Application de la méthode des dérivées d'ordre élevé à l'optimisation de structures

Jean-Daniel Beley* — Claude Broudiscou — Philippe Guillaume** — Mohamed Masmoudi** — Frédéric Thevenon***

** CADOÉ SA - Novacité Alpha*
43 boulevard du 11 novembre, 69603 Villeurbanne cedex
*** CNRS - Université Paul Sabatier - INSA - UMR MIP 5640*
118, route de Narbonne, 31062 Toulouse cedex

RÉSUMÉ. *En optimisation de forme, l'utilisation de dérivées d'ordre élevé permet d'obtenir à partir d'une seule analyse par éléments finis une représentation explicite de la fonction coût par son polynôme de Taylor. La différentiation automatique fournit les dérivées exactes du problème discret, à un coût inférieur à celui d'une analyse par éléments finis. Deux exemples industriels en élasticité linéaire illustrent la puissance de cette nouvelle méthode.*

ABSTRACT. *In shape optimization problems the use of higher order derivatives leads to an explicit Taylor's expansion of the cost function, obtained from only one finite element analysis. The automatic differentiation provides exact derivatives of the discrete problem and its additional cost is low with respect to the cost of a finite element analysis. Two industrial examples in linear elasticity show the interest of this new method.*

MOTS-CLÉS : *optimisation de forme, différentiation automatique, optimisation de structures mécaniques, dérivées successives.*
KEYWORDS : *shape optimization, automatic differentiation, structural optimization, higher order derivatives.*

1. Introduction

La simulation numérique prend une place de plus en plus importante dans l'élaboration de produits industriels. Cette simulation a pour but de restituer le comportement physique de la structure étudiée (élasticité, aérodynamique, électromagnétisme, ...). Cette structure est définie par ses paramètres (géométrie, propriété de matériaux, ...). Curieusement, ce sont *ces paramètres*, dont une valeur initiale a été fournie à l'ordinateur, qui sont les *véritables inconnues* du concepteur. Il est donc essentiel de quantifier au mieux l'influence des paramètres de conception du produit. Le but de l'optimisation de forme est d'obtenir les meilleures performances du produit en choisissant une configura-

tion optimale de ces paramètres. Ce choix doit bien entendu être effectué dans des délais compatibles avec les contraintes industrielles.

Les performances de la structure étudiée sont mesurées par différents critères : poids, rigidité, maximum des contraintes (cas des structures), modes propres, traînée (cas d'une aile d'avion), diagramme de rayonnement d'une antenne, ... Si l'on note x le jeu de paramètres considérés, une approximation de l'état $U(x)$ est donnée par la résolution, très coûteuse en temps, d'une équation linéaire ou non-linéaire de la forme

$$F(x, U(x)) = 0. \tag{1}$$

L'état $U(x)$ représente par exemple la déformation élastique de la structure, le champ rayonné par une antenne, l'écoulement autour d'une aile d'avion. Chaque critère, ou fonction coût, est alors de la forme

$$j(x) = J(x, U(x)). \tag{2}$$

Les méthodes classiques d'optimisation de forme utilisent en général des méthodes de descente : à chaque itération, la valeur d'une fonction objectif est diminuée. Le choix de la fonction objectif en fonction des différent critères, le choix du paramétrage de la forme géométrique, ainsi que celui de la méthode d'optimisation, peuvent s'avérer assez délicats : certains critères sont difficiles à définir (comme l'esthétique), les solutions obtenues peuvent s'avérer en pratique irréalisables, des oscillations du bord du domaine à optimiser peuvent se produire. Ces méthodes ont en commun le fait qu'à chaque itération l'équation (1) doit être résolue à nouveau pour une autre valeur des paramètres x. Typiquement, cela signifie qu'à chaque itération on factorise une nouvelle matrice, opération qui représente l'essentiel du temps passé au cours de cette itération. Dans le cas linéaire, cette matrice est tout simplement celle du système (1) qui s'écrit alors $A(x)U(x) - B(x) = 0$. Dans le cas non-linéaire, cette matrice est celle obtenue par linéarisation de l'équation (1).

Or il s'avère qu'en général l'application $x \longrightarrow U(x)$ est analytique même lorsque $U(x)$ est une fonction irrégulière (cas d'une fissure chargée) : $U(x)$ peut donc être représenté par sa série de Taylor, ou en pratique approché par un polynôme de Taylor. Se pose alors le problème du calcul de la valeur en un point des dérivées partielles d'une fonction de plusieurs variables pour des ordres de dérivation qui peuvent être élevés. Ce calcul des dérivées peut être fait "à la main", mais cela peut s'avérer assez fastidieux. Une autre possibilité est d'utiliser la *différentiation automatique* (DA), qui, à partir d'un code évaluant une fonction $f(x)$, engendre un nouveau code donnant la valeur de $f(x)$ ainsi que de ses dérivées successives $f'(x)$, $f''(x), \cdots$, jusqu'à un ordre fixé par l'utilisateur.

La méthode que nous avons développée, Calcul Adaptatif et Dérivées d'Ordre Supérieur, évite de recommencer la résolution de (1) pour chaque nouveau jeu de paramètres. Son principe est très simple : l'état ayant été évalué pour un premier jeu de paramètres, la détermination explicite d'un polynôme de Taylor

à plusieurs variables permet ensuite d'approcher la valeur de l'état en un autre point. Les avantages de cette approche sont multiples :

- réduction importante des temps de calcul : en effet, le calcul des dérivées de l'état par rapport aux paramètres fait appel à la résolution d'un système dont la matrice a déjà été factorisée pour résoudre (1). Or le rapport *temps de factorisation / temps de résolution* est pour les grands systèmes de l'ordre de plusieurs centaines. Ainsi le calcul de quelques centaines de dérivées partielles de l'état par rapport aux paramètres ne prend pas plus de temps qu'une itération d'une méthode de descente classique,
- plus grande souplesse d'utilisation : lorsqu'il y a plusieurs critères à optimiser, il n'est pas toujours facile de déterminer le poids à donner à chacun de ces critères dans la fonction objectif qui pilote l'algorithme d'optimisation. Si l'on dispose d'une représentation polynomiale explicite de l'état, il n'est pas alors forcément nécessaire de définir une telle fonction objectif. L'évaluation des différents critères est quasiment immédiate, et le concepteur peut observer en temps réel la réponse de sa structure pour toutes les variations des paramètres à l'intérieur d'un domaine de validité. Il peut alors déterminer les paramètres optimaux par simple exploration de ce domaine,
- parallélisation naturelle de la méthode : à un ordre de dérivation fixé, les calculs des différentes dérivées partielles par rapport aux paramètres sont totalement indépendants, et peuvent être effectués en parallèle, le degré de parallélisation croissant rapidement avec le nombre de paramètres et l'ordre de dérivation,
- grande précision du calcul des dérivées : pour des problèmes linéaires, l'expérience numérique et la théorie [GuM 94] montrent que le calcul de l'état obtenu par approximation polynomiale pour un nouveau jeu de paramètres donne le même résultat que le calcul direct de l'état par résolution de (1) avec ces nouveaux paramètres,
- ouverture vers de nouvelles méthodes d'optimisation globale : des résultats très prometteurs ont été obtenus dans un problème de chimie quantique [MRP 95].

Plusieurs problèmes industriels ont ainsi été résolus :

- paramétrisation du calcul dans des guides d'ondes 2D (pour Alcatel Espace) [GuM 95],
- paramétrisation de guides d'ondes 3D en collaboration avec France Télécom,
- paramétrisation de champs électrostatiques et magnétostatiques en collaboration avec le Cedrat,

- paramétrisation du champ électromagnétique calculé par équation intégrale en collaboration avec Aérospatiale.

Les trois premières expériences ont donné lieu à trois logiciels industriels. Après une étude de faisabilité concluante, la dernière expérience est actuellement le support d'un projet européen pour l'industrialisation de la méthode (Projet Européen EMCP2).

Nous présentons dans cet article deux cas en élasticité linéaire 3D. Le cas non linéaire est plus délicat, mais des résultats très satisfaisants ont été obtenus par un des auteurs (F.Thévenon).

Nous décrivons d'abord dans la section 2 la méthode des dérivées d'ordre supérieur ainsi que sa mise en oeuvre. La section 3 est consacrée à la différentiation automatique, outil essentiel pour réaliser les calculs. Enfin nous présentons dans la section 4 deux exemples issus de l'industrie, sur lesquels a été appliquée la méthode des dérivées d'ordre supérieur.

2. Les dérivées d'ordre supérieur

Chaque évaluation d'un critère ou fonction coût $j(x) = J(x, U(x))$ conduit à une analyse par éléments finis coûteuse en temps. Les méthodes modernes d'optimisation de forme [Cea 86] ont permis de réduire le nombre de ces analyses en utilisant les dérivées premières ou secondes [Fuj 86] de la fonction coût par rapport à la géométrie. Mais ce n'est qu'en 1992 que deux des auteurs, mettant à profit la puissance des outils de la différentiation automatique, ont introduit les dérivées d'ordre supérieur, développant à la fois une étude théorique et des applications industrielles ([GuM 92], [GuM 93]).

Un certain nombre d'idées reçues avaient jusque là limité l'utilisation des dérivées aux ordres 1 ou 2 ; les dérivées d'ordre plus élevé apparaissaient comme coûteuses, difficiles à calculer et d'utilisation pratique incertaine.

