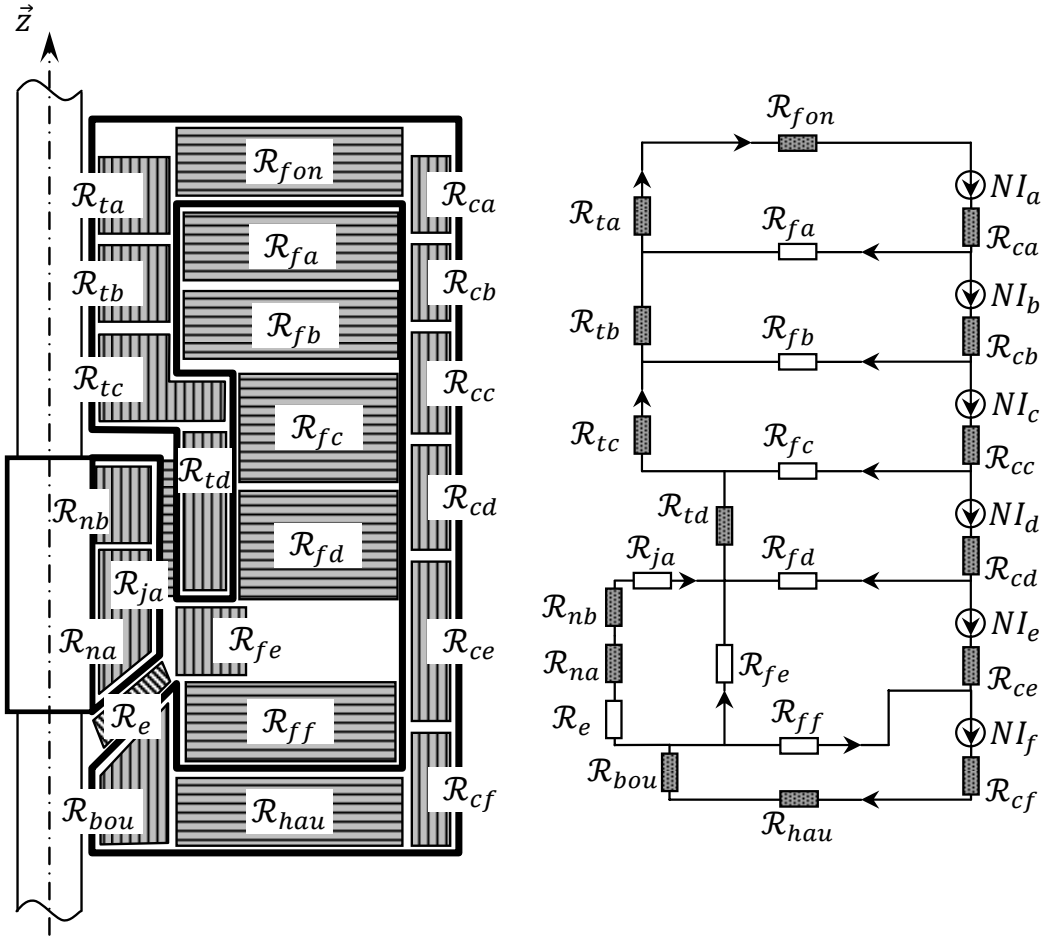
C.2 Modélisation de l'actionneur linéaire

Le modèle mathématique de l'actionneur linéaire présenté sur la figure C.2 est formulé en utilisant les réseaux de réluctances. Ce type de modélisation est préférable pour le dimensionnement des dispositifs électromagnétiques, car elle repose en grande partie sur des formules analytiques. Ce modèle simple donne de rapidité au processus d'optimisation faisant possible l'utilisation des algorithmes d'optimisation SQP, avec l'évaluation de gradients à moindre coût. Cependant, le principal inconvénient de cette modélisation est le manque de précision pour des modèles peu denses en réluctances. Elles sont bien adaptées dans une phase de pré-dimensionnement permettant de valider un cahier des charges ou de trouver un point de départ pour l'optimisation d'un modèle plus précis. Par exemple, souvent après avoir validé un cahier de charges avec SQP pour un modèle de di-

mensionnement à base de réseaux de réluctances, on utilise des algorithmes d'optimisation stochastiques appliqués au modèle initial affiné avec la méthode d'éléments finis, afin de trouver un optimum global précis.