Or d'une part les résultats de J.Morgenstern [Mor 85] et de V.Strassen [Str 90] montrent que le calcul de dérivées est peu coûteux et peut être réalisé à partir du code de calcul de la fonction en utilisant la différentiation automatique. D'autre part il a été démontré [GuM 94] [Gui 94] que dans le cas de problèmes linéaires, les dérivées de la fonction coût du problème discret :

- sont égales aux dérivées discrétisées de la fonction coût du problème continu (nous appelons problème continu le problème issu des équations de la physique, et problème discret celui issu de la discrétisation du problème continu par la méthode des éléments finis). Ceci justifie l'utilisation de la différentiation automatique qui, bien entendu, ne peut traiter que le problème discret,

- peuvent être calculées avec autant de précision que la fonction coût elle-même (i.e. avec le même ordre d'erreur),

- permettent, à partir d'une seule analyse par éléments finis, d'obtenir une très bonne approximation de la fonction coût du problème discret par son polynôme de Taylor (car celle-ci est, dans une large classe de problèmes, une fonction analytique par rapport aux données).

Il se pose cependant le problème du domaine de validité de la méthode, lié au rayon de convergence de la série de Taylor. Nous décrirons quelques techniques permettant d'agrandir ce domaine.

2.1. Description de la méthode

La méthode proposée est générale, mais on se contente de la présenter dans le cas linéaire. Dans le cas non linéaire, on applique les mêmes techniques aux équations linéarisées du problème.

Soit x un paramètre (géométrie, propriété de matériaux, ...) et $U(x)$ la solution de l'équation d'état

$$A(x)U(x) = B(x). \tag{3}$$

On suppose que pour résoudre cette équation, la matrice $A(x)$ a été factorisée. On peut alors tirer profit de cette factorisation de la manière suivante : dans le système

$$A(x+v)U(x+v) = B(x+v)$$

correspondant à un nouveau paramètre $x+v$, plutôt que de factoriser la matrice $A(x+v)$, on se propose d'approcher la solution $U(x+v)$ de ce système par son polynôme de Taylor d'ordre m :

$$U(x+v) \simeq U(x) + U'(x)v + \cdots + \frac{1}{m!}U^{(m)}(x)v^m.$$

En effectuant une seule analyse, on connaîtra la solution dans un voisinage du paramètre initial x. L'analycité du problème continu sous-jacent à l'équation (3) a été mis en évidence pour toute une famille de problèmes [Des 76], et celle du problème discret $x \longrightarrow U(x)$ est immédiate à partir du moment où les applications $x \to A(x)$ et $x \to B(x)$ sont analytiques, ce qui est le cas dans la plupart des méthodes de type éléments finis ou différences finies. Nous renvoyons à [GuM 93], [GuM 94] pour les aspects théoriques et les estimations d'erreur.

Le calcul des coefficients de Taylor est ensuite une simple application de la formule de Leibnitz : la première variation $U'(x)$ de $U(x)$ par rapport à x est donnée par la résolution du système

$$A(x)U'(x) = B'(x) - A'(x)U(x),$$

obtenu par dérivation de l'équation (3). De même, la dérivée d'ordre m de $U(x)$

par rapport à x, notée $U^{(m)}(x)$, est la solution du système

$$A(x)U^{(m)}(x) = B^{(m)}(x) - \sum_{i=1}^{m} C_m^i A^{(i)}(x)U^{(m-i)}(x). \qquad (4)$$

On remarque que tous ces systèmes sont formés à partir de la même matrice $A(x)$. Celle-ci ayant été factorisée une fois pour toutes, les résolutions successives de ces systèmes sont bien moins coûteuses en temps de calcul que la factorisation de la matrice elle-même. Seuls les seconds membres varient d'un système à l'autre. Cela reste vrai s'il y a plusieurs paramètres. Pour obtenir les seconds membres, il suffit de calculer les dérivées successives de $A(x)$ et de $B(x)$ par rapport à x, soit à la main, ce qui est le plus efficace mais peut être très fastidieux, soit en utilisant la DA. Pratiquement, celle-ci est mise en oeuvre sur les matrices élémentaires : la matrice $A(x)$ est de la forme

$$A(x) = \sum_{K \in \mathcal{K}} A_K(x)$$

où chaque matrice élémentaire $A_K(x)$, relative à l'élément K du maillage éléments finis $\mathcal{K}$, a au plus quelques dizaines d'éléments non nuls. La DA est alors appliquée sur la routine de calcul de $A_K(x)$, et fournit les dérivées successives $A_K^{(m)}(x)$. Il ne reste plus alors qu'à réassembler les matrices dérivées

$$A^{(m)}(x) = \sum_{K \in \mathcal{K}} A_K^{(m)}(x).$$

La même procédure est appliquée au calcul des dérivées de $B(x)$.

2.2. Quelques propriétés de la méthode

Pour des perturbations raisonnables du paramètre x, si nous comparons la solution obtenue par la formule de Taylor avec celle résultant d'un calcul direct, nous obtenons exactement le même résultat, c'est à dire la même représentation binaire [GuM 94]. Ceci s'explique par les faits suivants :

- la différentiation automatique calcule les dérivées exactes du problème discret,
- le problème discret est analytique.

Pour ces raisons, les propriétés précédentes sont vérifiées même dans le cas où l'on utilise des formules de quadrature pour calculer les coefficients des matrices élémentaires A_K et ceux du second membre B. On peut donc aussi utiliser cette méthode avec des éléments finis de surface.

Dans le cas de plusieurs paramètres, la méthode proposée est parallélisable. En effet, pour un ordre de dérivation fixé m, les calculs des dérivées croisées sont

en partie indépendants. Le degré de parallélisme augmente rapidement avec le nombre de paramètres et l'ordre de dérivation.

Cette méthode est particulièrement efficace pour les problèmes de grande taille, car le rapport *complexité de la factorisation / complexité de la résolution d'un système factorisé* augmente avec la dimension du système. Donnons un exemple : supposons que la matrice $A(x)$ soit une matrice symétrique $N^3 \times N^3$ provenant de la discrétisation par éléments finis d'un problème posé sur le maillage quadrangulaire d'un domaine cubique, comportant $N^3 = h^{-3}$ degrés de liberté. La largeur de bande est de l'ordre de N^2, et le coût d'une factorisation de Cholesky est de l'ordre de $N^7/2$ multiplications. Le coût de la résolution des deux systèmes triangulaires est de l'ordre de $2N^5$ multiplications. Le rapport *factorisation / résolution* est donc de $N^2/4$. Si on a par exemple $N = 20$ éléments par coté, ce qui représente une discrétisation modérément fine, ce rapport est de 100, ce qui donne un net avantage à l'approximation de la solution par développement de Taylor.

2.3. Perturbation de maillage

Pour définir une perturbation de maillage, on part d'une perturbation donnée du bord du maillage. Celle-ci provient d'une variation des paramètres qui définissent la géométrie (coordonnées des sommets s'il s'agit d'un polyèdre, coefficients de fonctions splines dans le cas général), variation demandée par l'utilisateur ou issue d'un calcul de gradient au cours d'une procédure d'optimisation. Il faut alors obtenir un prolongement de cette déformation à l'intérieur du maillage. Couramment, un tel prolongement est obtenu par résolution d'un problème d'élasticité avec déplacement imposé sur le bord. Ceci revient à imposer de la rigidité au maillage, et peut générer un maillage peu satisfaisant (ex. : triangles aplatis). Nous présentons dans ce paragraphe quelques critères permettant de construire une perturbation de maillage adaptée à la méthode des dérivées d'ordre élevé.

2.3.1. Domaine de convergence de la série de Taylor

Nous notons ici X la table des coordonnées des noeuds du maillage initial. Un maillage perturbé a pour table $X + V_X$ où V_X est une perturbation des coordonnées du maillage. Si l'on considère l'application $t \to A(X + tV_X)$, on s'aperçoit que cette application n'a de sens que sur un certain intervalle $[-t_1, t_2]$, $t_1,\ t_2 > 0$. En effet, pour de trop grandes valeurs de $|t|$, les mailles finissent en général par se croiser, l'aire (ou le volume) de certains éléments passe par zéro, et comme cette aire figure au dénominateur des matrices élémentaires, on a une division par zéro. La fonction $t \to A(X + tV_X)$ est ainsi en général une fonction méromorphe, et son développement en série de Taylor a un rayon de convergence fini $\rho(V_X)$.

Or il est clair que ce rayon de convergence $\rho(V_X)$ dépend de la perturbation V_X. On a donc intérêt à choisir les valeurs de V_X aux points intérieurs du maillage de telle sorte que $\rho(V_X)$ soit le plus grand possible. Ceci donne lieu à un problème d'optimisation. On peut encore améliorer le rayon de convergence de la série de Taylor par "changement de variable" : au lieu de $t \to A(X+tV_X)$, on considère une application de la forme $t \to A(X+\sum_{i=1}^{d} t^i V_i)$. Dans la pratique, on choisit d=2.

2.3.2. Erreur de discrétisation par éléments finis

Il a été établi dans [MuS 76] que la dérivée par rapport au domaine dans une direction V ne dépend que de la composante normale de V sur le bord du domaine.

Cependant, il a été prouvé dans [Mas 87] qu'après approximation par éléments finis, le calcul numérique de cette dérivée dépend fortement du comportement de la perturbation à l'intérieur du domaine. L'erreur entre la dérivée d'une fonction coût et la dérivée de son approximation fait directement intervenir la régularité de la perturbation du domaine. Pour être plus précis, introduisons les notations suivantes : soit Ω le domaine, V une perturbation du domaine, i.e. le domaine perturbé $\Omega + V$ est l'ensemble des points $x + V(x)$ pour $x \in \Omega$, et V_X l'ensemble des valeurs prises par V sur les noeuds. Soit $\Omega \to \tilde{j}(\Omega) = \tilde{J}(\Omega, u(\Omega))$ une fonction coût dérivable, l'état $u(\Omega)$ étant la solution du problème continu, $X \to j(X) = J(X, U(X))$ l'approximation de cette fonction coût, $U(X)$ étant la solution du problème continu discrétisé. Soit $d_\Omega \tilde{j}(\Omega; V)$ la dérivée dans la direction V de la fonction coût du problème continu, et $d_X j(X; V_X)$ celle du problème discret dans la direction V_X. On note h le pas de discrétisation du maillage, et $||\cdot||_{W^{m,\infty}(\Omega)}$ la norme de l'espace de Sobolev $W^{m,\infty}(\Omega)$, ensemble des transfomations du domaine Ω qui possèdent des dérivées Lipschitziennes jusqu'à l'ordre $m-1$. Nous avons obtenu le résultat suivant [Mas 87]:

$$|d_\Omega \tilde{j}(\Omega; V) - d_X j(X; V_X)| \leq c h^{k+k'} ||V||_{W^{k'+1,\infty}(\Omega)}. \qquad (5)$$

Ce résultat a lieu si l'on utilise des éléments finis de type Lagrange de degré k et si $V \in W^{k'+1,\infty}(\Omega)$ pour un certain k', $1 \leq k' \leq k$.

Cela veut dire que la régularité de la perturbation V intervient à deux niveaux dans l'estimation de l'erreur sur le calcul de la dérivée :

- par la norme de V : $||V||_{W^{k'+1,\infty}(\Omega)}$, est une constante multiplicative de l'erreur ; plus V est irrégulière, plus cette norme est grande,
- dans l'ordre de l'erreur (via k') : l'ordre de l'erreur augmente avec le degré de régularité de V.

Ces résultats théoriques ont été étendus au calcul de dérivées d'ordre supérieur et validés par des expériences numériques [GuM 94].

Pour ces raisons, l'algorithme qui génère les maillages perturbés doit construire des déformations V le plus régulières possible.

2.3.3. Étude de stabilité

Le calcul des dérivées d'ordre élevé entraîne la résolution successives de plusieurs systèmes linéaires (4). A chaque résolution, des erreurs s'accumulent dans les seconds membres. A priori, le résultat final dépendra donc fortement du conditionnement de la matrice A. Or les résultats numériques obtenus dans [GuM 94] ont montré une stabilité remarquable: la série de Taylor calculée converge tellement bien vers le résultat d'un calcul direct qu'on obtient la même représentation binaire quand on dérive jusqu'à l'ordre 50. Nous donnons ici une explication de ce comportement de la série.

Rappelons d'abord le résultat classique de stabilité pour la résolution d'un système linéaire $AX = B$. On suppose la matrice A symétrique définie positive, et on note $(\lambda_i, v_i)_{i=1}^n$ ses éléments propres. Les valeurs propres sont rangées par ordre croissant. On a alors

$$\frac{|\delta X|}{|X|} \leq \frac{\lambda_n}{\lambda_1} \frac{|\delta B|}{|B|}.$$

où δX est l'erreur provoquée par une perturbation δB de B:

$$A(X + \delta X) = B + \delta B$$

et $|\cdot|$ désigne la norme usuelle de $\mathbb{R}^n$. Si l'on examine la situation de plus près, on se rend compte que l'erreur de résolution de $AX = B$ dépend en fait fortement de B. En effet, si le second membre s'écrit $B = \sum_{i=1}^d b_i v_i$ avec $d < n$, alors nous avons une estimation plus fine de l'erreur:

$$\frac{|\delta X|}{|X|} \leq \frac{\lambda_d}{\lambda_1} \frac{|\delta B|}{|B|},$$

(à partir de $A\,\delta X = \delta B$ on obtient $|\delta X| \leq \frac{1}{\lambda_1}|\delta B|$, puis de $X = \sum_{i=1}^d \frac{b_i}{\lambda_i} v_i$, on déduit $|X|^2 = \sum_{i=1}^d \frac{|b_i|^2}{\lambda_i^2} \geq \frac{1}{\lambda_d^2}|B|^2$).

On obtient un résultat similaire lorsque les composantes de B selon $(v_i)_{i=d+1}^n$ ne sont pas nulles mais restent petites. Si on décompose $B = \sum_{i=1}^d b_i v_i + \sum_{i=d+1}^n b_i v_i = B_0 + B_1$ et si $|B_1| \leq \alpha |B|$ avec $\alpha < 1$, la majoration précédente devient

$$\frac{|\delta X|}{|X|} \leq \frac{\lambda_d}{\lambda_1 \sqrt{1 - \alpha^2}} \frac{|\delta B|}{|B|}.$$

Ceci illustre le fait que la stabilité de la résolution d'un système linéaire dépend de la "régularité" du second membre. Or les résolutions successives (4) font intervenir la perturbation V dans les seconds membres. On peut montrer que la décomposition de ces seconds membres dans la base des vecteurs propres

de A donne des composantes selon $(v_i)_{i=d+1}^{n}$ d'autant plus petites que la perturbation V est régulière. La régularité de V intervenait dans l'estimation d'erreur (5), elle intervient donc aussi au niveau de la stabilité des calculs.

2.4. Gestion de la différentiation automatique

Étant donné une fonction codée dans un langage donné, un logiciel de différentiation automatique génère un exécutable de calcul de ses dérivées.

Naturellement, le programme généré dépend des données fournies à l'exécution. En général, la fonction f est analytique par morceaux. Le programme dérivé ne tiendra compte que de la branche analytique $\tilde{f}$ où la fonction a été calculée (cf. Figure 1).

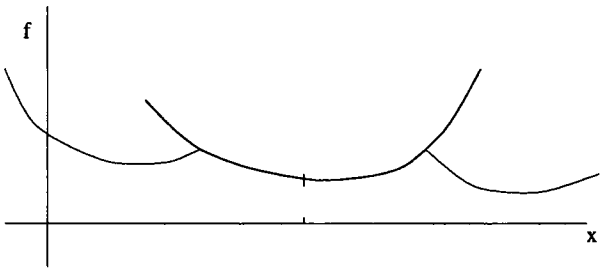

FIG. 1 – *Le prolongement analytique*

Considérons une fonction de classe C^d

$$f : \mathbb{R}^n \longrightarrow \mathbb{R}^m.$$

Étant donné $x_0, x_1, \ldots, x_d \in \mathbb{R}^n$, il est possible d'obtenir le développement

$$f(\sum_{i=0}^{d} t^i x_i) = \sum_{i=0}^{d} t^i y_i.$$

Le programme F engendré par différentiation automatique est destiné à effectuer ce développement. Il est appelé avec les arguments

$$F(x, y)$$

où x est l'entrée $x = (x_0, x_1, \ldots, x_d)$, et y la sortie $y = (y_0, y_1, \ldots, y_d)$.

Si $x_1 = x_2 = \ldots = x_d = 0$ alors le programme F donne l'évaluation de f au point x_0.

Si $x_2 = x_3 = \ldots = x_d = 0$, alors le programme donne le développement de Taylor de $f(x_0 + tx_1) = \sum_{i=0}^{d} t^i y_i$.

Dans notre cas, ceci s'applique au niveau des déformations de maillages à la fonction $t \to A(X + \sum_{i=1}^{d} t^i V_i)$ définie au paragraphe 2.3.1.

La différentiation automatique permet ainsi de mettre en oeuvre facilement les critères de qualité de maillages mentionnés au paragraphe précédent pour le calcul des dérivées des matrices élémentaires.

2.5. Gestion des paramètres géométriques

Si les aspects fondamentaux relatifs à la gestion de la géométrie sont aujourd'hui bien maîtrisés, l'implantation de ces outils nécessite beaucoup de savoir faire et de nombreuses années de travail. Ceci est d'autant plus vrai lorsque l'on s'intéresse à la paramétrisation de la géométrie : une géométrie peut être définie par un certain nombre de paramètres qui peuvent être liés par des relations mathématiques (des contraintes d'égalité). En général, on dispose d'au moins un paramètre libre (indépendant) permettant de faire varier les autres.

Le logiciel *I – DEAS Master Series* propose un environnement permettant de mettre à jour automatiquement la géométrie à partir de la donnée par l'utilisateur des paramètres libres.

L'utilisateur choisit ses paramètres de conception et leur domaine de variation. Ces variations se traduisent par des modifications de la surface de la structure. Le logiciel *Adomesh* accède à la base de données via l'environnement *Open Architecture* pour obtenir :

- les paramètres et leur domaine de variation,
- pour chaque valeur des paramètres, la géométrie correspondante,
- le maillage initial.

A partir de ces informations, le logiciel *Adomesh* calcule les perturbations optimales V_i du maillage pour chaque nouvelle géométrie.

3. La différentiation automatique (DA)

Nous avons tous eu des difficultés pour calculer la primitive d'une fonction, alors que la dérivation se fait d'une manière relativement aisée en appliquant les règles élémentaires de dérivation.

Paradoxalement, la situation s'inverse lorsqu'il s'agit de calcul numérique. En effet, l'intégration numérique est une opération stable alors que la dérivation numérique ne l'est pas. De plus l'algorithme de dérivation dépend (très fortement) de paramètres difficiles à estimer.

Or le calcul du gradient d'une fonction est d'une importance capitale pour les applications. Ces différents constats sont à l'origine de la différentiation automatique. Un programme de calcul n'utilise que des opérations élémentaires (+, -, ×, /, cos, sin, ...). En un point donné, pour obtenir la valeur de la différentielle de la fonction définie par ce programme (en FORTRAN, C, ADA, ...), il suffit d'évaluer la dérivée exacte de chacune de ces opérations au point courant.