C.2.1 La modélisation basée sur les réseaux de réluctances

Pour l'actionneur linéaire de la figure C.2, la solution pour la modélisation réluctante proposée en [17], est illustrée sur la figure C.2a. Ici, chaque réluctance correspond à une zone précise dans le dispositif. Il existe ainsi des réluctances dans l'entrefer, dans le fer et des réluctances de fuite. La bobine est divisée en six sources d'ampères-tours afin de prendre en compte les flux de fuite la traversant. Dans le dispositif considéré, ces flux ont une orientation radiale.



a. b.

FIGURE C.2 – Modélisation avec les réseaux de réluctances ; a. Situation réluctante finale avec les directions de circulation des flux ; b. Le circuit réluctant ; hachure pointillée - réluctance dans le fer ; hachure blanche - réluctance de fuite ou dans l'air

La schéma équivalent du circuit réluctant est construit en s'appuyant sur l'analogie électrique-magnétique. Ainsi, les courants sont équivalents aux flux magnétiques, les ré-

luctances aux résistances et les potentiels magnétiques sont équivalents à ceux électriques, etc.

C.2.2 La résolution des réseaux de réluctances avec l'approche implicite

Le circuit réluctant, illustrée sur la figure C.2b, permet d'établir le système d'équations liant les différents flux du circuit à ses sources d'ampère-tours. En appliquant le loi de Kirchhoff en tensions et grâce à l'analogie électrique-magnétique, on peut écrire dans une maille quelconque du circuit :

$$\sum \mathcal{R}(\phi) \cdot \phi - \sum N \cdot I = 0 \tag{C.8}$$

où $\mathcal{R}$ dénote la réluctance et $N \cdot I$ les sources d'ampères-tours. L'inconnue de cette équation implicite est le flux de maille, ϕ. Elle est donc résolue à courant constant. Les non-linéarités sont introduites dans l'équation C.8 par le produit algébrique $R(\phi) \cdot \phi$. Les valeurs des réluctances dépendent du flux magnétique la traversant ($\mathcal{R}(\phi)$) pour les matériaux ferromagnétiques non-linéaires :

$$\mathcal{R}(\phi) = \frac{1}{\phi} \cdot L \cdot H(B) \tag{C.9}$$

Nous utilisons l'alliage FeCo comme matériau magnétique pour les culasses du dispositif. La caractéristique $H(B)$ de cet alliage est celle de vacoflux50 [83]. Cette caractéristique est donnée sous la forme d'une courbe expérimentale sur la figure C.3 et sous la forme d'une expression analytique dans l'équation C.10.

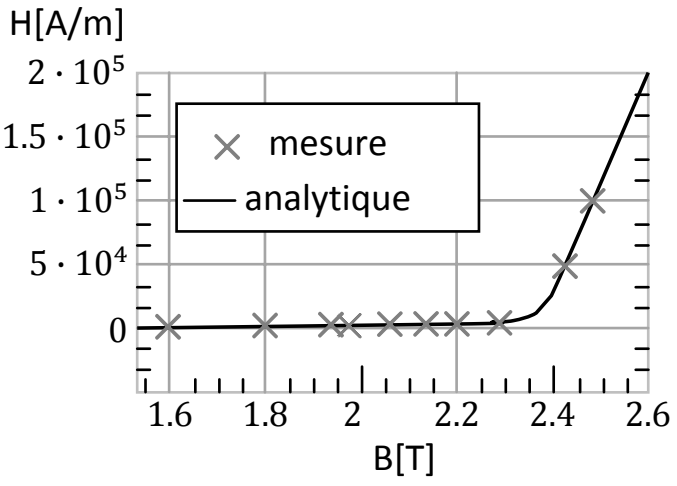


FIGURE C.3 – Caractéristique ferromagnétique de l'alliage FeCo (vacoflux50)

$$H_{FeCo}(B) = \begin{array}{l} 8380 \\ +(40 \cdot B + 0.15 \cdot B^{12}) \\ \cdot \left[\frac{1}{2} - \frac{\arctan(30 \cdot (B-2.37))}{\pi}\right] \\ +795\,775 \cdot (B - 2.35) \\ \cdot \left[\frac{1}{2} + \frac{\arctan(30 \cdot (B-2.37))}{\pi}\right] \end{array} \tag{C.10}$$

Afin d'éviter l'interpolation des données de mesure données par la courbe C.3, une expression analytique est préférable pour l'optimisation du dispositif, sachant que les gradients des formules analytiques peuvent être évalués facilement. Ainsi, nous proposons la formule analytique en C.10 qui reproduit avec une précision élevée la courbe du matériau. Ces formulations analytiques peuvent s'obtenir par optimisation SQP sous contraintes en minimisant une somme quadratique des erreurs entre la formule construite par la définition de certains coefficients et la mesure.