Contrairement aux méthodes de différences finies, la valeur de la dérivée ainsi calculée par DA est exacte aux erreurs d'arrondis près. Par ailleurs, la DA est fondamentalement différente de la différentiation symbolique telle qu'on la trouve dans les codes de calcul formel: cette dernière, à partir de la connaissance d'une représentation par une formule mathématique de la fonction à dériver, donne une autre expression mathématique pour représenter la dérivée. La DA opère directement sur un programme complet, qu'elle transforme pour donner un nouveau programme qui calcule la fonction ainsi que ses dérivées. Le processus de différentiation automatique peut être représenté par le schéma:

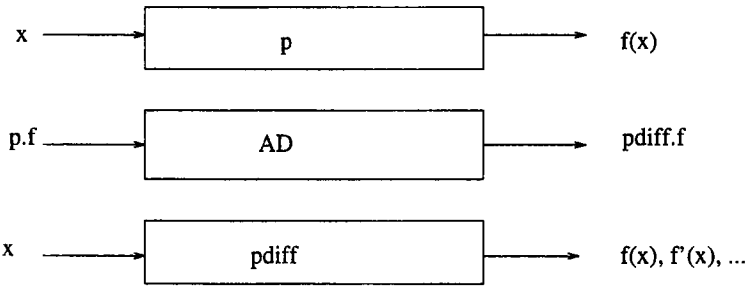

FIG. 2 – *La différentiation automatique*

La différentiation automatique peut être utilisée dans de nombreux domaines du calcul numérique:

- en optimisation pour le calcul du gradient et éventuellement du Hessien,
- pour le calcul de l'application linéaire tangente afin de résoudre des problèmes non linéaires,
- pour l'étude de la stabilité d'un calcul numérique.

L'étude de stabilité mesure, sans coût supplémentaire, la sensibilité du résultat par rapport à toutes les variables intervenant dans son calcul (ce qui peut permettre de repérer les opérations détériorant le plus la précision des calculs) ;

elle se fait en utilisant le mode inverse, dont l'autre intérêt principal est de calculer un gradient avec un rapport *temps de calcul du gradient/temps de calcul de la fonction* indépendant du nombre de variables.

Nous développons brièvement ces différents points dans cette section en nous inspirant essentiellement du travail de J.C.Gilbert [GVM 91], auquel nous renvoyons le lecteur pour plus de détails.

3.1. Fonctions définies par leur programme

Soit la fonction

$$\begin{array}{rccc} f: & \mathbb{R}^n & \longrightarrow & \mathbb{R}^m \\ & x & \longrightarrow & f(x) \end{array}$$

avec $x = (x_1, \ldots, x_n)$; les x_i $(i = 1, ..., n)$ sont appelées variables *indépendantes*, et un certain nombre de variables auxiliaires $x_{n+1}, \ldots, x_N$ appelées variables *intermédiaires* sont introduites.

3.1.1. Le modèle simple

Prenons par exemple le calcul de

$$f(x) = \frac{\cos(x_1 + x_2) + x_3}{1 + x_1 x_3 \exp(x_2)}$$

Il peut se décomposer en une succession d'opérations élémentaires :

$$\begin{array}{ll} x_4 := x_1 + x_2 & x_8 := \exp(x_2) \\ x_5 := \cos(x_4) & x_9 := x_7 \times x_8 \\ x_6 := x_5 + x_3 & x_{10} := x_9 + 1 \\ x_7 := x_1 \times x_3 & f = x_{11} := x_6 / x_{10} \end{array}$$

Dans ce modèle dit simple une variable intermédiaire est introduite à chaque instruction, ce qui fait utiliser $N = n + K$ variables si le nombre d'instructions est K. Nous pouvons écrire ce calcul sous la forme :

$$\begin{cases} for\ i := n+1\ to\ N\ do\ x_i := \varphi_i(x_1, \ldots, x_{i-1}) \\ f := x_N \end{cases}$$

Les φ_i sont appelées fonctions *intermédiaires*. En général les fonctions φ_i ne dépendent que d'une partie des autres variables x_j, $j < i$. Si on note $S_i \subset \{1, \ldots, i-1\}$ l'ensemble des indices correspondant à ces variables effectivement utilisées et x_{S_i} l'ensemble des x_j pour $j \in S_i$, l'algorithme s'écrit alors :

$$\text{(M1)} \qquad \begin{cases} for\ i := n+1\ to\ N\ do\ x_i := \varphi_i(x_{S_i}) \\ f := x_N \end{cases}$$

3.1.2. Le modèle étendu

Le modèle précédent est trop simpliste et ne prend pas en compte, par exemple, le cas de variables intermédiaires redéfinies dans le calcul d'un produit scalaire :

$$\begin{cases} p := 0 \\ for\ i := 1\ to\ n\ do\ p := p + x_i * a_i \\ f := p \end{cases}$$

La redéfinition des variables (indépendantes ou non) conduit au modèle étendu :

$$\text{(M2)} \qquad \begin{cases} for\ k := 1\ to\ K\ do\ x_{\mu_k} := \varphi_k(x_{S_k}) \\ f := x_N \end{cases}$$

où μ_k est l'indice de la variable affectée par la k-ième instruction. On doit faire naturellement l'hypothèse de consistance suivante : une variable doit avoir été définie avant d'être utilisée comme paramètre d'une autre variable.

3.1.3. Les fonctions intermédiaires

Les fonctions φ_i peuvent être, comme nous l'avons vu dans l'exemple, des fonctions de calcul de base, mais aussi une ligne de programme, une procédure : cette grande liberté est une des forces de la DA. Il leur est cependant demandé d'être régulières dans un voisinage du point où elles sont évaluées. Dans la plupart des codes de DA actuels une fonction intermédiaire identifiée par le code ne dépend que d'une ou deux variables, la réduction étant faite automatiquement par le différentiateur. Ceci peut amener à introduire un grand nombre de variables intermédiaires.

De même qu'en optimisation de forme, il existe ensuite deux façons de procéder en DA : le mode *direct* et le mode *inverse* (ou mode *adjoint*). De nombreux auteurs ont étudié la complexité et l'encombrement mémoire de chacun de ces modes [BaS 83], [Mor 85]. Nous présentons maintenant ces deux modes à partir d'un exemple simple, en les comparant à la méthode directe et à celle du Lagrangien utilisées en optimisation de forme.

3.2. Les méthodes directes

Dans tout ce qui suit nous utilisons les notations classiques : d_x dérivée par rapport à $x = (x_1, \ldots, x_n)$, ∂_u dérivée partielle par rapport à u, avec en particulier $d_x x_i = e_i$ pour $1 \leq i \leq n$, ${}^t e_i = (0, \ldots, 0, 1, 0, \ldots 0)$.

Nous commençons par rappeler la méthode directe en optimisation de forme.

3.2.1. La méthode directe en optimisation de forme

Soit

$$A(x)U(x) = B(x) \tag{6}$$

le système linéaire obtenu après discrétisation par éléments finis d'un problème linéaire en structure statique : $A(x)$ est la matrice de rigidité, $B(x)$ un vecteur représentant la charge, $U(x)$ le déplacement correspondant et x un vecteur de paramètres représentant la forme, les propriétés de matériaux, ... Nous voulons minimiser une fonction coût

$$J(x, U(x))$$

et pour cela calculer, si elle existe, la différentielle de $j(x) = J(x, U(x))$:

$$d_x j(x) = \partial_x J(x, U(x)) + \partial_U J(x, U(x)) d_x U(x).$$

Pour cela on calcule $d_x U(x)$ en résolvant les systèmes linéaires

$$A(x)\partial_{x_i} U(x) = \partial_{x_i} B(x) - \partial_{x_i} A(x) U(x) \quad 1 \leq i \leq n,$$

obtenus en dérivant le système (6). On remarque que le calcul de $d_x j(x)$ par cette approche nécessite la résolution de n systèmes supplémentaires.

3.2.2. Le mode direct en DA

Le mode direct en DA est tout à fait semblable à la méthode directe en optimisation de forme : il suffit de calculer le gradient de chacune des fonctions intermédiaires en utilisant la règle de dérivation des fonctions composées ; le calcul d'une fonction et de son gradient se font en parallèle en rajoutant dans le programme initial les instructions relatives au calcul de ce dernier. Par exemple à partir de l'algorithme (M2) nous écrivons :

$$\begin{cases} \textit{for } i := 1 \textit{ to } n \textit{ do } d_x x_i = e_i \\ \textit{for } k := 1 \textit{ to } K \textit{ do } \{ \\ \quad \partial_{x_i} x_{\mu_k} := \sum_{j \in S_k} \partial_{x_j} \varphi_k(x_{S_k})\, \partial_{x_i} x_j \quad 1 \leq i \leq n \\ \quad x_{\mu_k} := \varphi_k(x_{S_k})\} \\ \partial_{x_i} f := \partial_{x_i} x_N \quad 1 \leq i \leq n \\ f := x_N \end{cases}$$

Si nous voulons limiter l'encombrement mémoire, nous pouvons calculer les dérivées partielles les unes après les autres, mais cela peut aller jusqu'à doubler le temps de calcul nécessaire pour calculer f et f' en un point :

$$
\left\{
\begin{array}{l}
for\ i := 1\ to\ n\ do\ \{ \\
\quad for\ l := 1\ to\ n\ do\ u_l := 0 \\
\quad u_i := 1 \\
\quad for\ k := 1\ to\ K\ do\ \{ \\
\qquad u_{\mu_k} := \sum_{j \in S_k} \partial_{x_j} \varphi_k(x_{S_k})\, u_j \\
\qquad x_{\mu_k} := \varphi_k(x_{S_k})\} \\
\quad \partial_{x_i} f := u_N \\
\quad f := x_N\}
\end{array}
\right.
$$

où $u_k = \partial_{x_i} x_k$. Ce dernier algorithme est similaire à celui de dérivation numérique (différences finies), mais il a l'avantage de ne pas comporter d'erreur d'approximation.

Malheureusement, pour ces deux algorithmes, le temps de calcul est proportionnel au nombre de paramètres n : c'est une des raisons de l'introduction du mode inverse.

3.3. Les méthodes inverses

3.3.1. La méthode du Lagrangien en optimisation de forme

Le coût élevé du calcul de la différentielle dans le problème modèle du paragraphe 3.2.1 a conduit à définir le Lagrangien [Céa 86] :

$$L(x, U, P) = J(x, U) + (A(x)U - B(x), P)$$

où $P \in \mathbb{R}^N$ est un multiplicateur de Lagrange de même dimension que U, et $(\cdot, \cdot)$ désigne le produit scalaire usuel de $\mathbb{R}^N$.

Si $U = U(x)$ est la solution du système linéaire (6), alors

$$J(x, U(x)) = L(x, U(x), P) \quad \forall P \in \mathbb{R}^N$$

et, compte tenu de $d_x P = 0$, la différentielle de J est donnée par la relation

$$d_x J(x, U(x)) = \partial_x L(x, U(x), P) + \partial_U L(x, U(x), P)\, d_x U(x).$$

Dans le but d'éliminer $d_x U(x)$, choisissons $P = P(x)$ de façon à ce que $\partial_U L(x, U(x), P(x)) = 0$, ce qui fournit le système *adjoint* :

$$^t A(x) P(x) = -\partial_U J(x, U(x)), \tag{7}$$

dont la solution est l'état adjoint. La dérivée de j est alors

$$d_x j(x) = \partial_x J(x, U(x)) + (\partial_x A(x) - \partial_x B(x), P(x)).$$

On remarque que le calcul de $d_x j(x)$ ne nécessite plus que la résolution d'un seul système supplémentaire, ce qui rend cette méthode très avantageuse par rapport à la méthode directe du paragraphe 3.2.1.

3.3.2. Le mode inverse en DA

Le mode inverse en DA (ou "backward mode") a sur le mode direct deux avantages fondamentaux: un temps de calcul indépendant du nombre n de variables indépendantes, et la possibilité d'avoir des informations sur la sensibilité de la fonction calculée par rapport aux variables intermédiaires. Par souci de simplicité nous ne l'exposerons que dans le cas du modèle simple (M1) du paragraphe 3.1.1 . Son principe est semblable à celui de l'état adjoint que nous venons de rappeler : à chaque instruction de (M1), considérée comme une contrainte, nous associons un scalaire p_k ce qui introduit le multiplicateur de Lagrange $P = (p_{n+1}, \ldots, p_N)$. L'opérateur de Lagrange associé s'écrit alors

$$L(x, U, P) = f + \sum_{k=n+1}^{N} (x_k - \varphi_k(x_{S_k}))p_k,$$

où $U = (x_{n+1}, \ldots, x_N)$ est le vecteur des variables intermédiaires. Rappellons que $x = (x_1, \ldots, x_n)$ est le vecteur des variables indépendantes, et que $f = x_N$. De la même façon que dans le paragraphe précédent, nous obtenons le problème adjoint :

$$\partial_U L(x, U(x), P) = 0 \Longleftrightarrow \partial_{x_i} L(x, U(x), P) = 0, \quad n+1 \leq i \leq N.$$

En tenant compte de $\partial_{x_i}\varphi_k = 0$ si $i \geq k$, ce système s'écrit explicitement :

$$\sum_{k=i}^{N} (\delta_{ki} - \partial_{x_i}\varphi_k(x_{S_k}))p_k = -\delta_{iN}, \quad n+1 \leq i \leq N,$$

où δ est le symbole de Kronecker.

L'état adjoint $P(x)$, solution du système linéaire triangulaire supérieur régulier précédent (diagonale formée de 1), se calcule aisément par l'algorithme:

$$\begin{cases} p_N = -1 \\ \textit{for } i := N-1 \textit{ down to } n+1 \textit{ do} \\ \quad p_i := \sum_{k=i+1}^{N} \partial_{x_i}\varphi_k(x_{S_k})\, p_k \end{cases}$$

d'où le nom de mode inverse. Nous en déduisons comme précédemment la différentielle de f

$$\partial_{x_i} f = \partial_{x_i} L(x, U(x), P(x)), \quad 1 \leq i \leq n,$$

ou encore

$$\partial_{x_i} f = - \sum_{k=n+1}^{N} \partial_{x_i} \varphi_k(x_{S_k})\, p_k, \quad 1 \leq i \leq n.$$

Comme dans la méthode du Lagrangien en optimisation de forme, le temps de calcul est indépendant du nombre de variables indépendantes, mais en contre partie la taille du vecteur $P(x)$ est égale au nombre d'instructions de l'algorithme de calcul de f. Dans le paragraphe suivant, nous donnons des indications sur la complexité des algorithmes de DA.

3.4. Complexité des algorithmes de DA

Dans les tableaux comparatifs entre la complexité des algorithmes de DA et ceux d'optimisation de forme nous utilisons les notations :

$f : D(f) \subset \mathbb{R}^n \longrightarrow \mathbb{R}$: une fonction à valeur réelle,

$\nabla f \in \mathbb{R}^n$: le gradient de f,

$F : D(F) \subset \mathbb{R}^n \longrightarrow \mathbb{R}^m$: une fonction à valeur vectorielle,

$DF \in \mathbb{R}^{m \times n}$: la Jacobienne de F,

$HF = D(DF)$: la Hessienne de F,

$y \in \mathbb{R}^n$: une direction de dérivation,

$u \in \mathbb{R}^m$: une direction de l'espace d'arrivée $\mathbb{R}^m$,

$L(\ldots)$: nombre d'opérations de base nécessaires pour calculer $(\ldots)$,

$S(\ldots)$: espace mémoire nécessaire pour calculer $(\ldots)$.

3.4.1. Complexité dans le calcul du gradient en optimisation de forme

Pour calculer la différentielle de la fonction coût nous devons, quelle que soit la méthode utilisée, calculer le second membre d'un système linéaire puis résoudre ce système et calculer une différentielle. Le coût principal, celui de la résolution du système linéaire, est en général le même avec la méthode directe ou avec celle du Lagrangien. Par contre le nombre de systèmes diffère suivant la méthode employée. Quelques résultats classiques sont rappelés dans le tableau 1.

	méthode directe	méthode adjointe
$L(f, \nabla f)$ $L(f, \nabla f, H)$	$O(nL(f))$ $O(n^2 L(f))$	$O(L(f))$ $O(n\,L(f))$
$S(f, \nabla f)$ $S(f, \nabla f, H.y)$ $S(f, \nabla f, H)$	$O(S(f))$	$O(S(f))$
$L(F, DF)$ $L(F, u^T DF)$ $L(F, DF.y)$	$O(nL(F))$ $O(nL(F))$ $O(L(F))$	$O(m\,L(F))$ $O(L(F))$ $O(mL(F))$
$S(F, DF)$ $S(F, u^T DF)$ $S(F, DF.y)$	$O(S(F))$	$O(S(F))$

TAB. 1 – *La complexité en optimisation de forme*

3.4.2. Complexité des algorithmes de DA

Les résultats essentiels, empruntés à K.Kubota [Kub 88], sont présentés dans le tableau 2 ; nous pouvons remarquer la similitude qui existe entre les résultats des deux tableaux, mais surtout noter que le mode inverse est celui à utiliser quand il y a un grand nombre de variables indépendantes.

	mode direct	mode inverse
$L(f, \nabla f)$ $L(f, \nabla f, H.y)$ $L(f, \nabla f, H)$	$\leq 4nL(f)$ $O(n\,L(f))$ $O(n^2 L(f))$	$\leq 4L(f)$ $\leq 14\,L(f)$ $\leq (10n+4)\,L(f)$
$S(f, \nabla f)$ $S(f, \nabla f, H.y)$ $S(f, \nabla f, H)$	$O(S(f))$	$O(S(f)+L(f))$
$L(F, DF)$ $L(F, u^T DF)$ $L(F, DF.y)$	$O(nL(F))$ $O(nL(F))$ $O(L(F))$	$\leq (3m+1)\,L(F)$ $\leq 4L(F)$ $\leq 4mL(F)$
$S(F, DF)$ $S(F, u^T DF)$ $S(F, DF.y)$	$O(S(F))$	$O(S(F)+L(F))$

TAB. 2 – *La complexité en DA*

4. Résultats numériques

4.1. Le logiciel ADOFEM

Les résultats numériques présentés ici ont été obtenus par ADOFEM®, logiciel de calcul de structures paramétré en statique linéaire utilisant la méthode des éléments finis, et qui s'appuie sur la méthode des dérivées d'ordre élevé.

4.2. Le cas d'une bielle

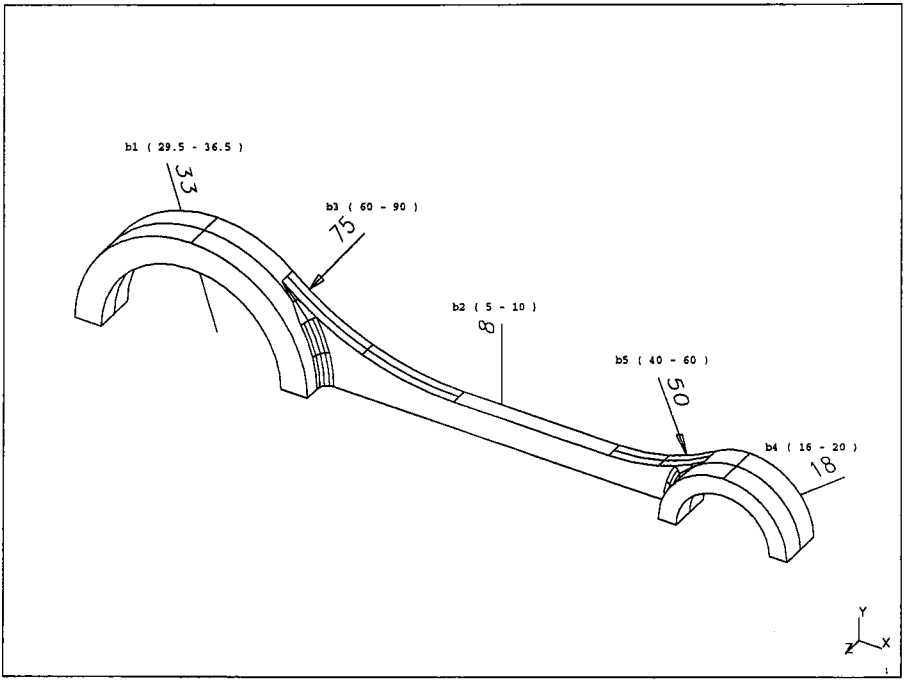

FIG. 3 – *Géométrie paramétrée*

Il s'agit d'une structure en acier (Module de Young : 2.068 10^{11} Pascal, coefficient de Poisson : 0.29), dont la géométrie est définie par la figure 3. Elle dépend de 5 paramètres dont l'intervalle de variation est indiqué sur la figure.