Les paramètres non implicites, ceux par rapport auxquels nous envisageons à obtenir des dérivées partielles, sont cachés dans la formulation implicite de l'équation C.9 dans les formules des réluctances ($\mathcal{R}$). Ces paramètres sont de nature géométrique (longueur, section du tube de flux correspondant) ou physique (perméabilité relative, polarisation magnétique à saturation, source d'ampère-tours, etc.). En adoptant la même notation, P, de ces paramètres, les inconnues du système d'équations implicites général du tableau V.6, appliqué aux circuits des réluctances, sont les flux des mailles du circuit :

$$\begin{cases} f_1(\phi_1, \cdots, \phi_n, P_1, \cdots, P_p) = 0 \\ \vdots \\ f_n(\phi_1, \cdots, \phi_n, P_1, \cdots, P_p) = 0 \end{cases} \tag{C.11}$$

Les flux de mailles sont utilisés pour le calcul des flux de chaque branche du circuit réluctant, plus précisément, les flux traversant chaque composant (réluctance) en appliquant la loi de Kirchhoff en courants. Ces flux sont ensuite utilisés pour le calcul de la force en considérant les sources d'ampères-tours constantes ($F(\phi(I = ct), X)$).

C.2.3 Calcul d'énergie

De manière générale, les densités d'énergie W et de coénergie W_{co} d'un système magnétique sont données par les relations intégrales C.12

$$\begin{aligned} W &= \int_0^B H dB \\ W_{co} &= \int_0^H B dH \end{aligned} \tag{C.12}$$

où B dénote l'induction du champ magnétique H.

Les densités d'énergie et de coénergie en C.12, s'expriment analytiquement en fonction des flux et des paramètres géométriques.

C.2.4 Calcul de force

Dans un réseau de réluctances, l'énergie et la coénergie de chaque élément est obtenue en multipliant ces densités par le volume associé à l'élément. En sommant ces grandeurs sur tous les éléments constitutifs du réseau, on obtient alors l'énergie et la coénergie globales du système.

La force magnétique s'obtient en dérivant la coénergie du système par rapport au déplacement en supposant les curants constants :

$$F = \left.\frac{\partial W_{co}}{\partial X}\right|_{I=ct} \tag{C.13}$$

où on adopte l'hypothèse que le système a un seul degré de liberté, en mouvement linéaire. Sachant que la coénergie est formulée analytiquement, la dérivée partielle en C.13 s'exprime aussi analytiquement. Des techniques de dérivation formelle peuvent s'appliquer pour ces dérivées sans des complications majeures.

Etant donné que le calcul de la coénergie s'effectue en fonction de flux, celui de la force dépend aussi de la résolution du système implicite C.11. Afin d'optimiser cette force pour le dispositif considéré avec des algorithmes basés sur le calcul des gradients, il est nécessaire d'évaluer les dérivées partielles de la solution du système implicite dans le sens illustré sur le tableau V.6.

C.3 Modélisation du déclencheur dynamique

Dans ce paragraphe, nous illustrons les différentes approches utilisées pour la modélisation du déclencheur dynamique. Nous réalisons l'évaluation des certaines grandeurs analytiques, ainsi que la modélisation à base des systèmes d'équations différentielles.

C.3.1 Modélisation à base des réseaux des réluctances du déclencheur dynamique

Pour le modèle réluctant du déclencheur dynamique illustré sur la figure C.4, on considère que les matériaux magnétiques utilisés sont linéaires et non saturables. On considère la perméabilité relative de l'aimant $\mu_{aim} = 1.1$ et son induction rémanente $B_r = 0.95\ T$.