Cette structure est soumise à un chargement réparti comme indiqué sur la figure 4 : en chaque noeud où elle est appliquée, les composantes de la force sont (10000., 10000., 0.) Newton. A l'autre extrémité de la bielle, on impose aux noeuds situés sur le cylindre de plus petit rayon de rester sur ce cylindre.

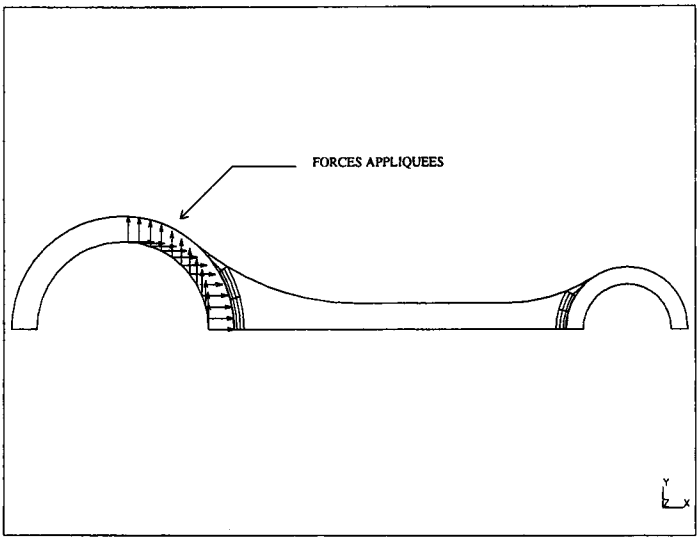

FIG. 4 – *Forces appliquées*

La géométrie et les conditions aux limites choisies permettent de réduire le domaine de calcul au quart de la bielle (2 plans de symétrie). Cela se traduit par des restrictions sur les déplacements des noeuds situés sur ces plans de symétrie.

Des éléments finis tétraédriques de type P_1 (3721 éléments correspondant à 1124 neuds, chaque noeud ayant 3 degrés de liberté) ont été utilisés pour traiter ce problème. Le système linéaire à résoudre est de taille légèrement supérieure à 3000.

Les ordres de dérivation par rapport aux paramètres b_i, $i = 1, 5$ sont respectivement $(5, 3, 1, 5, 0)$ et ce pour une précision exigée sur les déplacements de 1% (il s'est donc avéré que le paramètre b_5 n'avait aucun effet pour la précision demandée).

Il est impossible de visualiser les champs de contraintes et de déplacement dans un espace à cinq dimensions. On se limite donc à la visualisation du maximum de la contrainte de Von Mises sur toute la structure, exprimé en fonction de un ou deux paramètres.

Un seul calcul par éléments finis a été effectué en se plaçant en position

moyenne pour chacun des paramètres. Le maillage correspondant, construit par I-DEAS, est représenté sur la figure 5. Les maillages correspondants aux valeurs extrêmes des paramètres, construits par le logiciel ADOFEM et utilisés pour le calcul des dérivées, sont représentés sur la figure 6. Ces maillages ont la même topologie que le maillage initial.

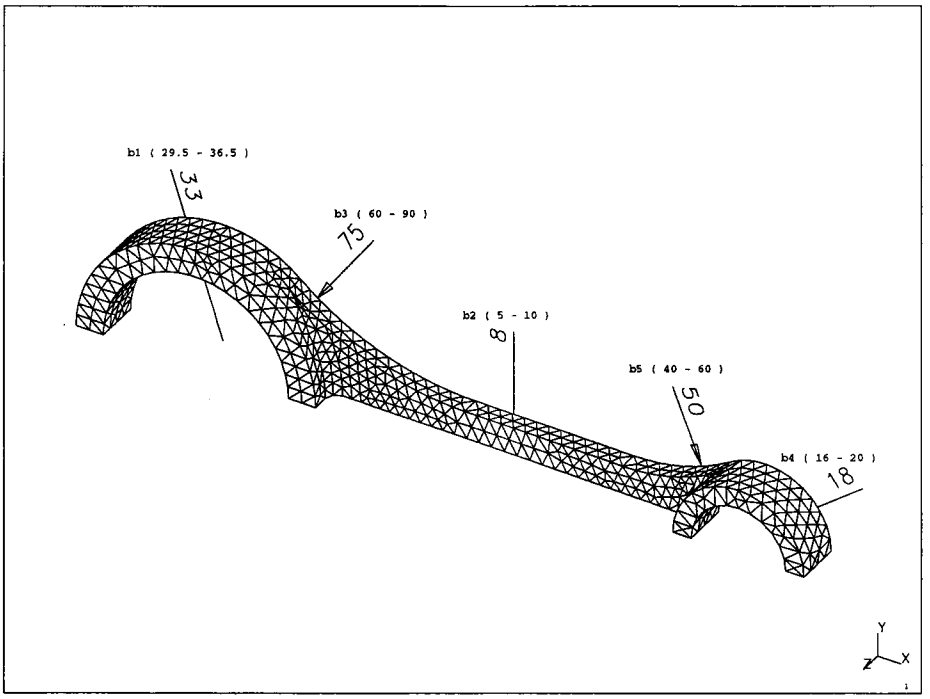

FIG. 5 – *Maillage par I-DEAS pour les valeurs moyennes des paramètres*

Le déplacement élastique et la contrainte de Von Mises sont paramétrés (polynôme de Taylor). Le maximum de la contrainte de Von Mises n'étant pas différentiable, la représentation polynomiale ne peut être utilisée que pour évaluer celle-ci en chaque point du domaine: le maximum est alors déterminé par un nouveau calcul pour chaque spécification des paramètres. On peut remarquer cette non différentiabilité sur les figures 7, 8, 9.

Dans le cas présenté, les résultats en masse et en contraintes sont normés à 1 pour la configuration initiale (correspondant à la valeur moyenne des paramètres).

La figure 7 montre la très bonne adéquation entre les résultats donnés par la

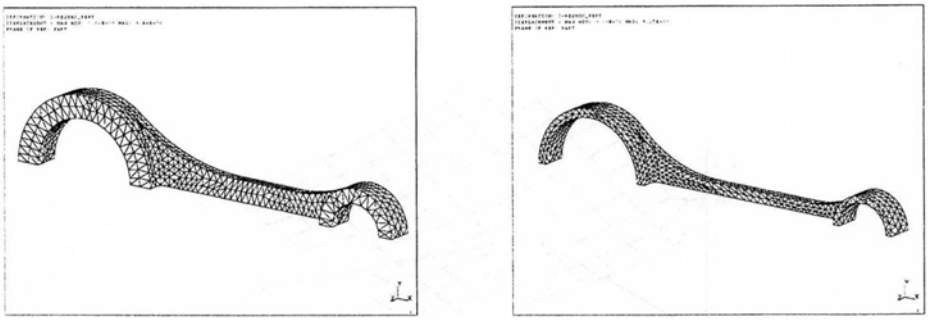

FIG. 6 – *Maillages perturbés par ADOFEM (paramètres à leurs valeurs extrêmes)*

paramétrisation et les résultats obtenus par 5 calculs directs (5 nouveaux calculs pour 5 valeurs du paramètre b1). Naturellement l'erreur sur les contraintes est bien supérieure à celle exigée sur les déplacements.

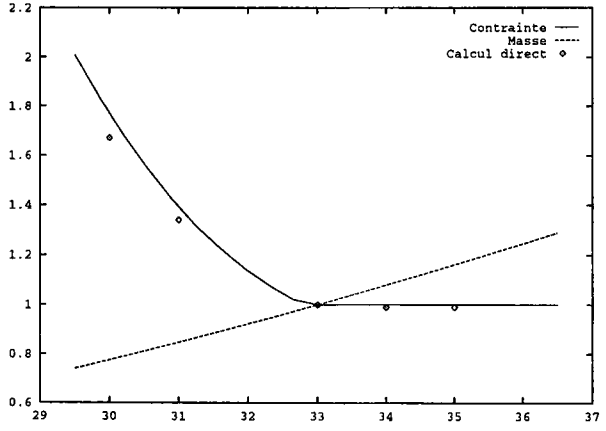

FIG. 7 – *Évolution du max de la contrainte de Von Mises et de la masse en fonction du paramètre b1*

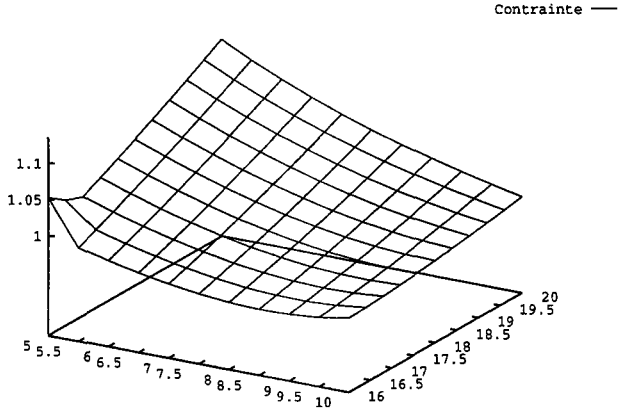

FIG. 8 – *Évolution du max de la contrainte de Von Mises en fonction des paramètres b2 et b4*

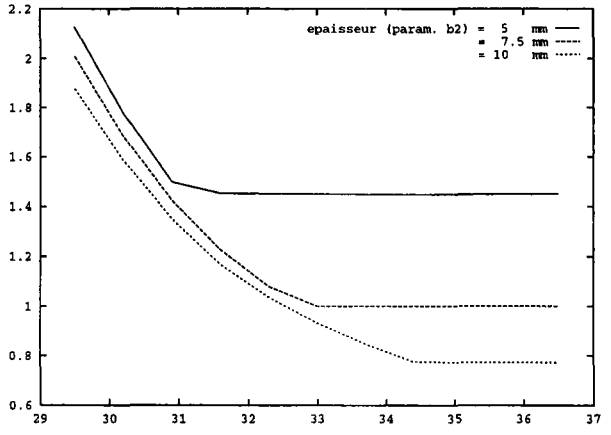

FIG. 9 – *Évolution du max de la contrainte de Von Mises en fonction du rayon (paramètre b1) pour différentes épaisseurs (paramètre b2)*

4.3. Un problème de coque

Il s'agit d'une structure en matériau plastique (Module de Young: 5 10^9 Pascal, Coefficient de Poisson: 0.35), dont la géométrie est définie par la figure 10. Elle dépend de 6 paramètres dont 3 paramètres de forme L_1, L_2, L_3 (lar-

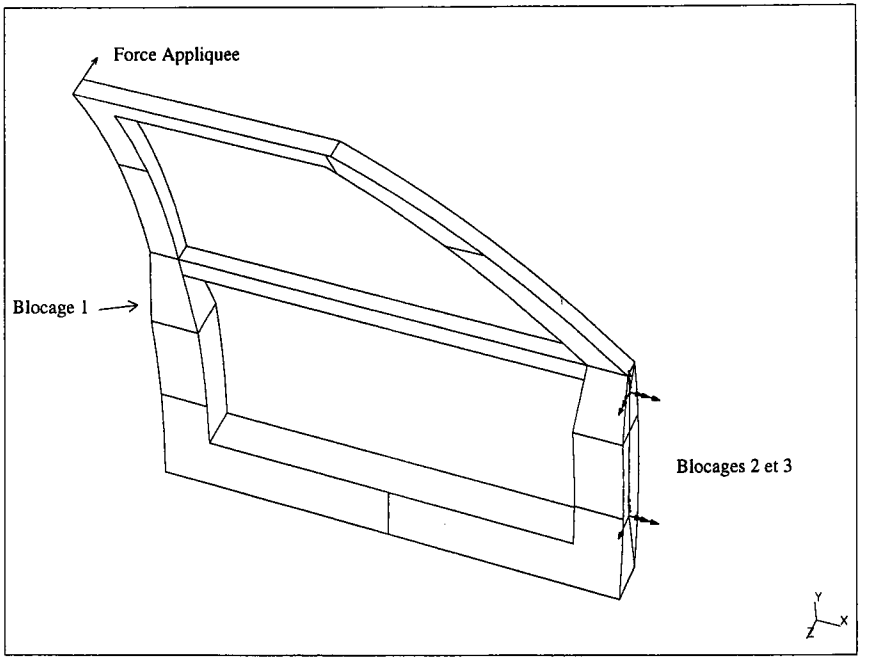

FIG. 10 – *Conditions aux limites*

geurs variables, cf. figure 12), 2 paramètres d'épaisseur e_1 (resp. e_2) pour la partie inférieure (resp. supérieure) de la géométrie et m le module d'Young. Les maillages correspondants aux valeurs extrêmes des paramètres de forme sont représentés sur la figure 12. Ici aussi, ils ont la même topologie que le maillage initial.

Les conditions aux limites de la structure sont données par la figure 10 qui indique que la porte est bloquée en trois points (charnières et verrou) et qu'elle subit un chargement appliqué sur la partie supérieure perpendiculairement à la surface.

Nous sommes intéressés par le déplacement de la porte qui est à l'origine d'infiltrations d'air.

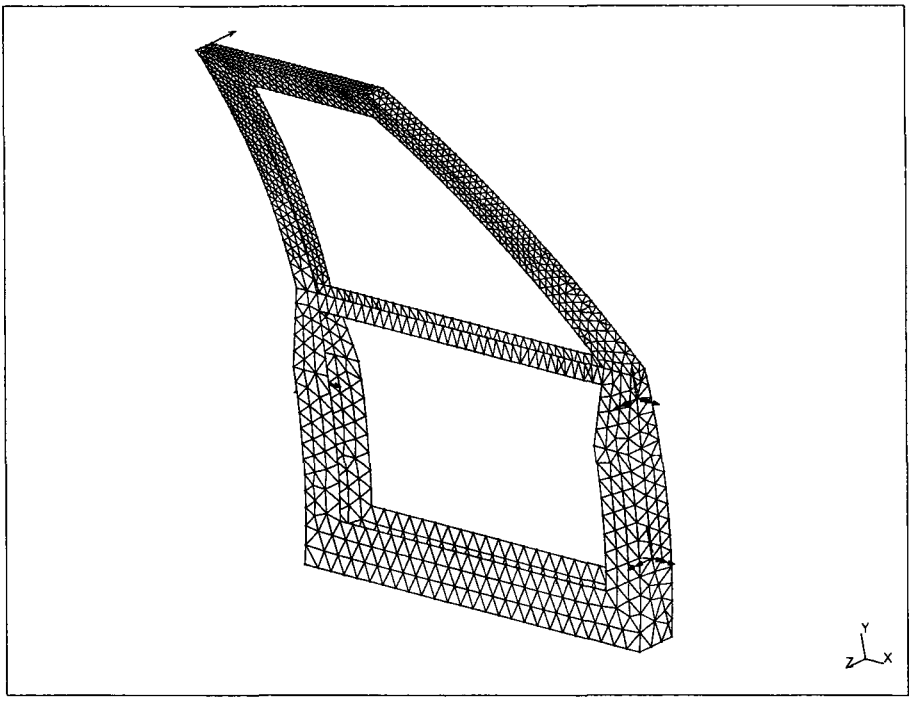

FIG. 11 – *Maillage par I-DEAS pour les valeurs moyennes des paramètres*

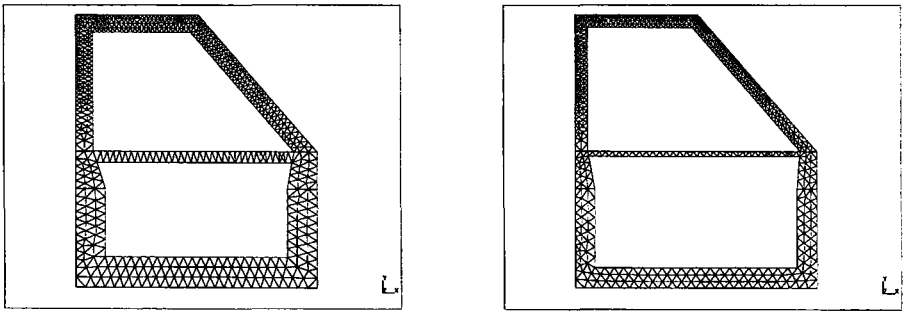

FIG. 12 – *Maillages perturbés par ADOFEM (paramètres de forme à leurs valeurs extrêmes)*

On a utilisé 3298 éléments *DKT* [BaD 92], associés à 1841 noeuds ayant chacun six degrés de liberté.

L'erreur exigée par l'utilisateur est de 3% ; elle a donné lieu à un développement de Taylor à des ordres allant de 5 à 8 suivant les variables. Le temps CPU d'une simple analyse est d'environ 40 secondes, et le temps CPU de l'analyse complète avec calcul des dérivées est d'environ 3000 secondes. Ce temps de calcul peut sembler important. Cependant, il y a ici 6 paramètres. Si par exemple on avait fait des analyses directes avec trois valeurs distinctes pour chacun de ces paramètres, cela aurait représenté en tout $3^6 = 729$ analyses avec un temps de calcul de $729 \times 40 = 29160$ secondes, soit 10 fois plus qu'en utilisant le calcul des dérivées. De plus, l'information obtenue par le calcul des dérivées porte sur des intervalles complets et non sur 3 valeurs pour chaque paramètre.

L'unique polynôme de Taylor est exploité pour calculer le déplacement de la porte en fonction de la variation de différents paramètres (figures 13, 14 et 15). On peut constater la non différentiabilité du maximum de déplacement en fonction de l'épaisseur sur la figure 15.