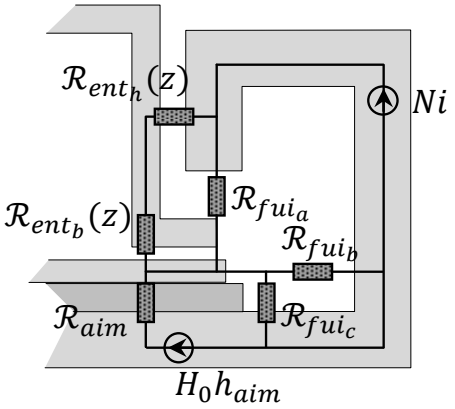


FIGURE C.4 – Schéma réluctant du déclencheur dynamique

En appliquant les lois de mailles dans ce circuit réluctant, on peut déduire le flux magnétique traversant le noyau mobile :

$$\phi_{noy} = \phi_{noy_{aim}}(z(t)) - \phi_{noy_{bob}}(z(t), i(t)) \tag{C.14}$$

où $\phi_{noy_{bob}}(z(t), i(t))$ dénote le flux créé par la bobine, traversant le noyau mobile et $\phi_{noy_{aim}}(z(t))$ dénote le flux créé par l'aimant :

$$\begin{aligned} \phi_{noy_{bob}}(z(t), i(t)) &= \frac{N \cdot i(t)}{\mathcal{R}_{aim} + \mathcal{R}_{ent}(z(t))} \\ \phi_{noy_{aim}}(z(t)) &= \frac{\mathcal{R}_{fui}}{\mathcal{R}_{fui} + \mathcal{R}_{ent}(z(t))} \cdot \frac{B_r \cdot h_{aim}}{\mu_0 \cdot \mu_{aim}} \cdot \frac{1}{\mathcal{R}_{aim} + \dfrac{\mathcal{R}_{ent}(z(t)) \cdot \mathcal{R}_{fui}}{\mathcal{R}_{ent}(z(t)) + \mathcal{R}_{fui}}} \end{aligned} \tag{C.15}$$

où h_{aim} est l'hauteur de l'aimant.

C.3.2 Le système d'état

Le modèle dynamique du dispositif représente un couplage entre le système d'état modélisant le circuit électrique d'alimentation de la bobine et les équations mécaniques de mouvement du noyau mobile. Ce système d'état fait apparaître deux équations différentielles électriques où l'état est représenté par les variables i et $\dot{i}$ et deux équations différentielles mécaniques où l'état est représenté par la position du noyau, z et sa vitesse, v.

$$\begin{cases} \dot{i} = \dfrac{di}{dt} \\ \ddot{i} = -\dfrac{R}{L} \cdot \dfrac{di}{dt} - \dfrac{i}{L \cdot C} \\ \dot{z} = v = \begin{cases} 0, \text{ si } F < 0 \\ v, \text{ si } F \geq 0 \end{cases} \\ \dot{v} = \dfrac{F}{m} = \begin{cases} 0, \text{ si } F < 0 \\ F, \text{ si } F \geq 0 \end{cases} \end{cases} \tag{C.16}$$

où F dénote la force résiduelle actionnant sur le noyau mobile, L est l'inductance totale de la bobine et m est la masse du noyau. Le circuit RLC série se modélise par une équation différentielle d'ordre 2, qui peut se réduire à deux équations différentielles ordinaires.

Afin d'éviter la simulation du déplacement du noyau mobile vers de bas, aspect qui n'est pas réalisable en réalité, nous formulons les conditions concernant le signe de la force résiduelle. Ainsi, pour une force négative aucun déplacement n'est pas signalé. Ceci introduit dans la résolution du système d'état une discontinuité, qui se positionne à l'instant du temps où le noyau mobile décolle. Normalement ce genre des discontinuités n'affectent pas la convergence des gradients comme l'on verra dans le paragraphe dédié aux résultats d'optimisation.

Les conditions initiales de ce système d'état sont formulées dans C.17 :

$$\begin{cases} i_0 = 0.0 \\ \dfrac{di}{dt} = \dfrac{E}{L} \\ z_0 = z_{min} \\ v_0 = 0.0 \end{cases} \tag{C.17}$$

Le critère d'arrêt est de nature *état final* et il est formulé par l'implicite C.18 :

$$z(t_r) = z_{max} \tag{C.18}$$

où z_{max} dépend des paramètres géométriques du dispositif et t_r dénote le temps de réponse.

C.3.3 Calcul de force

La force résiduelle actionnant sur l'aimant est le résultat de la composition de la force de l'aimant, de la force électromagnétique crée par la bobine, de la force du ressort et le poids du noyau :

$$F = F_{rs} - F_{aim} + F_{bob} - m \cdot g \tag{C.19}$$

La force magnétique totale, $F_{mag} = F_{bob} - F_{aim}$, est obtenue en modélisant le dispositif à base de réseaux de réluctances (voir C.3).