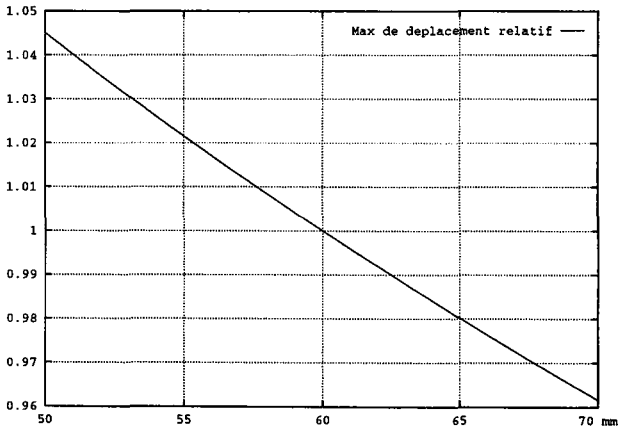

FIG. 13 – *Variation du déplacement en fonction du paramètre* L_1

5. Conclusion

Nous avons montré l'efficacité de l'utilisation des dérivées d'ordre élevé pour la simulation numérique de problèmes rencontrés dans l'industrie. De nombreux développements restent à faire. Les problèmes non linéaires peuvent être abordés avec les mêmes techniques, ainsi que les problèmes d'évolution. Nous n'avons pas abordé ici l'utilisation des approximants de Padé. De récentes études nous ont

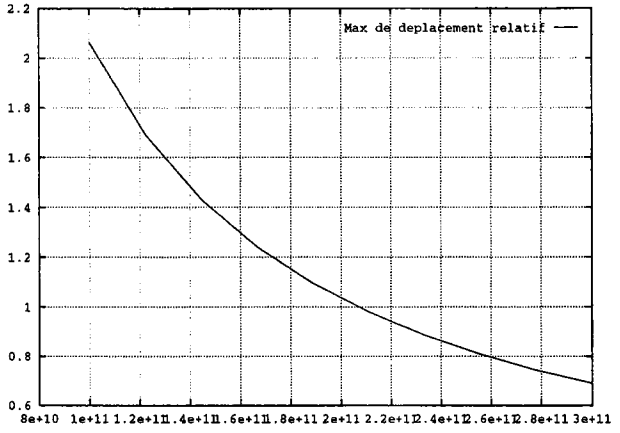

FIG. 14 – *Variation du déplacement en fonction du module d'Young*

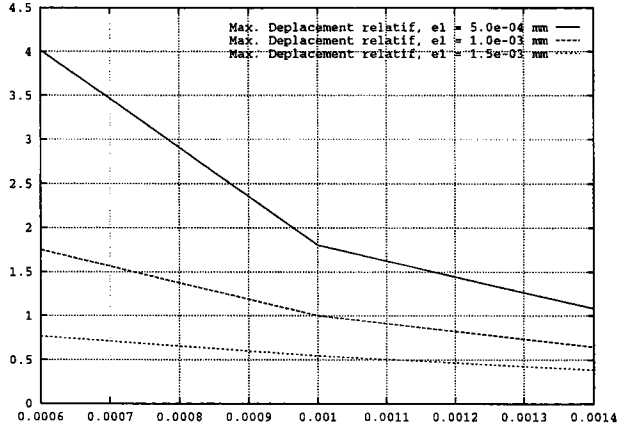

FIG. 15 – *Variation du maximum de déplacement en fonction de l'épaisseur e_2 pour différentes valeurs de e_1*

montré que leur utilisation permet d'approcher la solution sur un domaine bien plus important que celui associé à un polynôme de Taylor. C'est particulièrement le cas en électromagnétisme (présence de pôles) ou en dynamique. De plus, cela fournit un excellent outil de recherche de modes dans les problèmes linéaires ou non linéaires.

Bibliographie

[**BaD 92**] J. L. BATOZ, G. DHATT, *Modélisation des structures par éléments finis*, Hermès, Paris, 1992.

[**BaS 83**] W. BAUR, V. STRASSEN *The Complexity of Partial Derivatives*, Theoretical Computer Science, 22 (1983), 317-330.

[**Cea 86**] J. CEA, *Conception optimale ou identification de formes, calcul rapide de la dérivée directionnelle de la fonction coût*, M.A.A.N, 20 (1986).

[**Des 76**] DESTUYNDER, *Thèse de troisième cycle*, Université PARIS VI.

[**Fuj 86**] N. FUJII, *Necessary Conditions for a Domain Optimization Problem in Elliptic Boundary Value Problems*, SIAM.J. on Control and Optimisation24 (1986), 346-360.

[**Gil 92**] J.C. GILBERT, *Automatic Differentiation and Iterative Processes*, Optimization Methods and Softwares, 1 (1992), 13-21.

[**GVM 91**] J.C. GILBERT, G. Le VEY, J. MASSE *La differentiation automatique de fonctions représentées par des programmes*, INRIA, Rapports de recherche, No 1557, 1991.

[**GJS 90**] A. GRIEWANK, D. JUEDES, J. SRINIVASAN, C. TYNER, *ADOL-C, a Package for the Automatic Aifferentiation of Algorithms Written in C/C++*, ACM Trans. Math. Software.

[**Gri 89**] A. GRIEWANK, *On Automatic Differentiation*, In M. Iri and K. Tanabe (editors), Mathematical Programming : Recent Developments and Applications, Kluwer Academic Publishers, Dordrecht, 1989, 83-108.

[**Gui 94**] Ph. GUILLAUME, *Dérivées d'ordre supérieur en conception optimale de formes*, Thèse de l'Université Paul Sabatier, 1994.

[**GuM 92**] Ph. GUILLAUME, M. MASMOUDI, *Dérivées d'ordre supérieur en optimisation de domaines*, C.R. Acad. Sci. Paris, t.315, Série I (1992), 859-862.

[**GuM 93**] Ph. GUILLAUME, M. MASMOUDI, *Calcul numérique des dérivées d'ordre supérieur en conception optimale de forme*, C.R. Acad. Sci. Paris, t.316, série 1 (1993), 1091-1096.

[**GuM 94**] Ph. GUILLAUME, M. MASMOUDI, *Computation of High Order Derivatives in Optimal Shape Design*, Numerische Mathematik, 67, 1994, 231-250.

[**GuM 95**] Ph. GUILLAUME, M. MASMOUDI, *Solution to the Time-Harmonic Maxwell's Equations in a Waveguide, Use of Higher Order Derivatives for Solving the Discrete Problem*, to appear in SIAM Journal on Numerical Analysis.

[**Hil 85**] HILLSTROM, *User Guide for Jakef. Technical Memorandum*, ANL/MCS TM-16, Argone National Laboratory, Argone, Il 60439, 1985.

[KaL 91] D. KALMAN, R. LINDELL, *Automatic Differentiation in Astrodynamical Modeling*, Automatic Differentiation of Algorithms, A. Griewank, G.F. Corliss (editors), SIAM, 1991.

[Ken 83] KENNAN T. SMITH, *Primer of Modern Analysis*, Springer Verlag, 1983.

[Kub 88] K. KUBOTA, *A preprocessor for Fast Automatic Differentiation - Applications and Difficulties on Practical Problems*, in RIMS Kokyuroku 648 "fundamental Numerical Algorithms and their Software", 1988.

[Lay 91] J.D. LAYNE, *Applying Automatic Differentiation and Self-Validation Numerical Methods in Satellite Simulations*, Automatic Differentiation of Algorithms, A. Griewank, G.F. Corliss (editors), SIAM, 1991.

[MaF 92] T. MASANAO, N. FUJII, *Second Order Necessary Conditions for Domain Optimization Problems in Elastic Structures*, Journal of Optimization Theory and Applications, 72 (1992).

[Mas 87] M. MASMOUDI, *Outils pour la conception optimale de formes*, Thèse d'état, Nice, 1987.

[MRP 95] M. MASMOUDI, C. RAUZY-MASSAT, R. POTEAU, *Automatic Differentiation and Global Optimization*, to appear in Applied Mathematics and Computer Sciences.

[Mor 85] J. MORGENSTERN, *How to compute fast a function and all its derivatives, a variation on the theorem of Baur-Strassen*, SIGACT News, 1985.

[MuS 76] F. MURAT, J. SIMON, *Sur le contrôle par un domaine géométrique*, Thèse d'état, Paris, 1976.

[NBC 89] F. NAVARRINA, E. BENDITO, M. CASTELEIRO, *High Order Sensitivity in Shape Optimization Problems*, Computer Methods in Applied Mechanics and Engineering, North-Holland, 75 (1989), 267-281.

[OWB 71] G.M. OSTROVSKII, J.M. VOLIN, W.W. BORISOV, *Uber die Berechnung von Ableitungen*, Wissenschaftliche Zeitschcrift der tecnischen Hochschule fur Chemie, Leuna Merseburg, 13 (1971), 382-384.

[Roc 94] M. ROCHETTE, *Le manuel d'utilisation d'Adogen*, (1994).

[RoD 92] N. ROSTAING, S. DALMAS, *Automatic Differentiation Analysis and Transformation of Fortran Program Using a Typed Functional Language* International Conference on Computing Methods in Applied Sciences and Engineering, Feb. 11-14, 1992, Paris.

[Rou 87] B. ROUSSELET, *Shape Design Sensitivity From Partial Differential Equation to Implementation*, Eng. Opt. 11 (1987), 151-171.

[Sim 80] J. SIMON, *Differentiation With Respect to the Domain in Boundary Value Problems*, Numerical Functional Analysis and Optimization, 2 (1980), 649-687.

[Sim 89] J. SIMON, *Second Variation for Domain Optimization Problems*, International Series of Numerical Mathematics n°91, Birkhauser, 1989.

[Spe 80] B. SPEELPENNING, *Computing Fast Partial Derivatives of Functions Given by Algorithms*, PhD thesis, Department of Computer Science, University of Illinois at Urbana-Champain, Urbana-Champain,IL 61801, 1980.

[Str 90] V. STRASSEN, *Algebraic Complexity Theory*, in J. van Leeuwen (Editeur), Handbook of Theoritical Computer Science, volume A : Algorithms and Complexity, Elsevier, Amsterdam, 1990.

[Wen 64] R.E. WENGERT, *A Simple Automatic Derivation Evaluation Program* Comm. ACM, 7 (1964), 463-464.