Le flux traversant le noyau dépend de sa position, $z(t)$, et du courant de la bobine, $i(t)$. Ainsi, la force magnétique totale, formulée dans l'équation C.20, est une grandeur

évaluée à chaque instant du temps d'intégration du système d'état. Ceci est en effet le cas de résolution d'un système d'équations différentielles algébriques (EDA), où on rajoute une équation analytique au système initial. La dérivation de cette équation analytique se déroule naturellement avec ADOL-C, n'impliquant pas des différences majeures.

$$F_{mag} = \frac{\phi_{noy}^2(z(t), i(t))}{\mu_0 \cdot S_{noy}} \tag{C.20}$$

La force du ressort est donnée par l'équation analytique C.21 et elle dépend aussi de la position du noyau.

$$F_{rs} = k \cdot (L_0 - z(t)) \tag{C.21}$$

où k est la constante du ressort formulé en C.22 :

$$k = \frac{G \cdot d^4}{8 \cdot n_s \cdot D^3} \tag{C.22}$$

où G dénote le module du Coulomb du matériau du ressort. Les autres paramètres du ressort sont illustrés sur la figure V.23c.

C.3.4 Calcul de l'énergie de percussion

L'énergie cinétique de percussion (en position haute) se calcule en appliquant la formule C.23 de la mécanique :

$$W\_per = \frac{1}{2} \cdot m \cdot v^2(t_r) \tag{C.23}$$

où $v(t_r)$ est la vitesse en position haute, calculée à l'instant t égal au temps de réponse.

C.3.5 Calcul de la tenue au choque en position basse

Nous exprimons la tenue au choque en position basse étant l'accélération qui pourrait produire le décollement du noyau, même si la bobine n'est pas alimentée. Cette accélération est obtenue par l'égalité entre la force de l'aimant et celle du ressort en ajoutant aussi le poids du noyau.

$$t\_choq = \frac{F_{aim} - F_{rs} + m \cdot g}{m} \tag{C.24}$$

C.3.6 Contraintes du ressort

La contrainte de viabilité du ressort se formule dans l'équation C.25 :

$$c\_viab = \frac{n_s}{L_0} \tag{C.25}$$

et celle de fabrication en C.26 :

$$c\_fab = \frac{d}{D} \tag{C.26}$$

Dérivation Automatique pour le calcul des sensibilités appliqué au dimensionnement en génie électrique

Résumé Le dimensionnement par optimisation est aujourd'hui d'un intérêt majeur, car il fournit un moyen fiable et rapide en vue de déterminer les performances souhaitées de dispositifs, tout en minimisant une fonction de coût. Nous sommes particulièrement intéressés par l'optimisation sous contraintes basée sur le calcul de gradients. Ces algorithmes nécessitent des valeurs précises des dérivées de la fonction objectif et des performances à contraindre. Evaluer ces dérivées exactes se relève comme une tâche complexe et très laborieuse, vu que les fonctions de performances et de coûts sont souvent évaluées à partir d'algorithmes numériques complexes. La Dérivation Automatique est une technique efficace pour calculer les dérivées des fonctions décrites au moyen de programmes informatiques dans des langages de programmation de haut niveau tel que Fortran, C ou C + +. Cette technique s'utilise parfaitement pour l'optimisation avec des algorithmes basés sur le calcul de gradients, étant donné que les dérivées sont évaluées sans aucune erreur de troncature ou d'annulation. Ce travail emploie la Dérivation Automatique pour calculer les gradients de programmes de calcul des modèles de dispositifs électromagnétiques.

Mots clés *Dérivation Automatique, Calcul de Gradients, Optimisation sous Contraintes, Conception, Dimensionnement des Dispositifs Electriques.*

Automatic Differentiation for computing sensitivities applied for sizing electromagnetic devices

Abstract Sizing by optimization is nowadays of major interest since it provides a fast and reliable way to achieve, with low manufacturing costs, desired performances for products lacking of optimality usually by means of minimizing a cost function. We are particularly interested by constrained gradient based optimization. Such algorithms require accurately valued derivatives of the objective function. This may be the origin of serious problems provided that often such functions may result from complex numerical algorithms. Automatic Differentiation is introduced as a powerful technique to compute derivatives of functions given in the form of computer programs in a high level programming language such as FORTRAN, C or C++. Such technique fits perfectly in combination with gradient based optimization algorithms, provided that the derivatives are valued with no truncation or cancellation error. This work employs Automatic Differentiation to compute gradients of programs computing the sizing models of electromagnetic devices.

Keywords *Automatic Differentiation, Gradient based Optimization, Design, Sizing of Electromagnetic Devices*